Kalle Lyytinen
Pericles Loucopoulos
John Mylopoulos
Bill Robinson (Eds.)

Design Requirements Engineering: A Ten-Year Perspective

**Design Requirements Workshop
Cleveland, OH, USA, June 2007
Revised and Invited Papers**

Springer

Lecture Notes
in Business Information Processing   14

Series Editors

Wil van der Aalst
  Eindhoven Technical University, The Netherlands
John Mylopoulos
  University of Trento, Italy
Norman M. Sadeh
  Carnegie Mellon University, Pittsburgh, PA, USA
Michael J. Shaw
  University of Illinois, Urbana-Champaign, IL, USA
Clemens Szyperski
  Microsoft Research, Redmond, WA, USA

Kalle Lyytinen   Pericles Loucopoulos
John Mylopoulos   Bill Robinson (Eds.)

Design Requirements Engineering:
A Ten-Year Perspective

Design Requirements Workshop
Cleveland, OH, USA, June 3-6, 2007
Revised and Invited Papers

Springer

Volume Editors

Kalle Lyytinen
Case Western Reserve University, Cleveland, OH, USA
E-mail: kalle@case.edu

Pericles Loucopoulos
Loughborough University, UK
E-mail: P.Loucopoulos@lboro.uk

John Mylopoulos
University of Toronto, ON, Canada
E-mail: jm@cs.toronto.edu

Bill Robinson
Georgia State University, Atlanta, GA, USA
E-mail: wrobinson@cis.gsu.edu

Foreword

The critical nature of requirements was recognized around the time that the term *software engineering* was being coined to signal the need for a new discipline [NATO Software Engineering Conference, 1968]. In those early years, the focus of requirements practice was on the requirements document, which was the basis of a contract between customer and developer, which needed to be sufficiently definitive to determine, before system construction, what needed to be built, how much it might cost, and how long it might take to complete and put into operation. Requirements tended to change (or "creep") until arbitrarily "frozen" so that agreement could finally be reached and the development project could begin. When the development project was over, the resulting system was evaluated to see if the contract was fulfilled. The approach is very disciplined, but the reality is that it has been bent and is broken.

As software companies and IT departments grew, it was obvious that requirements were a major weak link in the business of software. Early studies and experience reports tried to put their finger on the problem, stating that requirements were often ambiguous, inconsistent, incomplete, missing, or just plain wrong. This recognition led to the emergence of requirements languages, tools, and techniques, as reported in workshops such as the International Workshop on Software Specifications & Design [IWSSD] and elsewhere [TSE Jan. 1777 special issue on requirements]. Eventually, a coalescence of people from academia, industry, and business formed a requirements engineering (RE) community, as manifested in two conference series [IEEE International Conference on Requirements Engineering (ICRE) and IEEE International Symposium on Requirements Engineering (ISRE)] (which merged in 2002), a journal [*Requirements Engineering*], an IFIP Working group [WG 2.9], and some spin-offs.

Over the years, the evolution of issues and ideas of the young field of RE has been paralleled, and challenged, by the amazingly quick development and penetration of software-intensive systems, especially since the emergence of personal computers, the widespread availability of the internet, and the World Wide Web. Systems are no longer assumed to be built from scratch or designed under the control of a single developer. It is no longer assumed that all requirements will be known in advance of building the system. It is now assumed, instead, that requirements will continue to change as time goes on, that design decisions will be made in response to new knowledge and understanding of requirements. It is no longer assumed that the software will be fully verified before it goes into operation, implying that some requirements will have to be monitored, checked, and reconsidered even after being deployed into operation. (Of course, safety-critical and other high-risk requirements will need more upfront consideration.)

What do the new assumptions and their ramifications, barely touched upon here, imply for the future of requirements and design for software-intensive systems? The papers in this book explore and articulate a rich set of issues and ideas. We need to learn more and be more explicit about questions such as: Where do the requirements come from that designers seek to satisfy? What do designers do when they design?

What knowledge is needed by designers, and how can it be made available to them when needed? In particular, how can requirements be captured so that they are available throughout the system lifecycle? What does it mean for a design to satisfy a requirement? How are multiple requirements from multiple stakeholders to be negotiated and evolved, perhaps frequently and continuously?

It is time to rethink the role of requirements and design for software-intensive systems, which are seeping into all corners of our lives and taking part in, or taking over, many aspects of everyday activities. In transportation, health care, banking, shopping, and government – just to mention a few domains where software-intensive systems play prominent and growing roles – software does much more than crunch numbers, maintain and transfer data, mechanize paperwork, and control machinery. As networks of computers and people grow in size and programmed functionality, our systems now implement end-to-end consumer services and organizational processes, institutional policies, and societal infrastructure. These systems entail requirements that are richer, more complex, and more elusive than ever, and designing to meet these requirements in our evolving socio-technical environment poses a plethora of new challenges.

To address the increasing scope and pervasiveness of today's software-intensive systems, and to address the concomitant challenges, we need to take stock and branch out. We need to question past assumptions and rethink research, practice, and education.

This book brings together an energetic and highly motivated, international community of researchers to assess the current state of affairs and define new directions. Recognizing the need to draw on a broad array of intersecting areas of study – software engineering, design research, human-centered computing, business management, industrial design, and an open-ended set of others – the authors have led a series of community-building events to bring together researchers with diverse backgrounds, research methodologies, and perspectives. This book is a tangible result of the synergies created by working across traditional boundaries.

September 2008   Sol Greenspan
                                                                      Program Director
                                                        Division of Computing and
                                                        Communication Foundations
                                                        National Science Foundation

Organization

Volume Editors

Kalle Lyytinen is the Iris S. Wolstein Professor at Case Western Reserve University (USA), Adjunct Professor at the University of Jyvaskyla (Finland), and visiting professor at the University of Loughborough (UK). He serves currently on the editorial boards of several leading information systems and requirements engineering journals including *Journal of AIS (*Editor-in-Chief*), Journal of Strategic Information Systems, Information&Organization, Requirements Engineering Journal, Information Systems Journal, Scandinavian Journal of Information Systems, and Information Technology and People,* among others. He is an AIS fellow (2004), the former chairperson of the IFIP 8.2 Working Group, and a founding member of SIGSAND. Lyytinen led the research team that developed and implemented MetaEdit+, the leading domain modeling and metaCASE platform globally. He has published over 180 scientific articles and conference papers and edited or written 11 books on topics related to the nature of the IS discipline, system design, method engineering, organizational implementation, risk assessment, computer supported cooperative work, standardization, and ubiquitous computing. He is currently involved in research projects that looks at IT-induced radical innovation in software development, IT innovation in architecture, engineering and construction industry, requirements discovery and modeling for large-scale systems, and the adoption of broadband wireless services in the UK, South Korea, and the USA.

Pericles Loucopoulos is Professor of Information Systems in the Business School, Loughborough University (UK) and adjunct professor at the University of Manchester. He is the Co-editor-in-Chief of the *Journal of Requirements Engineering*, Associate Editor for *Information Systems* and serves on the editorial boards of the *Information Systems Journal,* the *Journal of Computer Research*, *Business Process Management Journal*, *International Journal of Computer Science & Applications*, the *International Journal of Computing and ICT Research*, among others. Loucopoulos is a Fellow of the British Computer Society and serves on the European Advisory Board of the AIS SIG on Systems Analysis and Design. He has served as General Chair and Programme Chair of six international conferences and has been a member of over 150 Programme Committees of international conferences. He has published over 150 papers in academic journals and conference proceedings

on the engineering of information, and the tools, methods, and processes used to design, develop, and deploy information systems in order to meet organizational goals. He has written or edited 10 books related to requirements and information systems engineering.

John Mylopoulos is Professor of Computer Science at the University of Toronto (Canada) and Professor (chiara fama) in the Department of Information Technology and Telecommunications, University of Trento (Italy). Mylopoulos is Co-editor-in-Chief of the *Journal of Requirements Engineering*, and *Lecture Notes in Business Information Processing*, both published by Springer. He is a fellow of the American Association for Artificial Intelligence (AAAI), fellow of the Royal Society of Canada (Academy of Sciences), past president of the Very Large Databases (VLDB) Endowment (1998-2003), and co-recipient of the most influential paper award at ICSE 1994 (with Sol Greenspan and Alex Borgida). He has served as Programme/General Chair for international conferences in AI, Databases and Software Engineering, including the International Joint Conference on AI (1991, Sydney), the VLDB Conference (2004, Toronto), the IEEE International Conference on Requirements Engineering (1997, Annapolis) and the International Semantic Web Conference (2003, Sanibel Island FL)

Bill Robinson is an Associate Professor in the Department of Computer Information Systems and Associate of the Center for Process Innovation (CEPRIN), J. Mack Robinson College of Business at Georgia State University. Requirements and policies of information systems are the focus of his research, which includes analyses of supply-chain policies, requirements interactions and negotiations, and runtime requirements monitoring. He directs the Requirements Analysis and Monitoring Lab (RAML), supported by the National Science Foundation and industry. The lab has produced REQMON, a requirements monitoring extension to Eclipse TPTP. Dr. Robinson has authored over 60 publications, including ACM, IEEE, Springer, and Taylor & Francis transactions and magazines. He served the Requirements Engineering community as RE 1999 program chair, RE 2009 General Chair, Co-editor-in-Chief of *Requirements Engineering*, and Secretary of International Federation for Information Processing Working Group (IFIP) 2.9 on Requirements Engineering. He holds degrees from Oregon State University (BS Computer Science) and the University of Oregon (MS and PhD Computer Information Systems).

In addition to the duties of the four editors of the volume, significant editorial assistance was provided by the following:

Sean Hansen is a PhD Candidate in the Department of Information Systems at Case Western Reserve University. His research interests are in requirements engineering, IT strategy, knowledge management, and the application of contemporary cognitive theory to information systems development. His current research focuses on the emergence of new approaches to requirements engineering in response to advances in systems design and implementation. Prior to embarking on a career in research, Sean consulted to a wide range of industries in the areas of management and technology. He has presented at conferences including the International Conference on Information Systems (ICIS), the Americas Conference on Information Systems (AMCIS), the Annual Meeting of the Academy of Management (AoM), and the

Administrative Sciences Association of Canada (ASAC). He is currently completing his dissertation focusing on the management of requirements in contemporary design environments.

Nicholas Berente is a research fellow and lecturer at the University of Michigan's School of Information. His research focuses on complex design processes and the role of information systems in supporting innovation. He has a Bachelor's degree in finance from John Carroll University and an MBA and PhD from Case Western Reserve University. He is the founder and former president of Pentagon Engineering Corporation, a nationwide systems integrator that specialized in the information systems that support product development, which he sold in 2002.

Table of Contents

Section 1: Fundamental Concepts of Design

Section 2: Evolution and the Fluidity of Design

Section 3: Quality and Value-Based Requirements

Section 4: Requirements Intertwining

Section 5: Adapting Requirements Practices in Different Domains

High Impact Design Requirements - Key Design Challenges for the Next Decade*

Nicholas Berente[1], Sean Hansen[2], and Kalle Lyytinen[2]

[1] University of Michigan, School of Information, 1085 South University, Ann Arbor,
Michigan 48109, USA
`berente@umich.edu`
[2] Case Western Reserve University, Weatherhead School of Management, Department of
Information Systems, Cleveland, Ohio 44106, USA
`kalle@case.edu, sean.hansen@case.edu`

Abstract. The High Impact Design Requirements Project involves an international, cross-disciplinary group of researchers and practitioners focused on setting the agenda for the design requirements discourse. The group's initial workshop identified five key themes that should receive more attention in design requirements and practice: (1) fundamental concepts of design, (2) evolution and the fluidity of design, (3) quality and value-based requirements, (4) requirements intertwining, and (5) adapting requirements practices in different domains. This chapter presents an introduction to the project, the workshop, and these five themes.

1 Introduction

"Future software-intensive systems will be vastly different from those in use today. Revolutionary advances in hardware, networking, and human interface technologies will require entirely new ways of thinking about how software systems are conceptualized, built, understood, and evaluated. As we envision the future of complex distributed computing environments, innovative research is needed to provide the scientific foundations for managing issues of complexity, quality, cost, and human intellectual control of software design and development." (U.S. National Science Foundation[1])

The National Science Foundation's Science of Design website asserts that design practices for software-intensive systems are departing dramatically from the accepted methods of the past. This also is consistent with the views of many researchers and practitioners of today (e.g., see Hansen et al. [1] in this volume). If and when this is the case, it is imperative that researchers understand and seek to shape and improve new emerging design practices. The first step in designing any system involves the

---

[1] Taken from the National Science Foundation's "Science of Design" website, July 28, 2008: http://www.nsf.gov/funding/pgm_summ.jsp?pims_id=12766

generation of what are often referred to as "requirements" that state the system's functions, properties, design constraints, and the their rationale. Therefore, a firm understanding of requirements-related practices is fundamental in shaping and improving the future of system design in general.

With this objective in mind, leading researchers and practitioners from around the world came together with the goal of setting an agenda for future design requirements research and how to improve requirements practice in what has come to be called the "High Impact Design Requirements Project." The goal of this project is to broadly identify and articulate emerging trends in requirements practices and to elicit principles and directions for future research that will have significance for the practice of creating software-intensive systems. This is not a straightforward task, however, as the topic of "requirements" straddles a variety of disciplines and perspectives, has a long and varied history, and, as a number researchers have recently indicated (i.e., [2], [3], [4]), is experiencing some level of misalignment between research and practice.

To give the issue of requirements thorough and rigorous attention, the High Impact Design Requirements Project involves a multi-year effort focused on building a robust interdisciplinary research community around the topic through a series of workshops. This book represents the output of the first in this series of workshops, and it is intended to help researchers understand the current state of research and practice, as well as to elaborate on key themes identified in the workshop that are at the center of emergent practices and related principles.

In this chapter, we will briefly introduce the High Impact Design Requirements Project, describe the structure and process associated with the first workshop, and identify the key themes from the first workshop that inform the organization of this book and are expected to be critical, salient issues in future requirements engineering research. We conclude with a brief reflection on the prospects of the project.

2   The High Impact Design Requirements Project

The High Impact Design Requirements Project involves an interdisciplinary community of researchers and practitioners who focus on the determination and management of requirements in contemporary design efforts. Funded in part by the National Science Foundation's Science of Design Program (Grant Number: CCF0613606), the project is motivated by the perception that newer design environments and related requirements engineering processes are marked by a range of challenges and opportunities that are insufficiently recognized in the prevailing research literature and practice (methods, tools etc). Furthermore, the derivation and management of design requirements are themes common to many separate research and design science communities, including those focusing on design methods and theory, software architectures, human computer interaction, formal specification methods, verification principles, information systems research, industrial design, architecture, organization design, and media design. Despite shared concerns and interests, these research communities have had little exchange of ideas across disciplinary boundaries. The absence of interdisciplinary intellectual commerce is particularly discouraging given the growing need for a diversity of skills and insights in the design of software intensive systems. Modern software design increasingly involves aspects of industrial design (e.g., pervasive applications), media design (e.g., e-commerce and media

applications), human computer interaction (e.g., new modalities of interaction), and business architectures (e.g., open business platforms), just to name a few grand challenges. Herein we describe the initial phase of a community building effort intended to begin rectifying this situation by improving interdisciplinary exchange.

The core of the community building effort involves the planning and organization of a series of workshops, designed to accomplish three broad objectives: (1) engage separate research communities in a dialogue, (2) strengthen design science principles, and (3) open new vistas for the field of design requirements associated with software intensive systems that will meet the current and future needs of practice. The workshops are intended to reflect the truly inclusive focus of the project. This community-building effort is unique in that it draws thought leaders from a wide array of organizational environments and research disciplines, including such areas as design science, information systems, behavioral studies, software development, organizational design, and requirements modeling. The workshops address a range of application environments, such as e-business, pervasive computing, and large enterprise systems that deal with complex socio-technical designs.

The initial workshop was held in the United States in 2007 and the first follow-up workshop is being conducted in Europe in the Fall of 2008. In preparation for the workshops, the project team undertook a field study of practicing designers across a range of industrial and systems design settings to understand the perspectives of practitioners about prevailing requirements practices, anticipated developments in requirements practices, and emergent challenges encountered in contemporary design environments (see Chapter 3 of this book for a presentation of the field study). The themes identified in the field study were used to organize the discussion in the initial workshop. Next we will address the structure and processes associated with the first workshop.

3   Design Requirements Workshop

The first workshop on June 3-6, 2007, held at Case Western Reserve University in Cleveland, Ohio (USA) (see www.case.weatherhead/requirements) brought together 35 academics and design professionals from across the United States and Europe (see Appendix 1 for a list of participants and presenters). The workshop project team structured the three-day symposium to ensure a balance between full group discussions and more intensive breakout sessions focused on exploring specific issues and challenges. Plenary sessions initiated each day's conversations and established the basis for collaborative inquiry. Work groups of 8 to 10 participants convened in morning and afternoon sessions for targeted discussions of key trends in requirements theory and practice. Interdisciplinary panel discussions in the middle of each day's agenda enabled the entire group to delve into potential avenues of cross-fertilization of ideas and to share different perspectives. Each day closed with a plenary regroup to address questions and raise provocations that had emerged from the day's discussions.

The discourse from the workshop was captured a variety of ways – through webcasts, a workshop Wiki, and plenary presentations. Further, scribes assigned to each workgroup documented group discussions through a Wiki environment that was available to all participants for interactive editing and blogging. At the end of the workshop, each workgroup presented key themes of their discussions throughout the workshop to the entire group. Overall, the workshop involved five plenary

presentations, three panel discussions, eight plenary work group reports, and twenty-eight presentations within workgroups.[2]

In preparation for the workshop, each participant prepared a short discussion paper focusing on one or more facets of the cutting edge in theory and practice on requirements issues. Specific topics proposed for exploration included the following:

- The need for new avenues in requirements research – a new landscape and emergent challenges
- Developing a 10-year agenda for research in requirements engineering
- Opportunities for intellectual cross-fertilization across disciplines in design and requirements
- Analyses of contemporary requirements practice
- Perceived strengths of previous research and existing practice
- Perceived shortcomings of the requirements research traditions in different fields
- Theories and research methodologies pertinent to the emerging challenges associated with design requirements
- Case studies and practice reports of successful or failed practices and critical issues related to design of complex software-intensive systems

Building upon these initial perspectives, researchers were encouraged to reflect upon their own research interests and areas that they felt deserved greater attention in the coming years. The presentation and exploration of these various positions formed a basis for focused conversations during the workshop, and either directly evolved into the chapters of this book or inspired the inclusion of these chapters. Specific tracks for workgroup discussions were based on the findings of the preliminary field study. The workgroups included sessions on fundamental concepts of design, requirements evolution (both the fluidity of design and the management of changing requirements), visualization & representation, managing complexity, business process focus in requirements, stakeholder issues in requirements management and negotiation, and the impact of new technologies & architectures (see Chapter 3). Next, we will highlight some key themes generated during the workshop, and then briefly reflect on the outlook for the project going forward.

4   Highlights from the Workshop

The goal of the workshop was to trigger a dialog, and to use that dialog to generate key issues and themes for future research on requirements-related topics. Three tactics were employed to inspire this dialog: (1) the practitioner themes by which we organized the workshop (see Chapter 3); (2) position papers that participants prepared in advance, presented in the workgroup sessions, and made available through the website; and (3) plenary presentations by five diverse thought leaders. During the workshop, a number of themes recurred, in particular, many relating to the long-established and interrelated design challenges of complexity, uncertainty, emergence, and social

---

[2] For details about the workshop, see http://weatherhead.case.edu/requirements/; plenary presentations from Kalle Lyytinen, Fred Brooks, John King, Colin Potts, and Venkatesh Prasad are available at: http://tv.case.edu/actions/tv2/tv - search for "design requirements."

distribution. These challenges were met with requirements practices that involved iteration & evolution, architectures, risk management, and through a variety of methodological improvements in general.

Complexity, for example, is a topic that underpinned much of discussion, and represented a dialog that has a rich history in requirements research. As Fred Brooks indicated during his keynote presentation, complexity arises in part from the continuous changes to contexts, constraints, and functionality. Further, Brooks argued while much of past work on requirements implies the work of a single designer, current practice inevitably involves teams of distributed designers and large-scale system design has become more like an "interdisciplinary negotiation" than the design of a singular, stable system. In order to maintain conceptual integrity of a design process, Brooks argued that the importance of a solid architecture and the role of a master architect in managing complexity like in architecture or complex product designs cannot be understated. Participants characterized complexity a variety of ways, including by distinguishing between forms of complexity (King), by describing the evolving locus of complexity (Lyytinen), and questioning what we mean by complexity (Potts). Also, many offered a variety of solutions to such issues, such as stronger organizational risk assessment practices (Dunbar) and iteration in its various forms (Easterbrook). While the various perspectives on complexity reflected the wider discourse that has been taking place over the past two decades (see Chapters 3 & 4), a goal of this book is to look forward, and to contextualize many of these ideas in emerging trends and opportunities for relevant research as we look to the future.

As the purpose of this book is to focus primarily on the future, rather than attempting to be exhaustive of all potential trends and opportunities, we have endeavored to solicit themes that offer a departure from contemporary discourse in subject matter as well as in mindset. This does not mean that these themes are not related to (and are often interdependent with) the important and critical themes in current requirements thinking. Nor do we argue that these themes are all necessarily new. Instead, we hope that they can contribute to the foundation of a new platform for thinking about requirements research and practice that will take shape in the next decade. To this end we identified five broad topics that that are *under-addressed and insufficiently researched* in the current literature. We deemed these topics to be vital to research going forward as they, in one way or another, challenge some of the underlying beliefs and assumptions that underlie the majority of existing research.

These themes, in the order of the sections of this book, are (1) fundamental concepts of design, (2) evolution & the fluidity of design, (3) quality & value-based requirements, (4) requirements intertwining, and (5) adapting requirements practices in different domains. It is important to note that we do not promote these themes to be mutually exclusive nor as exhaustive of the critical emerging issues. Rather, we view them as some of the pillars in the emerging new research platform. The following sections will highlight why and how each is becoming increasingly important to the study of design requirements.

4.1 Fundamental Concepts of Design

"A point I want to emphasize in the requirements process is we do not usually know what the goal is. I will assert that that is a deep fact of reality that is ignored in much of the literature about requirements. We do not know what we

are trying to build. We cannot, in most cases, know what we are trying to build…. The hardest part of most designs of complex systems is not knowing how to design it, but what it is you are trying to design… when we are talking about eliciting requirements, we are talking about deciding what it is we are trying to design." (Fred Brooks)

In his keynote presentation, Fred Brooks indicated that requirements are the most difficult part of the design process, and that perhaps the software development community should look to other design disciplines such as mechanical engineering and architecture to see what it can learn. Moreover, he emphasized that the notions of a design as a linear, predictable activity that underlie many methods and tools of software development are too simplistic to provide a good understanding of design and its requirements. In recent years, a burgeoning design discipline [5], [6] appears to better understand general design practices and principles across disciplines. Further, the emerging design science [7], [8] paradigm looks to apply scientific theory and rigor to design artifacts and practices. As software is becoming increasingly implicated in a variety of design artifacts, it is imperative that requirements research maintain a dialog with these emerging perspectives in order to both feed and draw insight from the broader design discourse.

4.2   Evolution and the Fluidity of Design

"There's this profile of benefits and burdens that may not have been intended by the original designers, and frankly, who cares whether they were intended by the original designers. Who cares whether they were identified during the requirements process. Users generally don't know or can't predict what kinds of emergent additional uses or affordances exist in the implementation of a technology… In a sense the requirements are manifest in the use of a system when it's actually implemented, other than being a pure intent." (Colin Potts)

While evolutionary development has long been a central concern of requirements research and practice, in his presentation, Colin Potts described a shift in emphasis that is taking place in the requirements discipline. Requirements research frequently focuses on the need to understand and articulate user needs (i.e., requirements), often through ethnographic analysis of user activity with the system, and thus to build relevant functionality into future designs. However, in addition to this, there should also be a greater emphasis on designing artifacts with a level of malleability, or fluidity, in mind. This could involve practices such as co-design with users or developing toolkits for user customization, but can also involve intelligent agents that learn from usage, dynamically evolving artifacts, or user generated artifacts. The requirements community will be required to increasingly attend to post-development fluidity in a way that is notably different from the evolutionary discourses of the past. This attention should not only focus on the requirements themselves, but the meta-requirements associated with how adaptable requirements should be in classes of software-intensive systems.

4.3   Quality and Value-Based Requirements

"At the end of the day, what are we designing for? Are we designing really for a product? Are we designing for a service? Are we designing for … [an]

*experience? And as businesses we tend to think of how we can make a reasonable profit and be sustainable. But the consumer experience has got to be relevant, otherwise we get irrelevant…. People are trying to … stage the right kinds of experiences, and there are various ways of approaching that… "
(K. Venkatesh Prasad)*

As Venkatesh Prasad indicated in his presentation, a key goal of design is to "stage the right kind of experiences" and requirements practices must address the total experience a number of different ways. Brooks, Potts, Lyytinen, and others indicated during the workshop that the research and practice associated with the design of software intensive systems generally focuses on the functionality and performance of those systems, while at the same time focusing on the accuracy and completeness of the related requirements. Missing from much of the discourse are not only the "softer" performance characteristics, such as ease of maintenance, stability, and such, but also other characteristics of the designed system that involve critical attributes such as innovativeness, quality, economic considerations, and ethical implications. A focus on broader end-user environment and experience, as well as the organizational and societal implications of system use need to be more central to the requirements discourse.

4.4 Requirements Intertwining

"The business process and user process, or product environment, currently act as sources of requirements where [there is] an increased demand … for transparency across different systems and components, which guide requirements to integrate separate systems or components… architectures which have been adopted have a prominent role in hiding requirements. Much of that is normally based on packaged, commercial-off-the-shelf, or software components of different sorts. As a result, the processes are distributed widely within organizations, across organizations, and geographically. And they result in layered requirements across multiple dimensions…" (Kalle Lyytinen)

In his presentation, Kalle Lyytinen called attention to the way in which human activities traverse the ubiquity of software-intensive products. Requirements for new systems or changes to existing systems do not occur in a vacuum, and Greenfield development efforts are becoming exceedingly rare. Rather, oftentimes integration of existing systems, customization of packaged (COTS) software, embedded systems, and generic components are central to system development efforts. While researchers have long established that understanding end-user processes and target organizational environments is important to requirements practices (e.g., [9]), they generally look to business process modeling as a means to inform the development of a single system that supports a single process or set of processes. While this scenario is still the case in certain circumstances, increasingly any system being developed is but one component of a broader ecology of systems that support human activity. The emphasis is less on identifying processes that are dedicated to a single system, than understanding how the system fits within an array of nested business systems and processes with a diverse and ever-expanding set of related stakeholders. Systems are not only appropriated by heterogeneous communities, but other, non-user, community interests are increasingly relevant to contemporary system development. Accordingly, many of the workshop participants were advocating a more central role for the interaction of

different stakeholder perspectives and interests. Notions such as enterprise architecture, transparency, digital convergence, organizational structure, and multiple processes take a central position in these emerging views of requirements activity.

4.5 Adapting Requirements Practices in Different Domains

While not a central theme in the workshop, a number of participants made the observation that requirements research does not often distinguish between the different vertical domains of application or the different genres of software systems. While much of requirements research provides generalizable insights, it is important to understand the range of relevant design domains and their relationships with requirements practice.

5 Looking Ahead

Good design of complex systems is a daunting challenge, and as Fred Brooks indicated in his keynote address, requirements-related activity is "the hardest part of the challenge." Requirements researchers and practitioners must continually face new challenges, and this volume represents an attempt to summarize the current state of research and practice, as well as to identify trends and opportunities around broad five themes, and in each one of them point to areas that should be emphasized in the future.

With this volume, the High Impact Design Requirements Project is by no means complete. New workshops are already being scheduled, research outlets are being informed of the project, and the project is looking to expand its reach. Together we hope that we can set the agenda for requirements research and practice in the dawn of the 21st century and impact the design of society's vital systems at a time when this the design of complex software-intensive systems is becoming increasingly challenging, increasingly in demand, increasingly vital, and increasingly rewarding both intellectually and societally.

References

1. Hansen, S., Berente, N., Lyytinen, K.J.: Requirements in the 21st Century: Current Practice & Emerging Trends. In: Loucopoulos, P., Lyytinen, K.J., Mylopoulos, J., Robinson, W. (eds.) Design Requirements Engineering: A Ten-Year Perspective. LNBIP. Springer, Heidelberg (2008)
2. Kaindl, H., Brinkkemper, S., Bubenko Jr., J.A., Farbey, B., Greenspan, S.J., Heitmeyer, C.L., Leite, J.C.S.P., Mead, N.R., Mylopoulos, J., Siddiqi, J.: Requirements Engineering and Technology Transfer: Obstacles, Incentives and Improvement Agenda. Requirements Engineering 7, 113–123 (2002)
3. Zowghi, D., Coulin, C.: Requirements Elicitation: A Survey of Techniques, Approaches, and Tools. Engineering and Managing Software Requirements (2005)
4. Davis, A.M., Hickey, A.M.: Requirements Researchers: Do We Practice What We Preach? Requirements Engineering 7, 107–111 (2002)

5. Buchanan, R., Margolin, V.: Discovering Design: Explorations in Design Studies. University Of Chicago Press (1995)
6. Michel, R.: Design Research Now: Essays and Selected Projects. Birkhäuser, Basel (2007)
7. Hevner, A., March, S., Park, J., Ram, S.: Design Science in Information Systems Research. MIS Quarterly 28, 75–106 (2004)
8. March, S.T., Smith, G.F.: Design and natural science research on information technology. Decision Support Systems 15, 251–266 (1995)
9. Curtis, B., Kellner, M., Over, J.: Process modeling. Communications of the ACM 35, 75–90 (1992)

Appendix: Design Requirements Workshop Participants

Plenary Presentations:

- Fred Brooks, University of North Carolina (Keynote)
- John Leslie King, University of Michigan
- Kalle Lyytinen, Case Western Reserve University
- Colin Potts, Georgia Institute of Technology
- K. Venkatesh Prasad, Ford Motor Company

Participants:

- Michel Avital, Case Western Reserve University
- Liam Bannon, University of Limerick
- Nicholas Berente, University of Michigan
- Mark Bergman, Naval Postgraduate School
- JoAnne Brooks - MITRE Corporation
- Fred Collopy, Case Western Reserve University
- Roger Dunbar, New York University
- Steve Easterbrook, University of Toronto
- Martin Feather, NASA-JPL & California Institute of Technology
- Anthony Finkelstein, University College London
- Gerhard Fischer, University of Colorado
- Raghu Garud, Pennsylvania State University
- Sean Hansen, Case Western Reserve University
- John Henderson, Boston University
- Sol Greenspan, National Science Foundation
- Matthias Jarke, RWTH-Aachen
- Yusun Jung, Case Western Reserve University
- Helmut Krcmar, Technical University Munich
- Pericles Loucopoulos, Loughborough Business School
- Sal March, Vanderbilt University
- John Mylopoulos, University of Toronto
- Sasi Pillay – NASA Glenn Research Center

- Isabelle Reymen, Eindoven University of Technology
- William Robinson, Georgia State University
- Colette Rolland, Universite Paris-Sorbonne
- Walt Scacchi, University of California - Irvine
- Baldev Singh - Motorola, Inc.
- Nikhil Srinivasan, Case Western Reserve University
- Alistair Sutcliffe, University of Manchester
- Yair Wand, University of British Columbia

Current and Future Research Directions in Requirements Engineering⋆

Betty H.C. Cheng[1] and Joanne M. Atlee[2]

[1] Michigan State University, East Lansing, MI 48824, IUSA
chengb@cse.msu.edu
http://www.cse.msu.edu/~chengb
[2] University of Waterloo, Waterloo, Ontario N2L 3G1 Canada
jmatlee@uwaterloo.ca
http://se.uwaterloo.ca/~jmatlee

Abstract. In this paper, we review current requirements engineering (RE) research and identify future research directions suggested by emerging software needs. First, we overview the state of the art in RE research. The research is considered with respect to technologies developed to address specific requirements tasks, such as elicitation, modeling, and analysis. Such a review enables us to identify mature areas of research, as well as areas that warrant further investigation. Next, we review several strategies for performing and extending RE research results, to help delineate the scope of future research directions. Finally, we highlight what we consider to be the "hot" current and future research topics, which aim to address RE needs for emerging systems of the future.

Keywords: requirements engineering, modeling, analysis, elicitation.

1 Introduction

The success of a software system depends on how well it fits the needs of its users and its environment [1, 2]. *Software requirements* comprise these needs, and *requirements engineering (RE)* is the process by which the requirements are determined. Successful RE involves understanding the needs of users, customers, and other stakeholders; understanding the contexts in which the to-be-developed software will be used; modeling, analyzing, negotiating, and documenting the stakeholders' requirements; validating that the documented requirements match the negotiated requirements; and managing requirements evolution.[1]

⋆ A preliminary and shortened version of this paper was published in IEEE International Conference on Software Engineering, Future of Software Engineering, 2007. Portions of the preliminary version have been included with permission from IEEE.

[1] In addition, there are a number of software-engineering activities that are based on requirements information, such as cost estimation, project planning, and requirements-based derivations of architectures, designs, code, and test cases. Although these activities are "related" to a system's requirements, they play at most a minor role in determining and agreeing on the system's requirements; as such, we consider them to be outside the scope of requirements engineering.

In this paper, we offer our views on the research directions in requirements engineering. The paper builds on Nuseibeh and Easterbrook's paper [1], hereafter referred to as the "2000 RE Roadmap Paper", from the Future of Software Engineering track at ICSE 2000 [3]. Whereas the 2000 RE Roadmap Paper focused on current research in requirements engineering, this paper concentrates on research directions and identifies RE challenges posed by emerging and future software needs. We start, in Section 2, with an overview of the inherent difficulties in requirements engineering. In Section 3, we provide a summary of the state of the art of RE knowledge and research, and in Section 4, we enumerate general research strategies for advancing the state of the art. The strategies range from revolutionary breakthroughs to empirical evaluation to codifying proven solutions. In Section 5, we highlight what we consider to be RE research *hotspots*: the most pressing needs and grand challenges in RE research. Some hotspot topics are natural extensions to existing knowledge and technologies, whereas others arise as RE aspects of predicted software needs. We conclude with strategic recommendations for improving the research infrastructure for RE researchers, so that they can make better progress on addressing these problems.

2   Why Requirements Engineering Is Hard

In general, the research challenges faced by the requirements-engineering community are distinct from those faced by the general software-engineering community, because requirements reside primarily in the problem space whereas other software artifacts reside primarily in the solution space. That is, *requirements* descriptions, ideally, are written entirely in terms of the environment, describing how the environment is to be affected by the proposed system. In contrast, other software artifacts focus on the behavior of the proposed system, and are written in terms of internal software entities and properties. Stated another way, requirements engineering is about defining precisely the *problem* that the software is to solve (i.e., defining *what* the software is to do), whereas other SE activities are about defining and refining a proposed software *solution*.

Several consequences follow from this distinction that cause requirements engineering to be inherently difficult:

– Requirements analysts start with ill-defined, and often conflicting, ideas of what the proposed system is to do, and must progress towards a single coherent, detailed, technical specification of the system.
– The requirements problem space is less constrained than the software solution space – in fact, it is the requirements definition that helps to delimit the solution space. As such, there are many more options to consider and decisions to make about requirements, such as selecting from collections of proposed requirements, prioritizing requirements, deciding on the system boundaries, negotiating resolutions to conflicts, setting objective acceptance criteria, and so on [4].
– One means of simplifying the problem space is to constrain the environmental conditions under which the system is expected to operate. In such cases, reasoning about requirements involves reasoning about the combined behavior of the proposed system and assumptions made about the environment. To complicate things,

a system's environment may be a combination of hardware devices, physical-world phenomena, human (operator or user) behavior, and other software components.

 – Reasoning about the environment includes identifying not only assumptions about the normal behavior of the environment, but also about possible threats or hazards that the environment could pose to the system.
 – The resulting requirements artifacts have to be understood and usable by domain experts and other stakeholders, who may not be knowledgeable about computing. Thus, requirements notations and processes must maintain a delicate balance between producing descriptions that are suitable for different stakeholders and producing technical documents that are precise enough for downstream developers.

Due to all of the above, RE activities, in contrast to other software-engineering activities, may be more iterative, involve many more players who have more varied backgrounds and expertise, require more extensive analyses of options, and call for more complicated verifications of more diverse (e.g., software, hardware, human) components.

3   State of the Art of RE Research

In this section, we summarize the state of the art of RE knowledge and research, as a baseline from which to explore future research directions. This section can be viewed as an update to the 2000 RE Roadmap Paper [1], in that it incorporates advances made in the intervening several years.

To provide a visual map of RE research, we organize research results in a matrix structure that relates each result to the requirements task that it applies to and the contribution that it makes towards a solution. The research space is roughly decomposed into five types of requirements tasks (elicitation, modeling, requirements analysis, validation and verification, and requirements management) and three categories of solution technologies (notations, methodologies and techniques, and tools). The *Notations/Languages* column refers to notations and languages that are used to support the respective requirements tasks. *Methodologies and Strategies* cover different techniques, artifacts, and guidelines designed to support the RE tasks. Finally, *Tools* refer to automation-supported techniques intended to evaluate various types of RE artifacts. The resulting matrix is shown in Table 2 Not all RE research can be so cleanly classified, but this decomposition is useful for a high-level overview of solution-based research activities; evaluation-based research is discussed separately. Our decomposition is roughly comparable to the top-level decomposition in Zave's proposed scheme for classifying RE research [122].

The rest of this section is organized by requirements task, and thus reviews the contents of the matrix by row. We conclude with a discussion of evaluation-based RE research.

Elicitation. Requirements elicitation comprises activities that enable the understanding of the goals, objectives, and motives for building a proposed software system. Elicitation also involves identifying the requirements that the resulting system must satisfy

Table 1. Matrix Summarizing the State of the Art of Requirements Engineering

Requirements Tasks	Requirements Technologies		
	Notations/Languages	Methodologies and Strategies	Tools
Elicitation	Goals [23, 123, 190] Use cases [39], Policies [22] Scenarios [1, 53] Agents [120, 203] Anti-models [174, 182, 191] Nonfunctional requirements [34, 79]	Identifying stakeholders [169] Metaphors [148, 151] Persona [10, 41] Contextual requirements [40, 177] Inventing requirements [130]	Animation [96, 128, 187] Simulation [180] Invariant generation [106]
Modeling	Object models [101] Goal models [190, 202] Behavioral models [105, 183] Domain descriptions [12] Property languages [17, 56, 119] Notation Semantics [66, 134, 140, 179] Problem Frames [102]	RE reference model [89, 90, 146] Goal-based refinement [121, 120] Aspect-oriented [153, 14] Model elaboration [185] Viewpoints [143, 172] Patterns [55, 60, 102, 111, 188] NL-based facilitators [7, 38, 82, 109, 144] Formalization heuristics [22, 76] Methodologies [18]	Model merging [164, 181] Model synthesis [4, 45, 121, 184, 200] Model composition [93] Metrics-based evaluation [108]
Requirements Analysis		Negotiation [100] Aligning requirements with COTS [6, 160] Conflict management [159] Inquiry-based [152] Evaluation and selection [155] Inspections [67, 147] Checklists [195] Ontologies [108]	Linguistic analysis [20, 33, 194] Consistency checking [65, 95, 137] Conflict analysis [30, 92] Obstacle analysis [127, 193] Risk analysis and management [69, 201] Impact analysis [114] Causal order analysis [13] Prioritization [136] Metrics-Based Analysis [19] Variability analysis [87, 122, 124] Evolutionary requirements analysis [176]
Validation & Verification	Model formalisms [26, 57]	Inspection [67] State-based exploration [17, 199] Scenario-based [27, 88, 162]	Simulation [180] Animation [96, 128, 187] Invariant generation [106] Model checking [32, 62, 175] Model satisfiability [101]
Requirements Management	Variability modeling [25, 44, 156, 167]	Scenario management [3] Feature management [156, 167, 197] Global RE [50]	Traceability [37, 94, 163, 168] Stability analysis [27]

in order to achieve these goals. The requirements to be elicited may range from modifications to well-understood problems and systems (e.g., software upgrades), to hazy understandings of new problems being automated, to relatively unconstrained requirements that are open to innovation[2] (e.g., mass-market software). As such, most of the research in elicitation focuses on technologies for improving the precision, accuracy, and variety of the requirements details:

– Techniques for *identifying stakeholders* [19] help to ensure that everyone who may be affected by the software is consulted during elicitation.
– *Analogical techniques*, like metaphors [21] and personas [22,23], help stakeholders to consider more deeply and be more precise about their requirements.
– *Contextual and personal RE techniques* [24,25] analyze stakeholders' requirements with respect to a particular context, environment, and perhaps individual user, to help ensure that the eventual system is fit for use in that environment.
– Techniques for *inventing requirements*, like brainstorming and creativity workshops [26], help to identify nonessential requirements that make the final product more appealing.
– *Feedback techniques* use models [123], model animations [27, 28, 29], simulation [30], and storyboards to elicit positive and negative feedback on early representations of the proposed system.

Models can be used during elicitation to help catalyze discussion and to explore and learn about the stakeholders' needs. Such exploratory models, like use cases, scenarios, enterprise models, and some policy [9] and goal models [5], tend to be informal and intuitive, to facilitate early feedback from stakeholders. They tend also to be inexpensive to create and maintain, so that specifiers can keep them up-to-date as the requirements evolve.

Modeling. In *requirements modeling*, a project's requirements or specification is expressed in terms of one or more models. In contrast to models developed during elicitation, late-phase requirements models tend to be more precise, complete, and unambiguous. The process of creating precise models helps to evoke details that were missed in the initial elicitation. The resulting (more complete) models can be used to communicate the requirements to downstream developers.

Modeling notations help to raise the level of abstraction in requirements descriptions by providing a vocabulary and structural rules that more closely match – better than natural language does – the entities, relationships, behavior, and constraints of the problem being modeled. Each modeling notation is designed to elicit or record specific details about the requirements, such as what data the software is to maintain, functions on the data, responses to inputs, or properties about data or behavior.

Scenario-based models [10, 67, 68, 11, 66, 69, 51, 70] have been the focus of much recent research – partly because scenarios are easiest for practitioners and nontechnical stakeholders to use, but perhaps also because scenarios are naturally incomplete and thus lend themselves to a plethora of research problems. In addition, there is considerable research on techniques for creating, combining, and manipulating models:

---

[2] Innovations are inventions that people are willing to purchase.

– *Modeling strategies* provide guidelines for structuring models. For example, *RE reference models* [45, 46, 47] decompose requirements-related descriptions into the stakeholders' *requirements*, the *specification* of the proposed system, and assumptions made about the system's *environment*. In addition, they establish correctness criteria for verifying that the specified system will meet the requirements. In contrast, the *viewpoints* approach [52, 53] retains each stakeholder's requirements in separate models, and the synthesis of a consistent global model that captures all of the stakeholders' concerns is delayed until conflicts can be resolved knowledgeably.
– *Patterns* encode generic solutions to common modeling problems [44, 56, 57], assertion expressions [55], and natural-language requirements statements [54]. The RE community is also working on tools [59, 61, 62] to help specifiers apply these patterns.
– *Model transformations* combine or manipulate existing models to derive new models. For example, *model synthesis* [67, 68, 48, 69, 70] and *model composition* techniques [71] integrate complementary submodels into a composite model. In contrast, *model merging* techniques [65, 66] unify different views of the same problem.

Several of the above-mentioned projects directly address challenges raised in the 2000 RE Roadmap Paper. For example, heuristics for formalizing natural-language policies [9] and goal models [63] help to bridge the gap between informal and formal requirements. This gap is also narrowed by techniques for inferring abstractions [60] and preliminary models [58] from natural-language requirements, by tools that map constrained natural-language expressions to formal representations [59, 61, 62], and by research on formalizing the semantics of informal or semi-formal modeling notations [40, 124, 41, 43]. In addition, significant advances have been made in the modeling and analysis of nonfunctional requirements [17] and in establishing objective fit criteria for how well an eventual system must achieve various nonfunctional properties [18]. On the other hand, there has been little progress on special-purpose notations for modeling environment descriptions and assumptions [36]. Instead, existing notations like functions [47], operational specifications (e.g., Z, Alloy [32]), and constraint languages continue to be used.

Requirements Analysis. Most of the research in *requirements analysis* focuses on new or improved techniques for evaluating the quality of recorded requirements, making tradeoff decisions among requirements, and for improving the understanding of requirements. A limited number of these approaches are intended to support multiple stakeholder-based evaluation of the requirements, and thus require more human input throughout the evaluation process. As such, we put these techniques under the *Methodologies and Strategies* column, including negotiation [73], inquiry-based [77], and inspections [79]. Other requirements analysis techniques are more automated and tend to focus on a specific dimension of evaluation (such as risk, conflicts, or variability); and therefore we list these techniques under the *Tools* column. For example, several analysis tools look for well-formedness errors in requirements, where an "error" can be ambiguity [82, 125, 72, 126, 84], inconsistency [127, 85, 87], or incompleteness. Other analyses look for anomalies, such as unknown interactions among requirements [88, 89, 76], possible obstacles to requirements satisfaction [90, 91], or missing assumptions [95].

Both types of analyses reveal misunderstandings or questions about the requirements that usually call for further elicitation, thus moving back to the interactive techniques. Requirements analysis also includes consequence-based analysis techniques, such as risk analysis [92] and impact analysis [94], that help specifiers to understand better the requirements, their interrelationships, and their potential consequences, so that specifiers can make better decisions. As other examples, prioritization, visualization, and analysis techniques help a manager to select an optimal combination of requirements to be implemented [98, 100, 78, 96, 101], or to identify acceptable off-the-shelf solutions [128, 75].

Validation and Verification. *Requirements validation* ensures that models and documentation accurately express the stakeholders' needs. Unlike the above analyses that check a specification against objective well-formedness criteria, validation is normally a subjective evaluation of the specification with respect to informally described or undocumented requirements. As such, validation usually requires stakeholders to be directly involved in reviewing the requirements artifacts [129, 79]. Research in this area focuses on improving the information provided to the stakeholder for feedback, including animations [27, 28, 29], simulations [30], and derived invariants [31]. Many of the techniques are based on scenario validation [105, 106, 107].

In cases where a formal description of the stakeholders' requirements exists, obtained perhaps by validation, *verification* techniques can be used to prove that the software specification meets these requirements. Such proofs often take the form of checking that a specification model satisfies some constraint. For example, *model checking* [108, 109, 110] checks behavioral models against temporal-logic properties about execution traces; and *model satisfiability* [32] checks that there exist valid instantiations of constrained object models, and that operations on object models preserve invariants.

The notations listed in the *Validation & Verification* row of Table 3 represent formalisms that enable or ease verification. In contrast to specification notations, these notations' primary purpose is to facilitate automated verification rather than to communicate or document requirements. Verification models, expressed in these notations, are simplifications and abstractions of a specification to be verified [102, 103, 32].

Requirements management. *Requirements management* is an umbrella activity that comprises a number of tasks related to the management of requirements, including the evolution of requirements over time and across product families. Of particular interest are tools and techniques to ease, and partially automate, the task of identifying and documenting traceability links among requirements artifacts and between requirements and downstream artifacts [118, 119, 120, 121, 130]. Also included are analyses that determine the maturity and stability of elicited requirements, so that the requirements most likely to change can be isolated [105]. Lastly, the basic management of requirements has become a challenge and has inspired research on techniques to organize large numbers of requirements [115] that are globally distributed [117], and that are at different phases in development in different product variants [113, 114, 116].

Evaluation-Based Research. The above discussion is an overview of solution-based RE research, which emphasizes technological advances that make progress towards

solving RE problems; such research is often accompanied by proofs-of-concepts or pilot studies that show the potential of the proposed ideas. Orthogonal to this work is *evaluation-based research*, whose mission is to assess the state of the practice and evaluate proposed advances to the state of the art. Approximately 10% to 15% of RE research papers report on evaluation-based research, in the form of reports on the state of the practice [131], case studies that evaluate how well research ideas work when applied to industrial-strength problems³ [73, 132, 133, 134, 104], and field studies that evaluate research ideas in industrial settings [135, 136, 137]. Several recent research projects evaluate how well requirements technologies apply to, or can be adapted to, domain-specific problems, such as security [16], semantic webs [138], and user interfaces [139]. Additionally, there have been a few comparative studies that compare the effectiveness of competing elicitation techniques [140, 141], specification notations [142], and inspection techniques [143]. Finally, there have also been some post-mortem analyses on how requirements evolved in real-world systems [144, 145].

4   Research Strategies

In this section, we discuss ways of advancing the state of the art of RE research. We review several major strategies for conducting research, and look at how each have or might be applied to requirements-related research. The strategies range from inventing new disruptive ideas and technologies, to improving on current research, to adapting previous results to a new context, to evaluating or comparing technologies. Each strategy attempts to achieve a slightly different research objective, but all contribute in some way to advancing the state of the art, either by adding new knowledge or by improving the maturity of previous work.

Our collection of research strategies is synthesized from a number of different sources, including Shaw's overview of criteria for good research in software engineering [146], Redwine and Riddle's review of software technology maturation [147], Basili's review of research paradigms [148], and the combined experience of both authors. Table 2 introduces and briefly defines the eight research strategies that we discuss below.

Paradigm Shift.   A *paradigm shift* is a revolutionary solution that introduces radically new ideas or technologies to tackle a new or existing problem. A paradigm shift may be called for when researchers can no longer make progress on an important problem by extending or adapting existing technologies. Typically, there are two means by which a paradigm shift occurs: push and pull. A paradigm shift is *pushed* onto a community when new technology serendipitously makes major advances towards solving a problem for which it was not originally intended. A classic example of such a shift is the World Wide Web, which has significantly changed the way that society communicates and the way that services are delivered to consumers. A paradigm shift that is currently underway is the shift toward global software development and, by extension, global requirements engineering; we discuss this topic further in Section 5.1.

---

³ We use the term "industrial-strength" problems/projects to refer to project data that have characteristics of industrial examples, such as size, complexity, and/or domain-specific properties.

Table 2. Enumeration of research strategies

Research Strategy	Definition
Paradigm Shift:	Dramatically change the way of thinking, resulting in a revolution in knowledge or technology.
Leverage other disciplines:	Leverage and recast principles, practices, processes, or techniques from another discipline.
Leverage new technology:	Make advances by leveraging new tools or technology.
Evolutionary:	Make progressive improvements to existing research solutions and techniques.
Domain-specific:	Develop a new solution or technique that applies narrowly to a specific problem domain.
Generalization:	Generalize an existing solution or technique, so that it applies to a broader class of problems or data.
Engineering:	Develop processes or strategies that make it easier or cheaper to apply research solutions in practice.
Evaluation:	Evaluate existing research solutions – with respect to specified metrics, real or realistic problems, current practices, or related research results.

Alternatively, a paradigm shift can be *pulled* when there is a real or a perceived crisis that cannot be solved by improving current ideas and techniques [149]. For example, object-based design conventions were invented in response to serious concerns about how to structure programs and data in a way that promoted modularity. As the design conventions gained popularity, they evolved into object-oriented programming methodologies, were codified in new design methods, and were eventually supported by new programming language constructs.

Paradigm shifts are rare and are usually unplanned; but when they occur, they can have tremendous impact on a field. A paradigm shift starts with some innovative change in the way that a particular problem is studied. The change leads to disruptive innovations, which usually must mature before their benefits are recognized and appreciated enough to motivate rapid and widespread adoption.

Leverage other disciplines. An RE researcher can leverage another discipline by identifying the analogous relationships between the two disciplines and then recasting promising knowledge, philosophies, principles, or practices from the other discipline into solutions that are appropriate for requirements problems. For example, software engineering, as a discipline, emerged when researchers and practitioners attempted to manage the "software crisis" by borrowing and adapting from the engineering profession several ideas about design principles, development processes, and rigor. As another example, the concept of genetic algorithms leverages ideas from biology, in that the algorithms "evolve" by using feedback from previous computations to improve future

computations. Sutcliffe et al. [150] use genetic algorithms to improve the efficiency of searching for an optimal set of components that satisfy a given set of fitness criteria for reliability requirements. As a third example, Sutcliffe and Maiden's work [151] in establishing a domain theory for RE draws heavily from cognitive science and the human use of analogical reasoning.

Leverage technology.   Technological advances in computing and related fields can be combined and adapted to apply to problems in requirements engineering. In general, artificial intelligence, library science, information science, cognitive psychology, linguistics, statistics, and mathematics are all fertile areas for ideas and techniques that are suitable for such adaptation. For example, Ambriola and Gervasi [58], and separately Overmeyer et al. [62], use natural-language processing techniques to parse textual requirements descriptions and to generate corresponding semi-formal models, such as data-flow diagrams and communication diagrams. They and other researchers [58, 82, 125, 126, 84] use linguistic-analysis techniques to detect possible ambiguities and unintended inconsistencies in textual or use-case requirements. Hayes et al. [119] and Cleland-Huang et al. [152] use information-retrieval techniques to automatically retrieve traceability links among requirements.

Evolutionary research.   The antithesis of a paradigm shift is evolutionary research, in which the state of the art advances via incremental improvements to existing technologies. Although emerging software needs may pose new research challenges that the RE community will be called on to address, most software developed in the near future will resemble the types of systems being developed today. As such, the software community will continue to benefit from improvements to current requirements technologies (as overviewed in Section 3), which were created in response to the problems that today's practitioners face.

In many ways, evolutionary research is about moving research technologies down the research-strategy ladder listed in Table 2. Existing notations and techniques can be extended, adapted, or generalized to address a broader class of problems. Current technologies can be supported by new methodologies, patterns, strategies, and tools that ease their use and help to promote their adoption by practitioners. Empirical research can determine the problems and contexts for which a technology is most effective, and can identify aspects that could be further improved.

Domain-specific.   A researcher can sometimes make better progress by narrowing the scope of a requirements problem and studying it in the context of a particular application domain. For example, there is a paradigm shift towards more domain-specific specification languages that provide native facilities for describing important entities and behaviors in that domain and provide macros for eliding recurrent requirements details. Along these lines, the International Telecommunication Union (ITU) has standardized a number of specification, design, and testing languages; design methodologies; and interface specifications – all of which support software aspects of telecommunication systems.

Generalization.   Successful domain-specific or organization-specific techniques can sometimes be generalized to be more broadly applicable. For example, many of the

ideas in telecommunications notations, like Message Sequence Charts and the Specification and Description Language, have been incorporated into the more general Unified Modeling Language 2.0. As another example, techniques for avoiding, detecting, and resolving feature interactions, which were originally developed in the context of telephony features [71, 153], are now being studied and applied in other domains, such as Web services [154].
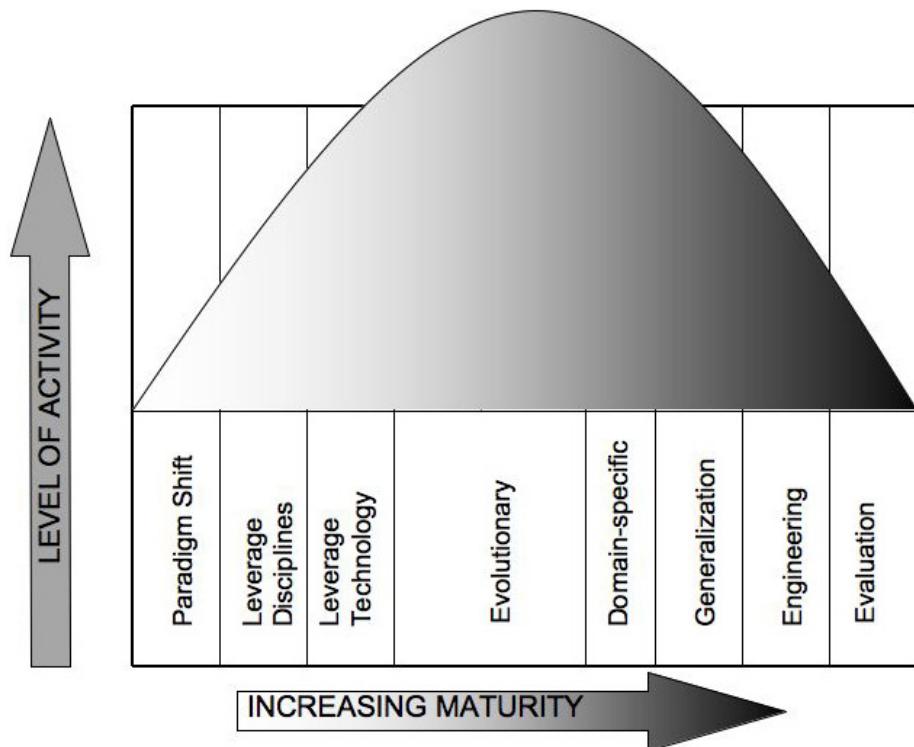
Engineering.   A surprising number of research problems arise in the course of trying to apply requirements technologies in practice. Engineering as a research strategy looks at how to simplify and codify RE knowledge and techniques so that they can be readily adopted by practitioners and taught in classrooms. For example, visual formalisms [38, 155, 156, 34] ease the task of creating and reviewing precise specifications. Patterns not only help specifiers to create models [157, 56, 57] and express constraints [55], via instantiation and adaptation, but they also offer some level of uniformity and repeatability of such descriptions. Heuristics and strategies offer advice on how to use particular elicitation [158], modeling [9, 63], or verification [159] technologies. Methodologies and processes provide guidance on how to integrate RE technologies to progress from an initial idea to a final specification document [77, 160]. One of the best known engineering-style research projects was Parnas et al.'s case study that applied state-of-the-art software engineering practices to (re)develop the engineering artifacts and code for the U.S. A-7 naval aircraft. This work led to research results in tabular specifications [34], hierarchical module structures, abstract interfaces, and new inspection strategies [80].

Evaluation.   Proposed RE technologies become theories, solutions, or practices through evaluation-based research that demonstrate effectiveness. Evaluation techniques include experience, collection and analysis of data, field studies, case studies, controlled experiments, and analytical reasoning. Evaluation criteria range from qualitative or statistical metrics, to effectiveness in solving real or realistic problems, to comparisons with competing technologies. A mature RE technology should be evaluated on real-world applications or in an industrial setting, to assess its scalability, practicality, and ease of use [137, 73, 132, 133, 134, 104]. In contrast, comparative studies evaluate the relative strengths and weaknesses of competing solutions to a problem. Notable comparative studies have investigated the criteria for choosing a specification language [142] and the effectiveness of methods for inspecting requirements documents [143].

Evaluation-based research need not be a massive project. A case-study may be based on a single study involving an industrial-strength project, on replicated studies of the same project, on studies of multiple projects, or on a longitudinal study that spans several phases of a project. Even the development of new assessment criteria, such as appropriate benchmarks, are valuable research contributions.

Level of Research Activity.   Figure 1 depicts the level of research activity in each of the areas, where (from left to right) the strategies are listed in order of increasing maturity of the research results.[4] The parabola delineates the level of activity in each area,

---

[4] Here, we use the term "maturity" to refer to the extent that the research results have been validated and refined by people, especially people other than the original authors.

Fig. 1. Level of activity using different research strategies

where the majority of the research activities are in the middle region (i.e., evolutionary and domain-specific). And the shading indicates the level of uncertainty involved in the research investigations. The lightest shading indicates the most radical, pioneering research that involves exploring unknown territory with very little assurance about the likelihood that the research will lead to successful and useful results. But if successful the research could have very high impact. In contrast, the darkest shading indicates that significant information is already known about the technique, but evaluation and empirical research is needed to evaluate the utility of the technique under various contexts.

5   RE Research Hotspots

As evidenced by the previous sections, the field of RE research is rich, and the scope of possible research directions is quite large. In addition, new RE research challenges are posed by emerging trends in software systems and predictions about future software needs. In this section, we highlight what we consider to be RE research *hotspots*: that is, those RE problems whose solutions are likely to have the greatest impact on software-engineering research and practice. We decompose our discussion into three major categories: research that will facilitate the use of RE techniques; improvements

to the current state of the art of RE techniques; and RE research challenges posed by emerging needs that require transformative research results.

Emerging needs include increasing scale of software systems, tighter integration between software and its environment, greater autonomy of software to adapt to its environment, and increasing globalization of software development. These trends reflect changes in stakeholders' needs, and as such they directly affect RE processes and practices. In some cases, current technologies can accommodate the new trends in requirements. In other cases, the trends pose new research challenges in requirements engineering, or raise the priorities of longstanding research problems.

Our list of hotspots is not meant to be exhaustive. Rather, it is intended to highlight some of the more pressing needs and grand challenges in RE research. We start with three hotspots that focus on extending and maturing existing research technologies with the objective of facilitating their use by a broader RE community: improved methodologies, requirements reuse, and more evaluation-based research. All three of these hotspots are even more critical when put in the context of global software development. As such, additional challenges posed by global requirements engineering round out this cluster of hotspots. Next, given that computing-based systems are increasingly being used for safety-critical application domains (e.g., automotive, medical, transportation, finance, and etc.), we highlight two hotspots targeted towards improving systems assurance: security and fault tolerance. Finally, we end by identifying three hotpots that arise from the need to explicitly deal with many levels of uncertainty, an inherent aspect of future systems: increased scale, tight coupling between the software and its physical environment, and self-management.

5.1 Facilitating the Use of RE Techniques

As can be seen in Table 2, a rich collection of RE techniques has been developed over the past thirty years. And yet discussions with developers from industry indicate that the most challenging dimension of developing computing-based systems still centers around requirements. As computing-based systems become increasingly complex, operating in more critical application domains, the need for more prevalent use of rigorous RE techniques becomes more essential. Realization of these pressing needs has prompted a number of industrial organizations to allocate more resources for RE-related tasks [161] and establish collaborative partnerships with academic researchers to explore viable solutions for near-term and longer-term RE needs [162, 163, 164, 165, 166, 167]. At the same time, the RE research community can facilitate a more widespread use of these techniques by making more of the requirements-related activities closer to that of an engineering process. As described in Section 3, to enable engineering, techniques need to be simplified and packaged for reuse by people other than the original authors, RE knowledge and strategies need to be codified, and industrial-strength applications of techniques need to be documented for potential users to assess the applicability to their respective problems. Currently, the typical approach for sharing research results with the community is in the form of conference papers. But due to practical reasons, these papers are typically not allocated enough space to allow authors to describe their respective techniques in sufficient detail for others to duplicate the investigations or apply the techniques to a new problem. Even those authors who go the extra step to publish a

journal version of the paper still are typically only able to add an incremental amount of details over the conference paper. In most cases, the authors add the theoretical details about a given technique in the journal paper. In addition to the theoretical foundations, it is also necessary to document the research results in sufficient detail for others to duplicate the results, even if it means putting the information into an unpublished technical report or on a publicly accessible website. It is commonly recognized that experimental work is expensive and time-consuming, therefore, the academy and the research dissemination venues (e.g., conferences and journals) need to emphasize more the importance of the documentation and publication of empirical research results to encourage work along these lines; we are seeing some progress along these lines [168]. Below, we mention three key areas of research needs that will further facilitate the use of existing and future RE techniques: empirical evaluation, codification of methodologies, and leverage and reuse of RE artifacts. Finally, we discuss how the challenges posed by global software development will further motivate the need to make RE techniques more amenable to technology transfer and reusable when working across geographical, cultural, and time zone boundaries.

Effectiveness of RE Technologies. The ultimate impact of RE research depends on how relevant the results are to industry's short- and long-term needs. So far, there has been surprisingly little evaluation as to how well RE research results address industrial problems. As mentioned in Section 3, most empirical RE research takes the form of proof-of-concept studies or pilot studies, both of which qualitatively evaluate how well a proposed solution or technique applies to a single concrete problem. Such studies tend to be aimed at research audiences, and are intended to convince readers that the RE technologies under evaluation advance the state of the art. However, given that most studies report success, how is a practitioner to determine when a study reports a significant enough advance to warrant changes to the state of the practice, and how is a practitioner to select from among competing technologies?

Practitioners need hard evidence that a new technology is cost-effective, in order to justify the overhead of changing their development processes. In particular, practitioners would benefit greatly from empirical studies that assess the costs and benefits of using proposed technologies, assess the scope of problems to which research results can feasibly be applied, and compare the effectiveness of competing technologies. There have been a few studies along these lines. Damian et al. have conducted a series of surveys that evaluates the impact of requirements-related activities on productivity, risk management, and the quality of both requirements and downstream artifacts [135, 169, 170]. The Comparative Evaluation in Requirements Engineering (CERE) workshops investigate how to facilitate comparative studies, such as the development of suitable benchmarks. The Economics-Driven Software Engineering Research (EDSER) workshops investigate how to improve practitioners' abilities to make economics-based design and process decisions, such as whether or not to adopt new technologies. Such empirical research that evaluates requirements technologies in the context of industrial settings and practices would help to accelerate the transfer of research results into RE practice.

Methodologies and Tools. Better guidance on how to apply the technologies more systematically would facilitate the migration of RE technologies from research into

practice. The goals of this type of engineering-style research are to improve the productivity of the requirements analyst and to improve the quality of the resulting requirements artifacts. Modeling conventions, methodologies, and strategies all help to simplify RE techniques so that the techniques can be used successfully by ordinary practitioners.

Engineering-style research is also needed to investigate how to integrate requirements technologies into a coherent requirements process. Most research projects focus on a single RE problem, such as elicitation or traceability. As a result, the state of the art in RE research is a collection of technologies that have been researched and evaluated in isolation, with little knowledge of how to combine techniques effectively. For example, despite the significant advances that have been made in requirements modeling and notations, there has been little work on how to integrate various types of requirements models. Research is needed to develop well-defined approaches to interrelate requirements goals, scenarios, data, functions, state-based behavior, and constraints. Broy and his group have made some progress on this problem, in the form of a modeling theory that incorporates many of the above-mentioned modeling elements [171]. As an example of synergy among RE technologies, Ebert [172] shows via an analysis of several industrial-development projects that four product-management techniques – for composing teams, negotiating requirements, planning long-term product and feature releases, and tracking a product's status – are most effective at reducing scheduling delays when the techniques are used together. Further research is needed on how to integrate RE technologies, so that practitioners know how to apply individual technologies effectively and synergistically. Finally, more research is needed to systematically derive downstream artifacts, such as architectural elements and test cases, from the RE artifacts.

Requirements Reuse. Another approach to making RE tasks more prescriptive and systematic would be to facilitate the reuse of existing requirements artifacts. The most strategic form of requirements reuse is *product lining,* where related products are treated as a product family, and their co-development is planned from the beginning. The family's common requirements are collected in reusable templates that can be instantiated and adapted to derive the requirements for an individual product. A key RE challenge for product-line development includes strategic and effective techniques for analyzing domains; identifying opportunities for product lining; and identifying the scope, commonalities, and variabilities of a product line. A second challenge relates to how requirements for product lines are documented. Feature models [112, 173] are commonly used to model a product-line core, but they quickly proliferate when used to model product-line instantiations. A promising but untested solution to this challenge is multi-level feature trees [113].

Specification and modeling patterns are also a form of reuse, in that they codify reusable modeling structures. For example, just as specification patterns [55] help to ease the creation of logic specifications, research into idioms and patterns for other modeling problems and notations [157, 56, 57] would improve the productivity of modelers. Problem frames [44] can be considered abstract patterns of context diagrams for common classes of software problems, and thus are also reusable. In addition, it may be possible to identify larger units of reusable requirements for particular domains or

particular types of applications. The automotive industry has expressed interest in using such "generic, reusable requirements" in developing complex automotive systems.

A reusable requirement should be accompanied by standard pattern fields, such as context, problem addressed, consequences, properties, and so on. However, this is usually not enough information to facilitate effective use of the pattern or reusable artifact. Adapting an instantiated pattern so that it adequately fits the desired context is still a bit of an art. Pattern use would be easier and more successful if practitioners had better guidance and examples of how to apply and adapt individual patterns.

Globalization. *Global software development* is an emerging paradigm shift towards globally distributed development teams [174]. The shift is motivated by the desire to capitalize on global resource pools, decrease costs, exploit a 24-hour work day, and be geographically closer to the end-consumer [117]. The downside is increased risk of communication gaps. For example, elicitation and early modeling are collaborative activities that require the construction of a shared mental model of the problem and requirements. However, there is an explicit disconnect between this need for collaboration and the distance imposed by global development.

Globalization poses two main challenges to the RE research community. First, new or extended RE techniques are needed to support outsourcing of downstream development tasks, such as design, coding, and testing. Distance aggravates the gap between the requirements and development teams, particularly if the teams are from different organizations, have different cultures, or have different work environments. In particular, because geographic distance reduces team communication [175], ill-defined requirements are at risk of ultimately being misinterpreted, resulting in a system that does not address meet the stakeholders' needs. In a preliminary effort to narrow communication gaps, Bhat et al. [117] have proposed a framework based on a people-process-technology paradigm that describes best practices for negotiating goals, culture, processes, and responsibilities across a global organization.

The second challenge is to enable effective distributed RE. Future requirements activities will be globally distributed, since requirements analysts will likely be working with geographically distributed stakeholders and distributed development teams may work with in-house customers. As such, practitioners need techniques to facilitate and manage distributed requirements elicitation, distributed modeling, distributed requirements negotiation, and the management of distributed teams – not just geographically distributed, but distributed in terms of time zone, culture, and language. Therefore, global software development further motivates the need for mature, easy to use RE techniques and reusable artifacts annotated with cost metrics, all of which will help global RE teams make better RE decisions and generally improve the overall RE process. Furthermore, with the geographical distribution of RE teams, it is even more important to have a well-defined foundation for how to integrate the various RE artifacts to ensure consistency and traceability. In addition to improving existing RE techniques for easier use, it is also necessary to extend RE techniques to explicitly overcome the challenges posed by the physical distribution of RE teams. For example, Damian and her group are interested in distributed requirements negotiation, and have investigated how best to use and combine different media technology to facilitate negotiations and quality

agreements [136, 176]. Sinha et al. have developed an Eclipse-based tool for distributed requirements engineering collaboration [117].

5.2 Assurance

Given the increasing pervasiveness of computing-based systems in high-assurance systems, where errors can lead to loss or injury to life, loss of property, and/or financial loss, more attention is being paid to requirements-related tasks that explicitly address the assurance of a system. The increasing industrial demand for RE support will stimulate more research activities, including those overviewed in Table 3. Given the critical nature of many computing-based systems, RE techniques that implicitly or explicitly improve the assurance of the overall system are particularly crucial for the RE community to investigate. For example, most validation and verification techniques used to assess correctness, satisfiability, and other assurance properties, such as model checking and model satisfiability are already hitting their scalability limits with current systems. With the added complexity and size of future systems, more research is needed to make the current analysis techniques more scalable to current and future systems. Modular verification [177], harnessing parallel processing power [178, 179] and bio-inspired approaches [180] for model checking are all examples of research in this direction. Below we highlight a few challenges of two key assurance areas: security and fault tolerance. In both cases, significant effort has been expended in developing techniques to be applied during design and/or implementation; furthermore, these techniques tend to be more reactive and based on known threats (e.g., failures). Given the increasing demand for security and fault tolerance for an ever-widening number of application domains, it is paramount for sufficiently rigorous RE techniques to be developed to support these areas to most effectively leverage the design and implementation techniques that have been and will continue to be developed. In addition, exploring more effective RE techniques for these areas may uncover or open up opportunities for previously unknown solution options.

Security. As computing systems become ever more pervasive and mobile, and as they automate and manage more consumer-critical processes and data, they increasingly become the targets of security attacks. Because the sources of security threats are mostly external to the software system, we elevate security above other nonfunctional requirements as one that poses additional challenges to RE.

Substantial work has been done on how to improve software security, in the form of solutions and strategies to avoid vulnerabilities, to protect systems and information, and to defend against or recover from attacks. However, most of these solutions are threat-specific, and are targeted more for the design or implementation stages. Thus, the RE challenge for developing secure systems is to identify potential security threats, so that designers can select and apply appropriate protections. This task involves significant study, modeling, and analysis of the environment in which the system will operate, and so far there has been little work on domain modeling – despite the fact that its importance was raised almost 15 years ago [181].

Moreover, there is no consensus on how security requirements themselves should be documented. Is security a nonfunctional requirement to be resolved and optimized at

design time along with other competing nonfunctional requirements? Or should security requirements be realized as functional requirements, in the manner that user interfaces and timing deadlines are woven into behavioral specifications? These are open questions for the modeling, analysis, and security communities to resolve.

Tolerance. Software is increasingly used to automate critical applications and services, such as transportation vehicles and systems, financial decisions and transactions, medical care, military command and control, and so on; in which security and assurance requirements are paramount. However, given the complexity of such systems, with respect to size, decentralized decision-making, and variability, the SE and RE communities may need to soften their views and expectations for security and correctness. Shaw [182] discusses the need to accept "sufficient correctness" for complex systems, instead of striving for absolute correctness that may lead to brittle systems.

> **Sufficient Correctness:** *The degree to which a system must be dependable in order to serve the purpose its user intends, and to do so well enough to satisfy the current needs and expectations of those users [182].*

When operating in an uncertain and dynamically changing environment, brittle systems tend to fail at the first encounter of adverse conditions. To avoid this problem, requirements elicitation should focus on requirements for acceptable behavior and on what it means for a system to be "healthy" [182]. One approach to relaxing the precision of correctness criteria is to specify (fault) tolerance requirements, which extend the ranges of acceptable behavior. For example, Wassyng et al. [183] have made some preliminary progress on specifying timing requirements in a way that is precise and yet captures allowable tolerances. Alternative approaches include focusing on negative requirements, which represent "unhealthy" conditions or behaviors that the system must avoid, and on requirements for diagnostic and recovery mechanisms.

5.3   Emerging Needs

A major factor that will pose significant challenges to the RE community specifically, and to the SE field generally, is the level of *uncertainty* that will be an inherent part of many future systems. At run time, future systems will need to handle uncertainty in many forms, ranging from unexpected user input, unanticipated environmental conditions (e.g., power outages, security threats, noisy wireless networks, etc.), heterogeneity of devices and the need for interoperability, and on-demand, context-dependent services. Uncertainty at run time will make it difficult to apply traditional RE techniques that are typically based on knowledge known at development time. Below we describe three key areas of research that call attention to the challenges posed to the RE research community due to the level of uncertainty of future systems: increasing scale on multiple dimensions, tight integration between computing systems and the changing physical environment, and the need for systems to be self-managing and self-configuring, while maintaining assurance constraints.

Scale. Software systems are growing in size. Moreover, the "scale" of large-scale systems no longer refers simply to significant size, as in lines of code. Scale factors also

include complexity, degree of heterogeneity, number of sensors, and number of decentralized decision-making nodes. Yet another scale factor is variability, as software systems need to accommodate increasingly larger sets of requirements that vary with respect to changes in the software's environment. An example class of systems that exhibit many of these new scale factors are the ultra-large-scale (ULS) systems [184] proposed for next-generation military command and control systems. Other potential ULS systems include future intelligent transportation-management systems, critical infrastructure protection systems (e.g., systems managing power grids, bridges, telecommunication systems), integrated health-providing systems, and disaster-response systems.

Modeling, abstraction, and analysis techniques that scale well are critical in designing future ULSs. Current modeling paradigms and analysis techniques cannot effectively manage the degrees of scale, complexity, variability, and uncertainty that are exhibited in these systems. Requirements will come from many different stakeholders, involve multiple disciplines (e.g., sensors, scientific computation, artificial intelligence), and perhaps be presented at varying levels of abstraction. Thus, new abstractions, innovative decomposition strategies, standardized composition operators, and increased automation of RE tasks are all needed to cope with the complexity. In addition, better techniques are needed to merge potentially vastly different types of requirements into a single coherent story, whereas new techniques need to be developed to infer requirements from legacy components. New techniques to support responsibility and functionality tradeoff analysis for ULS systems are needed, given that these systems will, by definition, be distributed and running atop heterogeneous platforms. For these systems, neither the qualitative framework from Mylopoulos *et al* [98], nor the probabilistic framework by Letier and van Lamsweerde [99] are sufficient [185], but both provide insight as to the key factors that need to be considered when evaluating alternative options. The challenge will be how to factor in many more stakeholder needs, orders of magnitude more options to consider involving complex dependencies, with functional and performance-based constraints. Detecting and resolving feature interactions and conflicts on such a large scale will pose a grand challenge. Taken together, these problems call for new paradigms for thinking about, modeling, analyzing, and managing requirements.

Increased Reliance on the Environment. The increase in scale is partly due to the rise of systems of systems, consisting of software, hardware, and people, all of which may be loosely or tightly coupled together. For example, *cyber-physical systems* (CPSs) are a new generation of engineered systems in which computing and communication are tightly coupled with the monitoring and control of entities in the physical world [162]. Example cyber-physical systems include intelligent transportation and vehicle systems; automated manufacturing; critical infrastructure monitoring; disaster response; optimization of energy consumption; smart wearable attire [186] for health care, personal safety, and medical needs; ecosystem monitoring [187]; and efficient agriculture [162].

Integrated systems pose particularly thorny requirements problems because of their coupling with and dependence on the physical environment. Such systems recast old RE problems of determining the software system's boundary into more complicated problems of assigning responsibilities: to the software system under consideration, to peer software systems, to hardware interface devices (which are increasingly

programmable), and to human operators and users [12]. Moreover, the environment or context in which a software system will run is often the least understood and most uncertain aspect of a proposed system; and RE technologies and tools for reasoning about the integration of physical environment, interface devices, and software system are among the least mature.

In an integrated system, the correctness of a software system depends heavily on its interactions with peer software systems, hardware interfaces, and the physical and human environment. To reason about such a software system, it becomes necessary to formalize the properties of the environments with which the software will interoperate. Jackson [188] explores a number of challenges in modeling and reasoning about a software system's environment, including working with formalizations that are necessarily "imperfect...[discrete approximations] of continuous phenomena", devising piecewise formalizations of the environment to support different proofs about the software, and ensuring that the environment is in a "compatible state" when the system initializes. [188]. Towards this end, better abstractions are needed to model the behaviors of physical and human entities and their interfaces with computing elements. New domain-specific languages may be needed to express these domain abstractions and knowledge; and new languages would call for corresponding simulation, verification, and visualization techniques, to validate the modeled environment.

Most importantly, there need to be better techniques for integrating models of the environment, interface devices, and software components. Computing devices tend to be modeled using discrete mathematics, such as logics and automata; physical devices tend to modeled using continuous mathematics, such as differential equations; and human-behavior modeling is an open problem, with researchers using a combination of goals, agents, relationship models, and performance moderator functions [189]. Researchers in the verification community are making progress on the modeling, simulation, and reasoning of hybrid models [190], but their work does not accommodate human-behavior models, and the scalability of techniques remains an elusive goal.

Self-Management. The difficulties of requirements engineering are made worse by the desire to create software systems that accommodate varying, uncertain, incomplete, or evolving requirements. For example, there is growing interest in *self-managing systems*, in which the software system is aware of its context and is able to react and adapt to changes in either its environment or its requirements [191] – such as a mobile device, whose available services vary with the user's location and with the local service provider(s). Examples of such systems include *self-healing systems* that are able to recover dynamically from system failure, faults, errors, or security breaches; and *self-optimizing systems* that are able to optimize their performance dynamically with respect to changing operational profiles.

Self-management capabilities are essential in software systems that, once deployed, cannot be accessed physically or electronically. For example, a cyber-physical system (CPS) may have large numbers of physically distributed sensors that are placed in difficult-to-reach locations, such as power transformers, nuclear reactor cores, and hazardous or toxic sites. Moreover, these sensors are increasingly programmable. If developers are not able to access remote elements to perform software updates, then the elements will need to update and correct themselves.

In the simplest case, a self-managing system adapts its behavior at run-time by replacing the running system with a new target behavior selected from among a set of predefined behaviors. These systems will require different perspectives on what types of requirements information should be documented, in contrast to traditional approaches, which typically focus on a static set of goals or functionality. The RE research problems posed by such a system include

- Identifying and specifying thresholds for when the system should adapt
- Specifying variable sets of requirements
- Identifying correctness criteria for adaptive systems
- Verifying models of adaptive systems and their sets of possible behaviors
- Monitoring the system and environment against the current requirements

There has been some preliminary work on modeling and verifying dynamic architectures [192, 193, 194, 195], on specifying adaptive software [196, 197], and on run-time monitoring [198, 199, 200].

However, the above approach assumes that it is possible to predict and predefine the requirements for a complete set of target behaviors. Such predictions may not be possible if the system is to recover dynamically from unexpected errors or attacks, or is to adapt at run-time to new environmental conditions or to new requirements that were not anticipated during development. In this case, what is needed is a *self-evolving* system that is able, at run-time, to derive new requirements and behaviors from either newly synthesized components or retrieved existing components (thus necessitating support for run-time matching of requirements-indexed components). Neither of those requirements-driven approaches to run-time behavior adaptation are possible with current RE techniques. Once assurance is factored into the adaptation process, the requirements analyst needs to specify how the system's requirements can evolve dynamically; specify abstract adaptation thresholds that allow for uncertainty and unanticipated environmental conditions; and verify the requirements-decision capabilities of the resulting system. Unfortunately, none of the existing modeling and verification techniques address the challenges posed by evolution, uncertainty, and incomplete information.

One research strategy would be to investigate whether ideas from other disciplines could be leveraged. For example, recent work by Kramer and Magee [191] and Garlan [201] leverage the control loop decision-making strategy used in control-based systems. Others have explored how inspiration from biological systems can be used to introduce innovative approaches to adaptive systems. Given that natural organisms are inherently able to respond to adverse and unexpected conditions, biological entities and systems may be suitable metaphors for dynamically adaptive software. *Biomimetics* comprises those techniques that attempt to imitate or simulate the behavior of natural organisms. For example, Sutcliffe and Maiden's work [151] in establishing a domain theory for RE draws heavily from cognitive science and the human use of analogical reasoning. As systems become larger, more complex, and more tightly integrated into consumer products, the role of cognitive science and psychology will be essential to understand how wide ranges of users will interact with sophisticated and "intelligent" systems. Work has been done to use biological inspiration to guide the specification and implementation of behavior of adaptive systems. One example is The NASA Autonomous Nano Technology Swarm (ANTS) project involves large collections (i.e.,

swarms) of mission-independent small, autonomous, self-similar, reconfigurable robot-like entities that can collectively perform mission-specific tasks [202], such as exploration of planetary surfaces. The general objective is to understand and then mimic the behavior of social insect colonies that can perform relatively sophisticated tasks based on efficient social interaction and coordination. Research into the requirements for the physical structure of these drones (member of the swarm) has also been biologically-inspired, including the tetrahedral shape and telescoping "tentacles." Evolutionary computations, such as genetic algorithms, evolve by using feedback from previous computations to attempt improvements to future computations. Another area of evolutionary computation is *digital evolution* that studies how a population of self-replicating computer programs that exist in a user-defined computational environment and are subject to mutations and natural selection can produce interesting behavior under resource constraints and possibly adverse environmental conditions. Recently, researchers have been exploring how digital evolution techniques can be extended to simulate a biological evolution process that discovers new unanticipated behavior [203], and thus new requirements, for potential target systems of dynamically adaptive systems [204,205,206,207].

6 Recommendations and Conclusions

In this paper, we have described a number of exciting and challenging research directions in requirements engineering. Some of these directions are natural extensions of work already being performed by RE researchers, whereas others are major discontinuities due to fundamental changes in computing needs. All of the problems described above will require substantial effort in order to make progress towards effective solutions. To help alleviate this effort, we offer some recommendations of short- and long-term actions that the RE community could take, to position itself to make more rapid progress on these research problems.

There are five recommendations that the RE community could take immediate action on, to start improving the maturity of current requirements technologies:

- Researchers should work with practitioners. Such partnerships can help to ensure that researchers have a through understanding of the real problems that practitioners face.
- RE researchers should work with other SE researchers and practitioners, to establish stronger links between their respective artifacts. If the transition between RE tasks and other development tasks were more seamless, management would view RE efforts more positively, because the resulting requirements knowledge and artifacts would make more concrete progress towards achieving downstream milestones.
- RE researchers should not neglect evaluation and empirical research. For practitioners to consider adopting a given research technique, they need to know how the technique compares with other similar techniques. Also, practitioners and their managers need to see that the technique can be applied to problems relevant to their organization, from both domain and scale perspectives.
- Industrial organizations should provide (sanitized) industrial-strength project data to researchers. It is especially critical that industry provide realistic data for

ultra-large scale or cyber-physical systems to ensure that researchers tackle problems that are representative of those faced by practitioners. Researchers can use this data to guide the development and validation of their new techniques, thereby yielding more relevant and useful research results that explicitly address industrial needs.
– RE researchers and practitioners, together, should establish repositories of RE artifacts. Such repositories can serve as a resource for practitioners and educators to share best practices and exemplar artifacts. Repositories can also store requirements patterns for potential reuse, case studies that evaluate individual or composite RE techniques, benchmark data for evaluating competing technologies, and tools that support specific RE techniques.

The above actions would help the RE research community to make immediate progress on improving existing knowledge and techniques. In addition, there are some longer-term actions that would help to improve the community's research infrastructure and its ability to confront the challenges posed by emerging systems:

– The RE community needs to be proactive in identifying the RE research problems that arise from new computing challenges. New challenges reflect changing stakeholders' needs. As such, RE researchers should be involved in the initial investigations of any new computing challenge, to help tease out the essential goals and to assess their impact on RE tasks.
– RE researchers and practitioners should form strategic partnerships. (This need is so critical that we felt it was worth repeating.) As partners, researchers and practitioners can collaborate on a shared vision of the important research problems, the formulation of viable solutions, and the transfer of these innovative solutions into practice. They can also develop a shared understanding of the limits of current technologies in addressing current and future problems.
– Researchers need to think beyond current RE and SE knowledge and capabilities, in order to make significant headway in addressing the challenges posed by emerging systems. They need to be willing to search for new solutions that may lead to paradigm shifts in RE practices, at the risk of possible failure.
– RE researchers should seek out collaborators from other disciplines to leverage successful techniques that could be used to address analogous challenges faced by cyber systems.
– RE academics need to educate the next generation of developers on RE problems and technologies. Students need curricula that combine the study of computing with the study of specialized application domains. They also need computing courses that teach them how to make design decisions that achieve requirements (e.g., modularity vs. performance requirements) in the context of the software's operating environment.

In conclusion, the RE research community has made significant progress along many fronts. At the same time, the demands placed on computing and the cyberinfrastructure have increased dramatically, raising many new critical RE research questions. For these reasons, it is an exciting time to be involved in RE research. Technologies that make significant advances in solving these problems are likely to lead to paradigm shifts that

will impact many future generations of developers, computing systems, and the ultimate stakeholders – consumers.

Acknowledgements

References

1. Nuseibeh, B., Easterbrook, S.: Requirements engineering: a roadmap. In: Proc. of the IEEE Int. Conf. on Soft. Eng. (ICSE), pp. 35–46 (2000)
2. Parnas, D.L.: Software engineering programmes are not computer science programmes. Ann. Soft. Eng. 6(1), 19–37 (1999)
3. Finkelstein, A. (ed.): The Future of Software Engineering. In: Proc. of the IEEE Int. Conf. on Soft. Eng. (ICSE) (2000)
4. van Lamsweerde, A.: Requirements engineering in the year 00: a research perspective. In: Proc. of the IEEE Int. Conf. on Soft. Eng. (ICSE), pp. 5–19 (2000)
5. Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J., Perini, A.: TROPOS: an agent-oriented software development methodology. J. of Auto. Agents and Multi-Agent Sys. 8(3), 203–236 (2004)
6. Leveson, N.G.: Intent specifications: An approach to building human-centered specifications. IEEE Trans. on Soft. Eng. 26(1), 15–35 (2000)
7. van Lamsweerde, A.: Goal-oriented requirements engineering: a guided tour. In: Proc. of the IEEE Int. Req. Eng. Conf. (RE), pp. 249–263 (2001)
8. Cockburn, A.: Writing Effective Use Cases. Addison-Wesley, Reading (2001)
9. Breaux, T.D., Antón, A.I.: Analyzing goal semantics for rights, permissions, and obligations. In: Proc. of the IEEE Int. Req. Eng. Conf. (RE), pp. 177–188 (2005)
10. Alfonso, A., Braberman, V., Kicillof, N., Olivero, A.: Visual timed event scenarios. In: Proc. of the IEEE Int. Conf. on Soft. Eng. (ICSE), pp. 168–177 (2004)
11. Damm, W., Harel, D.: LSCs: Breathing life into message sequence charts. Form. Meth. in Sys. Des. 19(1), 45–80 (2001)
12. Letier, E., van Lamsweerde, A.: Agent-based tactics for goal-oriented requirements elaboration. In: Proc. of the IEEE Int. Conf. on Soft. Eng. (ICSE), pp. 83–93 (2002)
13. Yu, E.: Agent orientation as a modelling paradigm. Wirtschaftsinformatik 43(3), 123–132 (2001)

14. Sindre, G., Opdahl, A.: Templates for misuse case description. In: Proc. of the Int. Work. on Req. Eng.: Found. for Soft. Qual., pp. 125–136 (2001)
15. Uchitel, S., Kramer, J., Magee, J.: Negative scenarios for implied scenario elicitation. In: Proc. of ACM SIGSOFT Found. on Soft. Eng. (FSE), pp. 109–118 (2002)
16. van Lamsweerde, A.: Elaborating security requirements by construction of intentional anti-models. In: Proc. of the IEEE Int. Conf. on Soft. Eng. (ICSE), pp. 148–157 (2004)
17. Chung, L., Nixon, B., Yu, E., Mylopoulos, J.: Non-functional Requirements in Software Engineering. Kluwer, Dordrecht (1999)
18. Gilb, T.: Competitive Engineering: A Handbook for System Engineering, Requirements Engineering, and Software Engineering using Planguage. Butterworth-Heinemann (2005)
19. Sharp, H., Finkelstein, A., Galal, G.: Stakeholder identification in the requirements engineering process. In: Proc. of the 10th Int. Work. on Datab. & Exp. Sys. Appl., pp. 387–391 (1999)
20. Pisan, Y.: Extending requirement specifications using analogy. In: Proc. of the IEEE Int. Conf. on Soft. Eng. (ICSE), pp. 70–76 (2000)
21. Potts, C.: Metaphors of intent. In: Proc. of the IEEE Int. Req. Eng. Conf. (RE), pp. 31–39 (2001)
22. Aoyama, M.: Persona-and-scenario based requirements engineering for software embedded in digital consumer products. In: Proc. of the IEEE Int. Req. Eng. Conf. (RE), pp. 85–94 (2005)
23. Cooper, A.: The Inmates are Running the Asylum. Sams (1999)
24. Cohene, T., Easterbrook, S.: Contextual risk analysis for interview design. In: Proc. of the IEEE Int. Req. Eng. Conf. (RE), pp. 95–104 (2005)
25. Sutcliffe, A., Fickas, S., Sohlberg, M.M.: PC-RE a method for personal and context requirements engineering with some experience. Req. Eng. J. 11(3), 1–17 (2006)
26. Maiden, N., Robertson, S.: Integrating creativity into requirements processes: Experiences with an air traffic management system. In: Proc. of the IEEE Int. Req. Eng. Conf. (RE), pp. 105–116 (2005)
27. Heitmeyer, C.L., Kirby, J., Labaw, B.G., Bharadwaj, R.: SCR*: A toolset for specifying and analyzing software requirements. In: Y. Vardi, M. (ed.) CAV 1998. LNCS, vol. 1427, pp. 526–531. Springer, Heidelberg (1998)
28. Magee, J., Pryce, N., Giannakopoulou, D., Kramer, J.: Graphical animation of behavior models. In: Proc. of the IEEE Int. Conf. on Soft. Eng. (ICSE), pp. 499–508 (2000)
29. Van, H.T., van Lamsweerde, A., Massonet, P., Ponsard, C.: Goal-oriented requirements animation. In: Proc. of the IEEE Int. Req. Eng. Conf. (RE), pp. 218–228 (2004)
30. Thompson, J.M., Heimdahl, M.P.E., Miller, S.P.: Specification-based prototyping for embedded systems. In: Nierstrasz, O., Lemoine, M. (eds.) ESEC 1999 and ESEC-FSE 1999. LNCS, vol. 1687, pp. 163–179. Springer, Heidelberg (1999)
31. Jeffords, R., Heitmeyer, C.: Automatic generation of state invariants from requirements specifications. In: Proc. of ACM SIGSOFT Found. on Soft. Eng. (FSE), pp. 56–69 (1998)
32. Jackson, D.: Software Abstractions: Logic, Language, and Analysis. MIT Press, Cambridge (2006)
33. Yu, E.: Towards modelling and reasoning support for early-phase requirements engineering. In: Proc. of the IEEE Int. Req. Eng. Conf. (RE), pp. 226–235 (1997)
34. Janicki, R., Parnas, D.L., Zucker, J.: Tabular representations in relational documents. Relational methods in computer science, 184–196 (1997)
35. Uchitel, S., Kramer, J., Magee, J.: Behaviour model elaboration using partial labelled transition systems. In: Proc. of ACM SIGSOFT Found. on Soft. Eng. (FSE), pp. 19–27 (2003)
36. Arimoto, Y., Nakamura, M., Futatsugi, K.: Toward a domain description with CafeOBJ. In: Proc. 23rd JSSST Convention (2006)

37. Bellini, P., Mattolini, R., Nesi, P.: Temporal logics for real-time system specification. ACM Comp. Sur. 32(1), 12–42 (2000)
38. Dillon, L.K., Kutty, G., Moser, L.E., Melliar-Smith, P.M., Ramakrishna, Y.S.: A graphical interval logic for specifying concurrent systems. ACM Trans. on Soft. Eng. & Meth. 3(2), 131–165 (1994)
39. Letier, E., Kramer, J., Magee, J., Uchitel, S.: Fluent temporal logic for discrete-time event-based models. In: Proc. of ACM SIGSOFT Found. on Soft. Eng. (FSE), pp. 70–79 (2005)
40. Evans, A., Bruel, J.M., France, R., Lano, K., Rumbpe, B.: Making UML precise. In: Proc. of the OOPSLA Work. on Formalizing UML. Why? How? (1998)
41. McUmber, W.E., Cheng, B.H.: A general framework for formalizing UML with formal languages. In: Proc. of the IEEE Int. Conf. on Soft. Eng. (ICSE), pp. 433–442 (2001)
42. Niu, J., Atlee, J., Day, N.: Template semantics for model-based systems. IEEE Trans. on Soft. Eng. 29(10), 866–882 (2003)
43. Taleghani, A., Atlee, J.M.: Semantic variations among UML statemachines. In: ACM/IEEE Int. Conf. on Model Driven Eng. Lang. and Sys., pp. 245–259 (2006)
44. Jackson, M.: Problem Frames. Addison-Wesley, Reading (2003)
45. Gunter, C.A., Gunter, E.L., Jackson, M., Zave, P.: A reference model for requirements and specifications. IEEE Soft. 17(3), 37–43 (2000)
46. Hall, J., Rapanotti, L.: A reference model for requirements engineering. In: Proc. of the IEEE Int. Req. Eng. Conf. (RE), pp. 181–187 (2003)
47. Parnas, D.L., Madey, J.: Functional documents for computer systems. Sci. of Comp. Prog. 25(1), 41–61 (1995)
48. Letier, E., van Lamsweerde, A.: Deriving operational software specifications from system goals. In: Proc. of ACM SIGSOFT Found. on Soft. Eng. (FSE), pp. 119–128 (2002)
49. Rashid, A., Sawyer, P., Moreira, A., Araujo, J.: Early aspects: a model for aspect-oriented requirements engineering. In: Proceedings of IEEE Joint International Conference on Requirements Engineering, pp. 199–202 (2002)
50. Baniassad, E., Clarke, S.: Theme: An approach for aspect-oriented analysis and design. In: Proc. of the IEEE Int. Conf. on Soft. Eng. (ICSE), pp. 158–167 (2004)
51. Uchitel, S., Kramer, J., Magee, J.: Incremental elaboration of scenario-based specifications and behavior models using implied scenarios. ACM Trans. on Soft. Eng. & Meth. 13(1), 37–85 (2004)
52. Nuseibeh, B., Kramer, J., Finkelstein, A.: Viewpoints: meaningful relationships are difficult! In: Proc. of the IEEE Int. Conf. on Soft. Eng. (ICSE), pp. 676–683 (2003)
53. Silva, A.: Requirements, domain and specifications: A viewpoint-based approach to requirements engineering. In: Proc. of the IEEE Int. Conf. on Soft. Eng. (ICSE), pp. 94–104 (2002)
54. Denger, C., Berry, D.M., Kamsties, E.: Higher quality requirements specifications through natural language patterns. In: Proc. of the IEEE Int. Conf. on Soft.-Sci., Tech. & Eng., pp. 80–90 (2003)
55. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: Proc. of the IEEE Int. Conf. on Soft. Eng. (ICSE), pp. 411–420 (1999)
56. Konrad, S., Cheng, B.H.C., Campbell, L.A.: Object analysis patterns for embedded systems. IEEE Trans. on Soft. Eng. 30(12), 970–992 (2004)
57. van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distributed and Parallel Databases 14(1), 5–51 (2003)
58. Ambriola, V., Gervasi, V.: Processing natural language requirements. In: IEEE Int. Conf. on Aut. Soft. Eng., pp. 36–45 (1997)
59. Cobleigh, R.L., Avrunin, G.S., Clarke, L.A.: User guidance for creating precise and accessible property specifications. In: Proc. of ACM SIGSOFT Found. on Soft. Eng. (FSE), pp. 208–218 (2006)

60. Goldin, L., Berry, D.: Abstfinder, a prototype abstraction finder for natural language text for use in requirements elicitation: Design, methodology, and evaluation. In: Proceedings of the First International Conference on Requirements Engineering, pp. 19–21 (April 1994)
61. Konrad, S., Cheng, B.H.: Facilitating the construction of specification pattern-based properties. In: Proc. of the IEEE Int. Req. Eng. Conf. (RE), pp. 329–338 (2005)
62. Overmyer, S.P., Lavoie, B., Rambow, O.: Conceptual modeling through linguistic analysis using LIDA. In: Proc. of the IEEE Int. Conf. on Soft. Eng. (ICSE), pp. 401–410 (2001)
63. Fuxman, A., Liu, L., Pistore, M., Roveri, M., Mylopoulos, J.: Specifying and analyzing early requirements: Some experimental results. In: Proc. of the IEEE Int. Req. Eng. Conf. (RE), pp. 105–114 (2003)
64. Berenbach, B.: The evaluation of large, complex UML analysis and design models. In: Proc. of the IEEE Int. Conf. on Soft. Eng. (ICSE), pp. 232–241 (2004)
65. Sabetzadeh, M., Easterbrook, S.: View merging in the presence of incompleteness and inconsistency. Req. Eng. J. 11(3), 174–193 (2006)
66. Uchitel, S., Chechik, M.: Merging partial behavioural models. In: Proc. of ACM SIGSOFT Found. on Soft. Eng. (FSE), pp. 43–52 (2004)
67. Alur, R., Etessami, K., Yannakakis, M.: Inference of message sequence charts. In: Proc. of the IEEE Int. Conf. on Soft. Eng. (ICSE), pp. 304–313 (2000)
68. Damas, C., Lambeau, B., van Lamsweerde, A.: Scenarios, goals, and state machines: a win-win partnership for model synthesis. In: Proc. of ACM SIGSOFT Found. on Soft. Eng. (FSE), pp. 197–207 (2006)
69. Uchitel, S., Kramer, J., Magee, J.: Synthesis of behavioral models from scenarios. IEEE Trans. on Soft. Eng. 29(2), 99–115 (2003)
70. Whittle, J., Schumann, J.: Generating statechart designs from scenarios. In: Proc. of the IEEE Int. Conf. on Soft. Eng. (ICSE), pp. 314–323 (2000)
71. Hay, J.D., Atlee, J.M.: Composing features and resolving interactions. In: Proc. of ACM SIGSOFT Found. on Soft. Eng. (FSE), pp. 110–119 (2000)
72. Kaiya, H., Saeki, M.: Using domain ontology as domain knowledge for requirements elicitation. In: Proc. of the IEEE Int. Req. Eng. Conf. (RE), pp. 186–195 (2006)
73. In, H., Rodgers, T., Deutsch, M., Boehm, B.: Applying WinWin to quality requirements: A case study. In: Proc. of the IEEE Int. Conf. on Soft. Eng. (ICSE), pp. 555–564 (2001)
74. Alves, C., Finkelstein, A.: Challenges in COTS decision-making: a goal-driven requirements engineering perspective. In: Proc. of the Int. Con. on Soft. Eng. and Know. Eng., pp. 789–794 (2002)
75. Rolland, C., Prakash, N.: Matching ERP system functionality to customer requirements. In: Proc. of the IEEE Int. Req. Eng. Conf. (RE), pp. 66–75 (2001)
76. Robinson, W.N., Pawlowski, S.D., Volkov, V.: Requirements interaction management. ACM Comp. Sur. 35(2), 132–190 (2003)
77. Potts, C., Takahashi, K., Antón, A.: Inquiry-based requirements analysis. IEEE Soft. 11(2), 21–32 (1994)
78. Regnell, B., Karlsson, L., Höst, M.: An analytical model for requirements selection quality evaluation in product software development. In: Proc. of the IEEE Int. Req. Eng. Conf. (RE), pp. 254–263 (2003)
79. Fagan, M.: Advances in software inspections. IEEE Trans. on Soft. Eng. 12(7), 744–751 (1986)
80. Parnas, D.L., Weiss, D.M.: Active design reviews: principles and practices. J. Sys. Soft. 7(4), 259–265 (1987)
81. Wasson, K.S., Schmid, K.N., Lutz, R.R., Knight, J.C.: Using occurrence properties of defect report data to improve requirements. In: Proc. of the IEEE Int. Req. Eng. Conf. (RE), pp. 253–262 (2005)

82. Berry, D., Kamsties, E.: 2. In: Ambiguity in Requirements Specification. Perspectives on Software Requirements. Kluwer Academic Publishers, Dordrecht (2004)
83. Chantree, F., Nuseibeh, B., de Roeck, A., Willis, A.: Identifying nocuous ambiguities in natural language requirements. In: Proc. of the IEEE Int. Req. Eng. Conf. (RE), pp. 59–68 (2006)
84. Wasson, K.S.: A case study in systematic improvement of language for requirements. In: Proc. of the IEEE Int. Req. Eng. Conf. (RE), pp. 6–15 (2006)
85. Engels, G., Küster, J.M., Heckel, R., Groenewegen, L.: A methodology for specifying and analyzing consistency of object-oriented behavioral models. In: Proc. of ACM SIGSOFT Found. on Soft. Eng. (FSE), pp. 186–195 (2001)
86. Heitmeyer, C.L., Jeffords, R.D., Labaw, B.G.: Automated consistency checking of requirements specifications. ACM Trans. on Soft. Eng. & Meth. 5(3), 231–261 (1996)
87. Nentwich, C., Emmerich, W., Finkelstein, A., Ellmer, E.: Flexible consistency checking. ACM Trans. on Soft. Eng. & Meth. 12(1), 28–63 (2003)
88. Carlshamre, P., Sandahl, K., Lindvall, M., Regnell, B., Dag, J.N.: An industrial survey of requirements interdependencies in software product release planning. In: Proc. of the IEEE Int. Req. Eng. Conf. (RE), pp. 84–93 (2001)
89. Hausmann, J.H., Heckel, R., Taentzer, G.: Detection of conflicting functional requirements in a use case-driven approach. In: Proc. of the IEEE Int. Conf. on Soft. Eng. (ICSE), pp. 105–115 (2002)
90. Lutz, R., Patterson-Hine, A., Nelson, S., Frost, C.R., Tal, D., Harris, R.: Using obstacle analysis to identify contingency requirements on an unpiloted aerial vehicle. Req. Eng. J. 12(1), 41–54 (2006)
91. van Lamsweerde, A., Letier, E.: Handling obstacles in goal-oriented requirements engineering. IEEE Trans. on Soft. Eng. 26(10), 978–1005 (2000)
92. Feather, M.S.: Towards a unified approach to the representation of, and reasoning with, probabilistic risk information about software and its system interface. In: Int. Sym. on Soft. Reliab. Eng., pp. 391–402 (2004)
93. Wyatt, V., DiStefano, J., Chapman, M., Aycoth, E.: A metrics based approach for identifying requirements risks. In: Software Engineering Workshop, p. 23 (2003)
94. Krishnamurthi, S., Tschantz, M.C., Meyerovich, L.A., Fisler, K.: Verification and change-impact analysis of access-control policies. In: Proc. of the IEEE Int. Conf. on Soft. Eng. (ICSE), pp. 196–205 (2005)
95. Baker, P., Bristow, P., Jervis, C., King, D., Thomson, R., Mitchell, B., Burton, S.: Detecting and resolving semantic pathologies in UML sequence diagrams. In: Proc. of ACM SIGSOFT Found. on Soft. Eng. (FSE), pp. 50–59 (2005)
96. Moreira, A., Rashid, A., Araujo, J.: Multi-dimensional separation of concerns in requirements engineering. In: Proc. of the IEEE Int. Req. Eng. Conf. (RE), pp. 285–296 (2005)
97. Berenbach, B., Borotto, G.: Metrcs for model driven requirements development. In: IEEE International Conference on Software Engineering, Shanghai, China (2006)
98. Gonzalez-Baixauli, B., do Prado Leite, J.C.S., Mylopoulos, J.: Visual variability analysis for goal models. In: Proc. of the IEEE Int. Req. Eng. Conf. (RE), pp. 198–207 (2004)
99. Letier, E., van Lamsweerde, A.: Reasoning about partial goal satisfaction for requirements and design engineering. In: Proc. of ACM SIGSOFT Found. on Soft. Eng. (FSE), pp. 53–62 (2004)
100. Liaskos, S., Alexei, Yu, Y., Yu, E., Mylopoulos, J.: On goal-based variability acquisition and analysis. In: Proc. of the IEEE Int. Req. Eng. Conf. (RE), pp. 76–85 (2006)
101. Sutcliffe, A., Chang, W.C., Neville, R.: Evolutionary requirements analysis. In: Proc. of the IEEE Int. Req. Eng. Conf. (RE), pp. 264–273 (2003)
102. Bultan, T.: Action language: A specification language for model checking reactive systems. In: Proc. of the IEEE Int. Conf. on Soft. Eng. (ICSE), pp. 335–344 (2000)

103. Dillon, L.K., Stirewalt, R.E.K.: Inference graphs: A computational structure supporting generation of customizable and correct analysis components. IEEE Trans. on Soft. Eng. 29(2), 133–150 (2003)

104. Whittle, J., Saboo, J., Kwan, R.: From scenarios to code: An air traffic control case study. In: Proc. of the IEEE Int. Conf. on Soft. Eng. (ICSE), pp. 490–497 (2003)

105. Bush, D., Finkelstein, A.: Requirements stability assessment using scenarios. In: Proc. of the IEEE Int. Req. Eng. Conf. (RE), pp. 23–32 (2003)

106. Gregoriades, A.: Scenario-based assessment of nonfunctional requirements. IEEE Trans. Softw. Eng. 31(5), 392–409 (2005); Member-Alistair Sutcliffe

107. Ryser, J., Glinz, M.: A scenario-based approach to validating and testing software systems using statecharts. In: 12th International Conference on Software and Systems Engineering and their Applications (ICSSEA 1999), Paris, France (1999)

108. Chan, W., Anderson, R.J., Beame, P., Burns, S., Modugno, F., Notkin, D., Reese, J.D.: Model checking large software specifications. IEEE Trans. on Soft. Eng. 24(7), 498–520 (1998)

109. Easterbrook, S., Chechik, M.: A framework for multi-valued reasoning over inconsistent viewpoints. In: Proc. of the IEEE Int. Conf. on Soft. Eng. (ICSE), pp. 411–420 (2001)

110. Sreemani, T., Atlee, J.M.: Feasibility of model checking software requirements. In: Conf. on Comp. Assur., pp. 77–88. National Institute of Standards and Technology (1996)

111. Buhne, S., Lauenroth, K., Pohl, K.: Modelling requirements variability across product lines. In: Proc. of the IEEE Int. Req. Eng. Conf. (RE), pp. 41–52 (2005)

112. Czarnecki, K., Eisenecker, U.W.: Generative Programming. Addison-Wesley, Reading (2000)

113. Reiser, M.O., Weber, M.: Managing highly complex product families with multi-level feature trees. In: Proc. of the IEEE Int. Req. Eng. Conf. (RE), pp. 146–155 (2006)

114. Schmid, K.: The product line mapping approach to defining and structuring product portfolios. In: Proc. of the IEEE Int. Req. Eng. Conf. (RE), pp. 219–226 (2002)

115. Alspaugh, T.A., Antón, A.I.: Scenario networks for software specification and scenario management. Technical Report TR-2001-12, North Carolina State University at Raleigh (2001)

116. Weber, M., Weisbrod, J.: Requirements engineering in automotive development experiences and challenges. In: Proc. of the IEEE Int. Req. Eng. Conf. (RE), pp. 331–340 (2002)

117. Damian, D., Moitra, D. (eds.): Global software development. IEEE Soft. special issue 23(5) (2006)

118. Cleland-Huang, J., Zemont, G., Lukasik, W.: A heterogeneous solution for improving the return on investment of requirements traceability. In: Proc. of the IEEE Int. Req. Eng. Conf. (RE), pp. 230–239 (2004)

119. Hayes, J.H., Dekhtyar, A., Sundaram, S.K.: Advancing candidate link generation for requirements tracing: The study of methods. IEEE Trans. on Soft. Eng. 32(1), 4–19 (2006)

120. Sabetzadeh, M., Easterbrook, S.: Traceability in viewpoint merging: a model management perspective. In: Proc. of the Int. Work. on Trace. in Emerg. Forms of Soft. Eng., pp. 44–49 (2005)

121. Settimi, R., Berezhanskaya, E., BenKhadra, O., Christina, S., Cleland-Huang, J.: Goal-centric traceability for managing non-functional requirements. In: Proc. of the IEEE Int. Conf. on Soft. Eng. (ICSE), pp. 362–371 (2005)

122. Zave, P.: Classification of research efforts in requirements engineering. ACM Comp. Sur. 29(4), 315–321 (1997)

123. France, R., Rumpe, B.: Model-driven development of complex systems: A research roadmap. In: Future of Software Engineering 2007. IEEE-CS Press, Los Alamitos (2007)

124. Gogolla, M., Ziemann, P., Kuske, S.: Towards an integrated graph based semantics for UML. In: Bottoni, P., Minas, M. (eds.) Graph Transf. and Vis. Model. Tech. ENTCS, vol. 72(3). Elsevier, Amsterdam (2002)
125. Fantechi, A., Gnesi, S., Lami, G., Maccari, A.: Application of linguistic techniques for use case analysis. In: Proc. of the IEEE Int. Req. Eng. Conf. (RE), pp. 157–164 (2002)
126. Sawyer, P., Rayson, P., Cosh, K.: Shallow knowledge as an aid to deep understanding in early phase requirements engineering. IEEE Trans. on Soft. Eng. 31(11), 969–981 (2005)
127. Campbell, L.A., Cheng, B.H.C., McUmber, W.E., Stirewalt, R.E.K.: Automatically detecting and visualizing errors in UML diagrams. Req. Eng. J. 37(10), 74–86 (2002)
128. Lauesen, S.: COTS tenders and integration requirements. In: Proc. of the IEEE Int. Req. Eng. Conf. (RE), pp. 166–175 (2004)
129. Ryan, K.: The role of natural language in requirements engineering. In: Proceedings of the IEEE International Symposium on Requirements Engineering, San Diego, CA, pp. 240–242. IEEE Computer Society Press, Los Alamitos (1993)
130. Marcus, A., Maletic, J.: Recovering documentation-to-source-code traceability links using latent semantic indexing. In: Proc. of the IEEE Int. Conf. on Soft. Eng. (ICSE), pp. 125–135 (2003)
131. Damian, D.E., Zowghi, D.: The impact of stakeholders? geographical distribution on managing requirements in a multi-site organization. In: Proc. of the IEEE Int. Req. Eng. Conf. (RE), pp. 319–330 (2002)
132. Lloyd, W.J., Rosson, M.B., Arthur, J.D.: Effectiveness of elicitation techniques in distributed requirements engineering. In: Proc. of the IEEE Int. Req. Eng. Conf. (RE), pp. 311–318 (2002)
133. Maiden, N., Robertson, S.: Developing use cases and scenarios in the requirements process. In: Proc. of the IEEE Int. Conf. on Soft. Eng. (ICSE), pp. 561–570 (2005)
134. Moody, D.L., Sindre, G., Brasethvik, T., Solvberg, A.: Evaluating the quality of information models: Empirical testing of a conceptual model quality framework. In: Proc. of the IEEE Int. Conf. on Soft. Eng. (ICSE), pp. 295–307 (2003)
135. Damian, D., Chisan, J.: An empirical study of the complex relationships between requirements engineering processes and other processes that lead to payoffs in productivity, quality, and risk management. IEEE Trans. on Soft. Eng. 32(7), 433–453 (2006)
136. Damian, D., Eberlein, A., Shaw, M., Gaines, B.: An exploratory study of facilitation in distributed requirements engineering. Req. Eng. J. 8(1), 23–41 (2003)
137. Freimut, B., Hartkopf, S., Kaiser, P., Kontio, J., Kobitzsch, W.: An industrial case study of implementing software risk management. In: Proc. of ACM SIGSOFT Found. on Soft. Eng. (FSE), pp. 277–287 (2001)
138. Dong, J.S., Lee, C.H., Li, Y.F., Wang, H.: Verifying DAML+OIL and beyond in Z/EVES. In: Proc. of the IEEE Int. Conf. on Soft. Eng. (ICSE), pp. 201–210 (2004)
139. Berstel, J., Roussel, G., Reghizzi, S.C., Pietro, P.S.: A scalable formal method for design and automatic checking of user interfaces. In: Proc. of the IEEE Int. Conf. on Soft. Eng. (ICSE), pp. 453–462 (2001)
140. Davis, A., Dieste, O., Hickey, A., Juristo, N., Moreno, A.M.: Effectiveness of requirements elicitation techniques: Empirical results derived from a systematic review. In: Proc. of the IEEE Int. Req. Eng. Conf. (RE), pp. 176–185 (2006)
141. Easterbrook, S., Yu, E., Aranda, J., Fan, Y., Horkoff, J., Leica, M., Qadir, R.A.: Do viewpoints lead to better conceptual models? an exploratory case study. In: Proc. of the IEEE Int. Req. Eng. Conf. (RE), pp. 199–208 (2005)
142. Ardis, M.A., Chaves, J.A., Jagadeesan, L.J., Mataga, P., Puchol, C., Staskauskas, M.G., Olnhausen, J.V.: A framework for evaluating specification methods for reactive systems experience report. IEEE Trans. on Soft. Eng. 22(6), 378–389 (1996)

143. Porter, A., Votta, L.: Comparing detection methods for software requirements inspections: A replication using professional subjects. Empir. Soft. Eng. 3(4), 355–379 (1998)

144. Antón, A.I., Potts, C.: Functional paleontology: The evolution of user-visible system services. IEEE Trans. on Soft. Eng. 29(2), 151–166 (2003)

145. Lutz, R., Mikulski, I.C.: Operational anomalies as a cause of safety-critical requirements evolution. J. of Sys. and Soft., 155–161 (2003)

146. Shaw, M.: What makes good research in software engineering? Int. Jour. of Soft. Tools for Tech. Trans. 4(1), 1–7 (2002)

147. Redwine Jr., S.T., Riddle, W.E.: Software technology maturation. In: Proc. of the IEEE Int. Conf. on Soft. Eng. (ICSE), May 1985, pp. 1989–2001 (1985)

148. Basili, V.R.: The experimental paradigm in software engineering. In: Proc. of the Int. Work. on Exper. Soft. Eng. Issues: Crit. Assess. and Futr. Dir., pp. 3–12. Springer, Heidelberg (1993)

149. Kuhn, T.: The Nature and Necessity of Scientific Revolutions. University of Chicago Press (1962) Book chapter transcribed by Andy Blunden in 1998; proofed and corrected (March 2005)

150. Neville, R., Sutcliffe, A., Chang, W.C.: Optimizing system requirements with genetic algorithms. In: IEEE World Congress on Computational Intelligence, pp. 495–499 (May 2002)

151. Sutcliffe, A., Maiden, N.: The domain theory for requirements engineering. IEEE Trans. on Soft. Eng. 24(3), 174–196 (1998)

152. Cleland-Huang, J., Settimi, R., Duan, C., Zou, X.: Utilizing supporting evidence to improve dynamic requirements traceability. In: Proc. of the IEEE Int. Req. Eng. Conf. (RE), pp. 135–144 (2005)

153. Zave, P.: Feature interactions and formal specifications in telecommunications. IEEE Comp. 26(8), 20–29 (1993)

154. Weiss, M., Esfandiari, B.: On feature interactions among web services. In: Proc. of the IEEE Int. Conf. on Web Serv. (ICWS), pp. 88–95 (2004)

155. Dulac, N., Viguier, T., Leveson, N.G., Storey, M.A.D.: On the use of visualization in formal requirements specification. In: Proc. of the IEEE Int. Req. Eng. Conf. (RE), pp. 71–80 (2002)

156. Harel, D.: Statecharts: A visual formalism for complex systems. Sci. Comp. Prog. 8(3), 231–274 (1987)

157. Konrad, S., Cheng, B.H.C.: Real-time specification patterns. In: Proceedings of the International Conference on Software Engineering (ICSE 2005), St Louis, MO, USA, 372–381 (May 2005)

158. Antón, A.I.: Goal-based requirements analysis. In: Proc. of the IEEE Int. Req. Eng. Conf. (RE), pp. 136–144 (1996)

159. Jeffords, R.D., Heitmeyer, C.L.: A strategy for efficiently verifying requirements. In: Proc. of ACM SIGSOFT Found. on Soft. Eng. (FSE), pp. 28–37 (2003)

160. Robertson, S., Robertson, J.: Mastering the Requirements Process. Addison-Wesley, Reading (1999)

161. Glinz, M., Wieringa, R.J.: Guest editors' introduction: Stakeholders in requirements engineering. IEEE Software 24(2), 18–20 (2007)

162. Cyber-Physical Systems (October 2006), http://varma.ece.cmu.edu/cps

163. Workshop on High-Confidence Automotive Cyber-Physical Systems (April 2008), http://varma.ece.cmu.edu/Auto-CPS

164. Lero: The irish software engineering research centre, http://www.lero.ie

165. Center of excellence for research in adaptive systems, https://www.cs.uwaterloo.ca/twiki/view/CERAS

166. Center for ultra-large-scale software-intensive systems. Industry/University Collaborative Research Center, http://ulssis.cs.virginia.edu/

167. Networked european software and services initiative,
     `http://www.nessi-europe.com`
168. Basili, V.R. (ed.): Empirical Software Engineering: An International Journal
169. Damian, D., Chisan, J., Vaidyanathasamy, L., Pal, Y.: Requirements engineering and down-
     stream software development: Findings from a case study. Empirical Soft. Eng. 10(3), 255–
     283 (2005)
170. Damian, D., Zowghi, D., Vaidyanathasamy, L., Pal, Y.: An industrial case study of imme-
     diate benefits of requirements engineering process improvement at the australian center for
     unisys software. Empirical Soft. Eng. 9(1-2), 45–75 (2004)
171. Broy, M.: The 'grand challenge' in informatics: Engineering software-intensive systems.
     IEEE Computer 39(10), 72–80 (2006)
172. Ebert, C.: Understanding the product life cycle: Four key requirements engineering tech-
     niques. IEEE Soft 23(3), 19–25 (2006)
173. Pohl, K., Weyer, T.: 5. In: Documenting Variability in Requirements Artefacts. Software
     Product Line Engineering. Springer, Heidelberg (2005)
174. Herbsleb, J.D.: Global software engineering: The future of socio-technical coordination. In:
     Future of Software Engineering (2007)
175. Herbsleb, J., Mockus, A.: An empirical study of speed and communication in globally dis-
     tributed software development. IEEE Trans. on Soft. Eng. 29(6), 481–494 (2003)
176. Damian, D., Lanubile, F., Mallardo, T.: An empirical study of the impact of asynchronous
     discussions on remote synchronous requirements meetings. In: Baresi, L., Heckel, R. (eds.)
     FASE 2006. LNCS, vol. 3922, pp. 155–169. Springer, Heidelberg (2006)
177. Fisler, K., Krishnamurthi, S.: Modular verification of collaboration-based software designs.
     In: ESEC/FSE-9: Proceedings of the 8th European software engineering conference held
     jointly with 9th ACM SIGSOFT international symposium on Foundations of software en-
     gineering, pp. 152–163. ACM, New York (2001)
178. Holzmann, G.J., Bosnacki, D.: The design of a multicore extension of the spin model
     checker. IEEE Transactions on Software Engineering 33(10), 659–674 (2007)
179. Dwyer, M.B., Elbaum, S., Person, S., Purandare, R.: Parallel randomized state-space search.
     In: ICSE 2007: Proceedings of the 29th international conference on Software Engineering,
     Washington, DC, USA, pp. 3–12. IEEE Computer Society, Los Alamitos (2007)
180. Fereira, M.: Genetic algorithm for model checking concurrent programs. In: Proceedings
     of ACM Genetic And Evolutionary Computation Conference, GECCO (2008)
181. Jackson, M., Zave, P.: Domain descriptions. In: Proc. of the IEEE Int. Req. Eng. Conf. (RE),
     pp. 54–64 (1993)
182. Shaw, M.: "Self-Healing": Softening precision to avoid brittleness. In: Work. on Self-
     Healing Sys (WOSS) (November 2002) Position paper
183. Wassyng, A., Lawford, M., Hu, X.: Timing tolerances in safety-critical software. In: Proc.
     of the Int. Sym. of Form. Meth. Eur., pp. 157–172 (2005)
184. Northrup, L., Feiler, P., Gabriel, R.P., Goodenough, J., Linger, R., Longstaff, T., Kazman,
     R., Klein, M., Schmidt, D., Sullivan, K., Wallnau, K.: Ultra-Large-Scale Systems: The Soft-
     ware Challenge of the Future. Software Engineering Institute, Carnegie Mellon (2006)
185. van Lamsweerde, A.: Evaluating alternative options. Private Communication (2007)
186. Ganti, R.K., Jayachandran, P., Abdelzaher, T.F., Stankovic, J.A.: SATIRE: A softwarwe
     architecture for smart atTIRE. In: Proc. of the Inl. Conf. on Mobile Sys., Appl., and Serv
     (Mobisys), pp. 110–123 (2006)
187. Maurice, P.A., Harmon, T.C.: Environmental embedded sensor networks. Environmental
     Engineering Science 24(1) (2007); Special issue on environmental embedded sensor net-
     works
188. Jackson, M.: What can we expect from program verification? IEEE Computer 39(10), 65–
     71 (2006)

189. Silverman, B.G., Johns, M., Cornwell, J., O'Brien, K.: Human behavior models for agents in simulators and games: Part I: enabling science with PMFserv. Presence: Teleoper. Virtual Environ. 15(2), 139–162 (2006)
190. Alur, R., Henzinger, T., Lafferriere, G., Pappas, G.: Discrete abstractions of hybrid systems. Proc. of IEEE 88(7) (July 2000)
191. Kramer, J., Magee, J.: Self-managed systems: An architectural challenge. In: Future of Software Engineering (2007)
192. Allen, R., Douence, R., Garlan, D.: Specifying and analyzing dynamic software architectures. In: Astesiano, E. (ed.) FASE 1998. LNCS, vol. 1382, pp. 21–37. Springer, Heidelberg (1998)
193. Canal, C., Pimentel, E., Troya, J.M.: Specification and refinement of dynamic software architectures. In: Proc. of the TC2 First Work. IFIP Conf. on Soft. Arch. (WICSA), pp. 107–126. Kluwer, Dordrecht (1999)
194. Kramer, J., Magee, J.: Analysing dynamic change in software architectures: a case study. In: Proc. of 4th IEEE International Conference on Configurable Distributed Systems, Annapolis (May 1998)
195. Kulkarni, S., Biyani, K.: Correctness of component-based adaptation. In: Crnković, I., Stafford, J.A., Schmidt, H.W., Wallnau, K. (eds.) CBSE 2004. LNCS, vol. 3054, pp. 48–58. Springer, Heidelberg (2004)
196. Zhang, J., Cheng, B.H.C.: Specifying adaptation semantics. In: Proc. of the Work. on Architecting Depend. Sys. (WADS), pp. 1–7 (2005)
197. Zhou, Z., Zhang, J., McKinley, P.K., Cheng, B.H.C.: TA-LTL: Specifying adaptation timing properties in autonomic systems. In: Proc. of the IEEE Work. on Eng. of Auton. Sys. (EASe) (2006)
198. Fickas, S., Feather, M.S.: Requirements monitoring in dynamic environments. In: Proc. of the IEEE Int. Req. Eng. Conf. (RE), pp. 140–147 (1995)
199. Robinson, W.N.: Monitoring web service requirements. In: Proc. of the IEEE Int. Req. Eng. Conf. (RE), pp. 65–74 (2003)
200. Savor, T., Seviora, R.: An approach to automatic detection of software failures in realtime systems. In: Proc. IEEE Real-Time Tech. and Appl. Sym., pp. 136–147 (1997)
201. Garlan, D., Cheng, S.W., Huang, A.C., Schmerl, B., Steenkiste, P.: Rainbow: architecture-based self-adaptation with reusable infrastructure. IEEE Computer 37(10), 46–54 (2004)
202. Clark, P., Rilee, M.L., Curtis, S.A., Truszkowski, W., Marr, G., Cheung, C., Rudisill, M.: BEES for ANTS: Space mission applications for the autonomous nanotechnology swarm. In: AIAA 1st Intelligent Systems Technical Conference, American Institute of Aeronautics and Astronautics (2004)
203. McKinley, P.K., Cheng, B.H., Ofria, C., Knoester, D., Beckmann, B., Goldsby, H.: Harnessing digital evolution. IEEE Computer 41(1) (January 2008)
204. Goldsby, H., Knoester, D., Cheng, B.H., McKinley, P., Ofria, C.: Digitally evolving models for dynamically adaptive systems. Technical Report MSU-CSE-07-4, Computer Science and Engineering, Michigan State University, East Lansing, Michigan (January 2007)
205. Goldsby, H., Cheng, B.H.C.: Avida-MDE: Harnessing digital evolution to evolve adaptive systems. In: Proceedings of ACM Genetic And Evolutionary Computation Conference (GECCO) (2008)
206. Goldsby, H.J., Cheng, B.H., McKinley, P.K.: Digital evolution of behavioral models for autonomic systems. In: Proceedings of the 5th IEEE International Conference on Autonomic Computing, Chicago, Illinois (June 2008)
207. Goldsby, H., Cheng, B.H.C.: Harnessing digital evolution to deal with uncertainty. In: Model-Driven Engineering Languages and Systems (2008) (accepted to appear)

Requirements in the 21st Century: Current Practice and Emerging Trends[*]

Sean Hansen, Nicholas Berente, and Kalle Lyytinen

Case Western Reserve University, 10900 Euclid Avenue,
Cleveland, Ohio, USA
hansen@case.edu, berente@case.edu, kalle@case.edu

Abstract. Requirements have remained one of the grand challenges in the design of software intensive systems. In this paper we review the main strands of requirements research over the past two decades and identify persistent and new challenges. Based on a field study that involved interviews of over 30 leading IT professionals involved in large and complex software design and implementation initiatives, we review the current state-of-the-art in the practice of design requirements management. We observe significant progress in the deployment of modeling methods, tools, risk-driven design, and user involvement. We note nine emerging themes and challenges in the requirement management arena: 1) business process focus, 2) systems transparency, 3) integration focus, 4) distributed requirements, 5) layered requirements, 6) criticality of information architectures, 7) increased deployment of COTS and software components, 8) design fluidity and 9) interdependent complexity. Several research challenges and new avenues for research are noted in the discovery, specification, and validation of requirements in light of these requirements features.

Keywords: Requirements, modeling, specification, validation, verification, change, large systems, complexity, stakeholders, field study.

1 Introduction

The first step in any design effort is to ask what it is that one intends to create: What objectives does it need to address? What must it be capable of doing? Who will it serve and how? To what constraints must it conform? These questions are fundamental to design in its myriad forms – industrial design, graphic design, instructional design, and business process design, among others [1, 2]. As we know from past research and practice, software design is no different in this regard. In this paper, we refer to tasks in the design of software-intensive systems where questions of this nature are addressed as the management of design requirements.

Design requirements represent a crossroads where several research, business, engineering, and artistic communities converge. Therefore design requirements discussions span a range of research disciplines, including computer science, information

---

systems, new product development, marketing, strategy, organizational theory, and a variety of engineering fields. In addition, a number of social science inquiries, including cognitive psychology, anthropology, sociology, and linguistics are relevant for the issues raised [3]. Not surprisingly, these diverse research communities do not always communicate well even when their core phenomenon of interest is largely shared. This diversity of research backgrounds is reflected in the rich variety of terms that have been employed to characterize the requirements arena. Requirements definition [4, 5], requirements analysis [6, 7], requirements determination [8, 9], requirements development [10, 11], requirements engineering [12, 13], and systems analysis [14, 15] have all been used to capture facets of the design requirements task. Outside software systems, the term requirements is often eschewed entirely in favor of *needs* or *customer attributes* [16]. For the purposes of the current study, we use the term *design requirements processes* to refer to the range of activities involved in determining what features and functions an artifact must embody and what constraints it must satisfy in order to address the types of questions outlined above. We will employ this term to emphasize the universal nature of requirements questions for contemporary software-intensive design efforts.

Despite the fact that design requirements form an interdisciplinary area of study [17, 18], the bulk of research on the subject comes from software engineering, computer science, and information systems domains. Within these communities, the criticality of requirements processes has been recognized for decades. In one of the earliest works to raise requirements questions, Ross & Schoman [4] stated that inadequate attention to the needs and envisioned functions of a system leads to "skyrocketing costs, missed schedules, waste and duplication, disgruntled users, and an endless serious of patches and repairs euphemistically call 'system maintenance'" (p. 6). A similar point was made by Bell & Thayer [19], who noted that problems originating in the requirements process often go undetected and later get attributed to bad design or technological limitations. The economic ramifications of requirements were recognized early on by Boehm [20] when he noted that the correction of requirements errors cost a fraction of the impact when errors go undetected until testing and implementation. Later, Boehm & Papaccio [21] mapped empirically the exponential rise in the cost of requirements errors as a systems development effort progressed.

Two decades ago, researchers had already highlighted many of the challenges associated with the requirements undertaking itself. Davis [8] observed that requirements challenges are inherent in any systems design effort because of the complexity of the requirements task, the limits to human information processing, and the intricate interaction between designers and intended users. The emergence of adversarial relationships between designers and other stakeholders has often been cited as a key impediment to effective requirements processes [22]. Even when good relationships have been established, the requirements processes are often inhibited because users do not thoroughly understand what they want to achieve [23]. In addition, the process remains sensitive to other forces that shape organizational life. Bergman et al. [24] noted that requirements processes are unavoidably intertwined with the politics of resource allocation and legitimacy of decision-making within organizational environments.

Ultimately, design requirements processes are challenging due to their Janus-faced nature.[1] Throughout the requirements effort, designers direct their gaze simultaneously in two opposite directions and toward two different social worlds: backwards to the needs of the stakeholders for whom they are designing an artifact, and forwards to the artifact itself and the demands set up by the development environment. Design requirements represent the gate or trading zone in the process at which the amorphous and ambiguous needs of a business or a consumer are married with the concrete design and engineering steps needed to address them [3, 18].

Despite a significant body of research on requirements, unresolved issues continue to haunt designers across the industrial spectrum. In particular, the "requirements mess" remains a challenge among information technology professionals [25]. Since the Standish Group first published its survey of information systems success and (more notably) failure [26], requirements researchers have been quick to note that the three leading sources of project difficulty – i.e., lack of user input, incomplete requirements, and changing specifications – are directly related to the creation and management of a projects' design requirements [11, 17, 27-29]. Likewise, Keil, et al. [30] observed that misunderstanding of requirements and the failure to gain user involvement were among the top project risks. In addition, researchers have noted the persistent gap between research and practice, despite the fact that the area of inquiry is ostensibly motivated by the real-world concerns of designers [3, 31-35]. This gap runs both ways: practitioners are slow to adopt the requirements methods developed by researchers [36], whereas researchers often turn a blind eye to the actual practices and needs of designers [37].

The present study seeks to address this discontinuity through a review of major threads in the past research into design requirements. We strive to assess the state-of-the-art in requirements practice and theory, identify gaps between research and practice, and solicit fruitful avenues for research in the coming years. The research questions that we seek to answer are diverse:

1. What activities and assumptions characterize the contemporary practices of managing design requirements?
2. How are requirements practices consistent with perspectives on design requirements, as reflected in the research literature?
3. What tasks are typical in current requirements processes and what are the newly emerging challenges?
4. What trends are driving requirements practice changes today and over the coming years?

To address these questions we report the findings of a field study about requirements practices among leading design professionals from across the industrial spectrum. We seek to glean key insights about the state of current practice and identify drivers of change in 21st century requirements design efforts.

The remainder of the study is organized as follows. In Section 2, we review the research literature, and introduce central concepts and topics that will inform our study.

---

[1] Janus was the Roman god of gateways, doorways, beginnings, and ends. This is a fitting metaphor for requirements researchers, who stand now at the threshold of a new era in requirements practice.

Section 3 explains the research approach adopted and research questions that we sought to address. Section 4 highlights key findings from the field study. The implications of these findings for the future of design requirements research is offered in Section 5. Section 6 concludes the study with a call to action for researchers and practitioners alike.

2   Requirements Research – A Short Overview

Before exploring the state-of-the-art in requirements practice, it is essential to understand the discourse that has emerged around requirements within the research literature. Accordingly, we will attempt to highlight some of the key concepts that have marked the requirements research tradition. As noted above, requirements processes have been implicated in a wide variety of design shortcomings. As a result, the research around requirements has remained predominantly prescriptive. It is replete with analytical frameworks, standards for requirements quality, elicitation approaches, and modeling methodologies. A wide array of textbooks and reviews have been published, advising practitioners on the most advisable approaches to requirements engineering [10, 12, 38-43]. By comparison, a relatively small percentage of the literature has focused on advancing a theoretical or empirical understanding of how design requirements are discovered, defined, negotiated, and managed by individuals and teams within organizations and why these processes are so difficult. Moreover, the prescriptive modeling and process methodologies have seldom been subjected to rigorous empirical scrutiny due to issues of cost, access, and threats to internal validity [44].

However, it is important to note that requirements processes are far from monolithic. Just as requirements represent one facet of a broader design effort, so too requirements processes can be divided into a number of facets. Within the research literature, multiple frameworks have been developed, positing anywhere from two to seven primary facets for requirements [45]. For the current discussion, we adopt a widely-employed and straightforward categorization of the requirements processes into three facets: 1) discovery, 2) specification, and 3) validation & verification (adapted from [39]).

During *discovery,* designers develop an understanding of the application domain and infer specific design needs through consultation with stakeholders and reviews of other sources of information [12]. This process includes the identification of all relevant stakeholders for the design effort. *Requirements specification* is a term that is treated both as a noun and a verb within the research literature. As a noun, a specification forms the document in which the requirements for a design effort are articulated, and it represents the fundamental agreement between the stakeholders and the design team [41, 46]. The verb form suggests the process of developing and managing the specification document; it is the process by which the design team abstracts and represents the requirements for the design effort [39, 44]. This interpretation of requirements specification as a process will be primarily used in the current discussion. Finally, during *requirements validation and verification* designers ensure that the requirements are of high quality, address the users' needs, are appropriate for the design effort, and have no inconsistencies or errors [47].

While this tripartite characterization appears to imply a linear approach, the three facets are normally employed iteratively, often moving progressively to more detailed levels [45]. The degree of iteration between the facets varies based on the methodology espoused by the design team. However, despite the strong interconnectedness of facets, most requirements research has focused on only one of them at a time. A more detailed exploration of these facets is warranted. Next we will highlight ideas that have emerged in each of these facets, acknowledge assumptions associated with each, and discuss persistent challenges to be explored.

2.1   Discovery

Discovery is the first component of any design effort – a designer or a design team must determine what organizational or customer needs must be addressed by the design artifact [13, 39, 48]. This process is also often referred to as requirements elicitation which conveys a widely held (i.e., traditional) position that knowledge about requirements resides with users or other stakeholders, and must be "teased" out and clearly articulated by the designer.[2] Discovery is also the primary process by which designers gain knowledge of the relevant application domain. As Loucopoulos and Karakostas [39] note, the critical role of understanding of the application domain "cannot easily be overestimated … when you have to solve somebody else's problem the first thing you have to do is to find out more about it" (p. 21; emphasis in original). This statement illustrates the assumption that the designer is in most cases regarded as an outside party in the application domain, who is brought in for a limited period of time to resolve a problem that is of potential concern to others.

While one may speak of several traditional approaches to discovery, there are a wide range of techniques that have been employed in this undertaking [32, 48]. Table 1 summarizes a number of key discovery techniques and their associated advantages and disadvantages. The most rudimentary form of requirements discovery is introspection on the part of designers [48]. During introspection, designers reflect upon or imagine design features that they would find desirable given their understanding of the application domain. Such introspection does not involve direct discussion with other design stakeholders and is therefore often discouraged, if divorced from interactive techniques. Among the most widely noted discovery techniques are one-on-one interviews between a designer and stakeholder, focus group discussions facilitated by members of the design team, and direct observation of business processes or stakeholder activities [32, 49]. Interviews and focus groups emphasize a discussion between representatives of the design team and those closest to the application domain around current experience, areas of discontent with the existing environment, and desired changes that a design artifact might engender. These methods involve both a scrutiny of the current state and generation of possible future states that could be pursued during the design undertaking. Direct observation eliminates explicit discussions, but underscores a designer's detailed understanding of the ways in which activities actually unfold in practice.

---

[2] The term *discovery* was adopted in an effort to broaden the understanding of requirements identification to cover envisioning or innovation on the part of design team members. This conception is meant to overcome the limitations of the passive "collection" or "capture" role reflected in the phrase *requirements elicitation*.

A number of data-intensive discovery techniques, such as protocol analysis [50, 51] and the use of ethnography [48, 52, 53], have been proposed to enhance identification and assimilation of tacit information during requirements processes. Finally, prototyping has been widely employed as a way to expand requirements elicitation activities. It refers to the development of a rough and relatively rudimentary design artifact that includes some essential features desired by relevant stakeholders [54]. Prototyping is particularly effective in establishing a common basis for understanding and communicating design ideas between a designer and stakeholders. For this reason, it may also be analyzed within the requirements validation facet.

While a wide array of discovery techniques are available, it is important to note that they are not mutually exclusive and a combination of techniques can be complementary [17, 32]. It has repeatedly been observed that no single technique is appropriate for all design contexts [13, 29, 55, 56]. There is also clear empirical evidence that the way in which the discovery process is structured impacts both the quality and quantity of the requirements, as a combination of techniques enable designers to adopt multiple perspectives on the application domain [57]. In addition, Hickey & Davis [29] note that the careful selection of appropriate techniques for a given situation is the hallmark of a truly experienced analyst. Regardless of the methods adopted, the process should be well aligned with the documentation of those requirements.

Despite the proliferation of requirements discovery techniques, several questions remain to be answered. It is unclear the degree to which espoused discovery practices have been adopted in real-world design efforts and under what conditions. As with many areas of social science research, requirements discovery is marked by a significant gap between research and practice [32, 33]. There is some evidence that formal discovery techniques have been effectively applied by technology consultants and expert users [29], but their degree of acceptance in a broader industrial context remains an open question. Other areas ripe for inquiry include: What skills do design team members need to effectively execute various discovery techniques? In the area of software engineering, what impact has the rise of commercial off-the-shelf (COTS) solutions had on approaches to requirements discovery within organizations? Do most designers adopt a one-shot approach or a more incremental perspective on requirements discovery? How has the need for speed and agility altered requirements discovery?

2.2 Specification

As stakeholders needs emerge, they must be rendered in some concrete format and representational scheme. This rendering effort is referred to as the specification process. Overall, a requirements specification supports interpretation and understanding among all design stakeholders around what the artifact is supposed to accomplish, while at the same time laying a sufficient technical foundation for the subsequent development effort. Thus, specification is more than just rendering requirements into some standardized format from the information expressed by stakeholders. It marks the point of transition where the stated needs of stakeholders will be extended with the functional and technical implications that flow from them. Nowhere is the Janus-faced nature of design requirements more evident than in the specification. Traditionally, the requirements literature has sought to emphasize the backward focus towards

Table 1. Summary of Selected Requirements Discovery Techniques

Discovery Techniques	Summary	Advantages	Limitations
Designer Introspection	Designers' reflect or imagine features that they would find desirable given their understanding of the application domain	Requires no specialized elicitation skills on the part of design team members Essential in innovative designs which break out from established approaches	Eliminates contact with other design stakeholders Ignores the needs of those most closely linked to an application domain Provides no basis for validation of requirements
Interviewing	One-on-one discussions between a user and designer using open-ended/unstructured, semi-structured, structured, and survey-based variants [32, 48, 49]	Effective for gathering large amounts of information about the application domain [32] Enables designers to focus on a limited number of users as representatives of other stakeholders Requires fewer specialized skills than other discovery techniques	Stakeholders are constrained by the line of questioning employed by the designer [48] Biases in questioning and anchoring effects direct the inquiry to the preferences of designers rather than the needs of stakeholders [58, 59] Gets only at work practices that can be explicitly expressed [8, 60] Appropriateness of the sampling and access to stakeholders are critical
Focus Groups	Designer-facilitated inquiry with a selected group of stakeholders about the current state of practice and the future design space; Adapted from marketing research [61]	By moving away from the individual focus groups engender a more thorough exploration; a statement by one participant may prompt conflicts, extensions and responses by others The presence of multiple stakeholders allows for the questioning and exploration of the assumptions and timely attention to areas of conflict	Designer/analyst facilitation may limit the conversation to the topics determined a priori by the design team Stakeholders are called upon to reflect in abstract on their practices and tacit features of the context remain unexplored Due to a representation from multiple stakeholder, the potential for destructive conflict and political maneuvering is raised Appropriateness of sample is still a concern and the perceived costs to the organization are often higher

Table 2. Summary of Selected Requirements Discovery Techniques (*continued*)

Discovery Techniques	Summary	Advantages	Limitations
Protocol Analysis	A stakeholder is asked to perform an activity and talk aloud about the steps – outlining the rationale for each action [50]; Grew often out of the development of expert systems [51]	Can augment an interview process by surfacing tacit elements of work Engenders a more reflective and thorough description on the part of the stakeholder.	Is built upon an overly-simplistic, computational model of cognitive processes, Apt to overlook nuances of activity in an actual context of use [48].
Prototyping	The development of an early, rudimentary version of system that includes the essential features [54].	Assists when requirements are poorly understood by enabling stakeholders to get experience of what a new artifact may be like Promotes discussion of system features that had not been considered during interviewing or group discussions [12]. Creates a common point of reference [54].	Users become attached to functionality provided in a prototype and may resist changes to the proposed design [54] By emphasizing iteration prototyping may result in "spaghetti code," [62, 63]. Problematic in the development of large systems having significant interdependencies with other systems [54].
Ethnographic Methods	Longitudinal observation within the application domain; Adapted from ethnomethodology in sociology and anthropology [64, 65], and inspired by advances in industrial design [2]	Ethnographic methods can discover knowledge of the application domain to a degree not achieved with traditional methods [48, 52] Mitigates the difficulties associated with tacit knowledge because designers experience the application domain not	Consumes significant time and resources because of long-term focus May be deemed infeasible for design efforts with short timelines or tight cost restrictions

the needs of stakeholders by stating that requirements are concerned with what is to be achieved by a design artifact (i.e., the "what") without regard to the manner in which it will be designed and implemented (i.e., the "how") [38]. Yet this stance "leaves unresolved the question of whether or not it is possible or desirable to separate the 'what' from the 'how' in practice" [66: 18]. With rising systems complexity and interdependence between systems, scholars have started to acknowledge the need for incorporating design considerations and key constraints on the design space during specification [39, 67].

Before discussing in more detail the primary treatments of specifications in the extant research literature, it is worthwhile to introduce a number of concepts that are central to the discussion of requirements specifications:

Abstraction refers to the ability to glean the essence of something from specific instances [45]. In the context of design requirements processes, abstraction enables designers to induce essential elements or processes from specific statements about the application domain and problem space. This helps to ensure that information which enters the specification is essential rather than idiosyncratic, and offers a sound baseline for design.

Decomposition is the process by which systems are partitioned into components. It is a critical capability in any complex design because it allows members of a design team to focus their efforts on manageable tasks. In addition it breaks a large design into its composite subsystems and supports designer's ability to explain and predict outcomes. Decomposition lies at the heart of contemporary advances in modular design and economies of scale and scope in design [68].

Traceability refers to the idea that all "lower" level statements of requirements should be associated with specific higher order objectives and properties and vice versa [69, 70]. In effect, there are two forms of traceability, which correspond to the two directions of the Janus's gaze. *Backward traceability* is the ability to tie a stated requirement and its design and implementation back to its source in business objectives. *Forward traceability* refers to the ability to trace a given requirement or feature to the components of the designed artifact or their interactions that ultimately address it [71]. The traceability concept is the compliment of decomposition. In design, traceability is essential to manage complexity and change and to guarantee that systems validate and "meet" requirements. It also enables designers to evaluate the implications of requirements change regardless of the level of detail at which they are introduced. Finally, traceability facilitates the assessment of completeness and consistency of requirements (see Validation & Verification).

In the development of a requirements specification document, designers generally combine natural language descriptions with formal or semi-formal models of the application, problem, or design space.

Natural Language. During discovery, the primary way in which stakeholders express their needs is through natural language. Accordingly, design requirements at the highest level (i.e., business or user requirements) are rendered through natural language descriptions. Natural language use has several distinct benefits. Foremost among these is that most stakeholders prefer natural language to more formal specifications [72]. Natural language also provides a common basis for communications between the stakeholders and designers (as well between different

stakeholders), and it can provide a great deal of information about the contexts of use [73, 74]. Finally, natural language use is inevitable as we can never achieve fully formalized articulations [73]. Natural language remains the ultimate meta-language where all meanings must be negotiated, and it thereby offers openness to sense-making and discovery [75].

Despite its strengths, most discussions of natural language in requirements research have emphasized the challenges it presents and have proposed ways to overcome its limitations through formal analysis. Researchers have long argued that the informal treatment of specifications leads to ambiguity, incompleteness, and inaccuracy [76]. Ambiguity arises because stakeholders and designers may interpret the same words in different ways. Similarly, distinct stakeholders may use the same term differently, leaving designers to decide which sense is appropriate for the design context. Questions regarding completeness and accuracy emerge because the informal nature of natural language inhibits explicit analysis. Finally, natural language descriptions hide inconsistencies because they provide little basis for direct comparison across statements. In an effort to overcome such shortcomings, researchers have pursued natural language processing capabilities to automate the generation of formal models from natural language inputs [77-79]. However, the bulk of the specifications research has focused on ways to augment natural language representations with formal and semi-formal models of requirements.[3]

Modeling. Perhaps no single subject within requirements research has received more attention than that of modeling [17]. Some even argue that model development lies at the very core of the entire requirements undertaking [80]. In this context, modeling refers to the creation of abstracted representations (i.e., models) of the real world through the use of limited and established symbol systems [81]. The portion of the real world to be modeled is the application domain and its relationships with the proposed design. The resulting models reflect abstractions, assumptions, and known constraints within that design domain [39].

There are several key benefits that have been attributed to formal specifications. By encapsulating large amounts of information, requirements models establish a baseline of understanding. In addition, they may facilitate communication between distinct stakeholder groups [80]. Models also enable formal analysis to identify unstated requirements, predict behavior, determine inconsistencies between requirements, and check for accuracy. Finally, models serve to simplify the application domain by focusing on essential features in line with the principles of abstraction and decomposition. While each of the proposed benefits of modeling is sound in itself, these arguments illustrate one of the tacit assumptions that plagues much of the modeling literature – an emphasis on the perspective of the designer. Within this literature, the focus is squarely placed on the ways in which modeling can be used to support or enhance the work of designers with less regard for the preferences of other stakeholders.

Models are developed at multiple levels of detail. Loucopoulos and Karakostas [39] identify three central levels of modeling in contemporary design efforts: enterprise modeling, functional requirements modeling, and non-functional requirements

---

[3] There has been markedly less discussion about the converse potential of overcoming the limitations of formal modeling techniques through innovative uses of natural language.

modeling. *Enterprise modeling* refers to the development of models to reflect the broader organizational or market context of a design, including the representation of relevant stakeholder groups and the social structure, critical processes, and guiding objectives of the enterprise or the marketplace. Enterprise model development helps

Table 3. Summary of Modeling Meta Models

Meta-Model Category	Description	Exemplars
State Models	Modeling a system as a set of distinct states and the modes of transition between states; Appropriate for representing reactive, event-driven systems.	▪ Finite state machines [90] ▪ Petri nets [91] ▪ Statecharts [92]
Structural Models	Modeling of a system based on the structural features of the application domain; One of the earliest efforts at formal systems modeling,	▪ Structured analysis and design techniques (SADT; [4, 93]
Activity Models	Modeling a system as a collection of activities; Appropriate for modeling "systems where data are affected by a sequence of transformations at a constant rate" (Machado et al. [88],p. 25).	▪ Structured analysis and structured design (SASD; [94, 95] tools such as data flow diagrams (DFD)
Object-Oriented Models	Approaches that incorporate many concepts and fundamental techniques introduced in other methods; Adds to these concepts such as decomposition into objects, inheritance, and encapsulation [96].	▪ Object modeling technique (OMT; [97] ▪ Object-oriented software engineering (OOSE; [98] ▪ Unified Modeling Language (UML) [84]
Agent-Based Models	Modeling of complex systems as a collection of autonomous decision-making agents; Especially useful for the simulation of emergent phenomena [99]	▪ Axelrod Cultural Model (ACM) [100] ▪ Construct-TM [101, 102] ▪ Sugarscape [103]
Goal-Oriented Models	Modeling of the underlying objectives that motivate a design effort; Goal-oriented models incorporate both goal features and linkages between distinct goals [104]	▪ KAOS methodology [105, 106] ▪ Non-Functional Requirements (NFR) framework [89, 107]

achieve a thorough understanding of the application domain and the interdependencies that it embodies. In the context of information systems development, enterprise models focus on interactions between a system and its environment. Examples of these types of models include rich pictures [82, 83], use cases [10, 84], business process modeling [85], and enterprise-level architectural modeling [86].

Functional requirements modeling focuses explicitly on representing requirements about the design artifact itself – abstracting it from the environment and making it amenable to design. Techniques for modeling functional design requirements have proliferated since the earliest efforts in the mid-1970s. Most of these modeling approaches are specific to the context of information systems design. The modeling techniques may be categorized based on the ontological perspectives they apply to the application domain [87]. Machado, et al. [88] refer to these ontological categories as meta-model characterizations. Table 2 provides a summary of meta-model categories and some of the associated modeling approaches. Finally, *non-functional requirements modeling* refers to the development of models to identify the constraints or restrictions on the design domain. In the information systems development discourse, non-functional requirements also incorporate the quality expectations for a system, often referred to collectively as "ilities" (e.g., reliability, adaptability; [89].

The bulk of the modeling literature has focused on techniques for modeling functional design requirements. While most of these modeling methods were introduced as individual techniques for representing an application domain, recent trends have been toward integrating across modeling perspectives [39, 108]. For example, the IDEF family of modeling methods enables a design team to apply multiple development ontologies to the requirements modeling [109]. The introduction of the Unified Modeling Language (UML) during the last decade has greatly extended the trend towards employing multiple perspectives. UML is an outgrowth of the object-oriented specification tradition, but incorporates a broad suite of modeling techniques, including class diagrams (an extension of E-R diagrams), state-chart diagrams, activity diagrams, and use case diagrams [84, 110].

In addition to the move toward integration across ontological perspectives, modeling research has been marked by two countervailing trends. The first emphasizes increased standardization in the specification of notation systems and processes. Hundreds of modeling techniques have emerged over the past 30 years, but this diversity in fact poses an impediment to adoption. Some researchers have called for a moratorium on new model development until existing models have been tried and exhausted by practitioners [36]. The development and adoption of UML provides an example of the benefits of standardization. The UML suite was developed when three "thought leaders" in object-oriented modeling recognized the convergence in their modeling methods and decided to work together to create an industry standard [84]. Since its introduction, UML has rapidly emerged as a de facto industry standard, creating a measure of modeling consistency across industries, organizations, and design environments backed by standardization organizations [111].

The second, and perhaps contradictory, trend is the move toward increased customization of modeling based on the types of systems or contexts involved. *Situational method engineering* (SEM) is a movement to customize development and modeling approaches to the needs of a given design task through the selection, recombination, reconfiguration, and adaptation of *method fragments*, many of which are

modularized abstractions of existing methods [112, 113]. Similarly, recent work on domain-specific modeling (DSM) emphasizes efficiencies to be gained by tailoring languages and modeling techniques to the specific vocabularies and abstractions of a given domain [114, 115]. While the trend toward customization focuses mainly on the composition and reconfiguration of methods, it has significant implications also for the evolution of requirements modeling tools and techniques [116]. The observation of these two trends raises the question – Can increased standardization be reconciled with desires for customization in modeling techniques and how do such goals align with specific needs in the future?

The conflict between these trends is but one of the pressing questions in research around requirements specification. Other important issues include the following: With significant adoption of UML is there still a need for novel approaches to the modeling of design requirements? Which components of the UML suite or other techniques have been adopted by design practitioners and why? Has the adoption of formal modeling techniques engendered a substantive improvement in requirements and design quality? How can different models be practically integrated? Turning again the issue of natural language, little attention has yet been paid to ways in which language and communication skills of design professionals could or should be enhanced to support high-quality requirements specification. These are among the issues that must be addressed by research on requirements specification in the coming years.

2.3   Validation and Verification

Validation and verification addresses the question of whether or not the requirements processes have been conducted effectively and the degree to which the specifications will support a productive design effort. Some researchers use only the term 'validation' or 'verification' when discussing this facet, but an important nuance between these two sides prevail. *Validation* is the effort to ensure that requirements accurately reflect the intentions of the stakeholders [117]. *Verification* focuses on the degree to which requirements conform to accepted standards of requirements quality [10, 47]. Boehm [47] captures the distinction succinctly when he states that validation addresses the question "Am I building the right product?"; while verification asks "Am I building the product right?" (p. 75).

Validation. Locating requirements validation as the end point of the design requirements may give a false impression that it is of limited importance and does not shape behaviors significantly. In truth, the validation begins almost simultaneously with discovery and continues through the specification. When a designer uses paraphrasing to check his or her understanding of a stakeholder's request or statement, validation is taking place. Indeed, one of the primary approaches to requirements discovery – namely prototyping – is often referred to as a key validation technique [39] .

One of the central issues in requirements validation is the potential for disagreement between individuals or stakeholders groups. Given diversity in their backgrounds, roles, and values, it should not be surprising that conflicts frequently emerge [17, 118, 119]. Effective management and resolution of such conflicts is essential if the design effort is to advance. A range of techniques have been proposed to help designers with conflict resolution:

Requirements prioritization refers to the process of determining the relative value of individual or sets of design requirements [120]. By assigning values to requirements, designers establish a mechanism for mediating between requirements conflicts that arise. Berander and Andrews [120] identify a number of prioritization techniques that have been applied, including numerical assignment (i.e., priority grouping), analytical hierarchy process [121], cumulative voting, and stack ranking.

Requirements negotiation involves the identification and resolution of conflict through exploration of the range of possibilities available [119, 122, 123].[4] These negotiations often draw upon fields of research on multiple criteria decision making and game theory [123, 125], and apply group support systems or collaborative environments for effective negotiation around requirements conflicts [118, 126-128].

Viewpoint Resolution builds upon the *viewpoints* thread within requirements research. *Viewpoints* refer to the emphasis on obtaining design requirements from individuals and groups having different perspectives on the design [129]. Leite and Freeman [130] introduce *viewpoints resolution* as a "a process which identifies discrepancies between two different viewpoints, classifies and evaluates those discrepancies, and integrates the alternative solutions into a single representation" (p. 1255). Thereby, viewpoint resolution feeds back into the modeling process by developing a model of requirements conflict.

Verification. The counterpart to validation is verification. With verification, we turn our attention back to the functional and technical implications of the requirements. Much of the discussion around requirements verification focuses on ensuring adherence to standards of requirements quality, including consistency, feasibility, traceability, and the absence of ambiguity [10]. *Consistency* refers to the idea that requirements should not conflict with the overall objectives of the design effort, or with each other. As the number of individual requirements statements proliferate in large scale projects, concerns over the potential for inconsistencies between statements rise and verification measures are implemented in efforts to safeguard against errors. *Feasibility* is the degree to which a given requirement can be satisfactorily addressed within the design environment of an organization. This includes not only a consideration of whether or not an artifact can be developed in line with the requirement, but also how it can be subsequently maintained. As discussed above, *traceability* is the degree to which individual requirements can be tied to both higher order objectives and detailed elements and their interactions within an artifact.

A number of techniques have been proposed to support this verification function. In one of the earliest discussions of the importance of verification (and validation), Boehm [47] presented a range of both manual and automated approaches to verification, including such simple techniques as manual cross-referencing, the use of checklists, and scenario development to the more complex activities of mathematical proofs, detailed automated models, and prototype development. Other key verification techniques include formal inspections [131], structured walkthroughs [132], and automated consistency checking [133].

---

[4] It is worthwhile to note that many researchers would position requirements negotiations as part of the specification phase of a requirements process or as a distinct phase altogether (e.g., [124]).

In total, the requirements research literature has provided a wide range of insights into the individual facets of requirements work. From the introduction of formal modeling techniques to the development of novel approaches to discovery, requirements scholars have consistently sought to improve the lives and resources of designers. Yet, the degree to which these efforts have resonated with practicing designers remains to be seen. While some researchers emphasize the increasing adoption of techniques from research (e.g., [34]), others bemoan the growing gulf between researchers and the practicing design community (e.g., [33]). In addition, our review illustrates a number of key research assumptions including a focus on individual systems, attention to a single facet of the process at a time, emphasis on notations to represent requirements, and the primacy of a designer-centric perspective. In the next section, we analyze these assumptions in the context of a field study.

3   Research Approach

In an effort to explore the current state of requirements practices across a variety of organizational and industrial contexts, we conducted a series of semi-structured interviews with IT and design practitioners from the United States and Europe. The data collection efforts were structured around an interview protocol that was jointly developed by the researchers. The interview protocol was designed to elicit responses to a number of distinct aspects of the professionals' design experiences, including a discussion of current design requirements processes; perceived impediments to the identification, specification, and management of design requirements; drivers of change in requirements practices over the preceding five-year period; key trends within the market or technological environments relevant to the given organization; and envisioned changes to the practice of requirements in the near future. The core protocol remained constant throughout the data collection process, however, in line with the grounded theory concept of constant comparison, some questions were added to the protocol based on insights from the initial interviews [134]. In addition, interview participants were encouraged to express their thoughts on any topics which they felt were relevant to requirements processes and contemporary design environments.

To foster external validity and to address threats to the internal validity of the study, the research team sought participation from individuals and firms engaged in a wide variety of design environments. A number of business and design contexts were initially targeted in the site selection process to ensure representation from areas where the researchers expected to observe significant market and technological change occurring. To ensure representation from leading edge and mainstream organizations the research team sought participation from senior technology leaders within a range of Fortune 500 organizations. A total of 30 interviews were conducted with 39 individuals participating. The interviews included firms from a wide range of industries, including software and information technology, automotive manufacturing, industrial design, aerospace, telecommunications, professional services, and healthcare sectors. Importantly, despite this range of industry domains, all of the professionals interviewed are engaged in the design of software-intensive systems. In order to protect the confidentiality of the respondents, none of the quotes or statements from the interviews are attributed to specific individuals or firms.

These initial focal contexts of studied systems and their requirements included the following:

- Large, complex organizational information systems – The design of very large information systems, often supporting inter-organizational exchange of information; including transportation systems, distribution networks, and defense contracting.
- Embedded systems – The design of systems and components intended for integration within broader design artifacts; includes automotive and aerospace design environments.
- eBusiness Applications – The design of artifacts and information systems for use within a Web-based delivery channel; includes portals, e-commerce establishments, and other Internet-oriented product and service providers.
- Middleware Systems – The design of integrated software platforms that support the exchange of data between distinct applications.

It should be noted that our sampling approach reflects a purposeful bias toward large, complex systems in an effort to focus on practices associated with the most challenging development contexts. The systems development efforts reflected in the study involved from tens to hundreds of man years. System costs ranged from several million to hundreds of millions of dollars.

All interviews were transcribed to support formal analysis of the data. Interview transcripts were coded using Atlas.ti, a qualitative analysis application. The interview protocol served as the preliminary coding structure for the data. However, in line with a grounded theory approach, additional codes were created as specific themes or recurring issues began to surface in the coding process [134]. The code structure was iteratively revised until the researchers determined that all relevant themes or issues were reflected [135]. Several of the interview transcripts were coded repeatedly as the final coding structure emerged. The aim of this analysis was to identify distinct patterns in current requirements processes as well as to observe emerging key themes and issues in the day-to-day practice of requirements work. In the Findings section we will explore these observations in detail.

4   Findings

4.1   Current Practice

The field study revealed a number of key observations regarding the current practice of requirements management. Several of these findings reflect general approaches to requirements determination issues while others relate to specific facets in the requirements process (e.g., discovery, specification, validation). We will briefly discuss the findings regarding current practices before delving into the emerging themes and issues that surfaced in the study. Table 3 provides a summary of our findings regarding current requirements practices.

Table 4. Current Requirements Practice

Overarching Practices	Development based on the use of CASE tools
	Risk mitigation common
	Broad external and internals stakeholder involvement in requirements processes
	Focus on data and process modeling
	Non-distinction of requirements tasks & functional/nonfunctional requirements
	Contingent application of requirements methodologies
	Limited focus on stakeholder conflict
Discovery Practices	Primarily focus groups and interviews
	Simultaneous elicitation and validation
	Responsibility for requirements discovery and justification rests largely with business
Specification Practices	Specifications based on CASE tools
	Widespread application of use cases
	Natural language, data, and process modeling representations based on UML standard
Validation & Verification Practices	Little use of formal verification practices
	Widespread use of prototypes and group walkthrough sessions
	Common expectation of a formal stakeholder signoff

4.2 Common Requirements Practice

Overall, requirements practices have evolved significantly over the past decade, often in line with prescriptions offered in the academic and consulting literature. For example, much of the requirements literature of the 1980s and 1990s prescribes a disciplined process, often emphasizing the control of development risks [8, 136]. In addition, there has been a significant emphasis on fostering the involvement of a variety of stakeholders and the application of formal modeling techniques such as UML, and the use of supporting CASE tools [137, 138]. Our data generally suggest practices which are consistent with most of these prescriptions. Development environments based on tools such as IBM's Rational Suite are commonplace. Several participants note that project risk mitigation is a central area of development focus, and some informants indicated that portfolio risks are consistently measured and actively managed. The individuals and teams interviewed indicated that requirements activities commonly include focus group discussions, cross-disciplinary project teams, and requirements sign-offs from an array of stakeholder groups. In addition to the widespread use of data models, several organizations note sophisticated process modeling activity, including the widespread application of use cases, even in situations where other elements of UML were not fully adopted. Other principles that are addressed in

the literature, such as traceability and structured process improvement (e.g., CMMI), while not prevalent in current design practices, received significant consideration in the future efforts of many interviewees, and were noted as directions in which firms are actively moving. Similarly, trends that are often addressed in both the academic and practitioner literature, such as web services/service-oriented architectures (SOA) and outsourcing of development, were reported as influencing current requirements practice.

Yet, in many cases designers did not employ concepts and distinctions that are common place in the research literature. For example, few of interviewees made a distinction between functional and non-functional requirements. While they do seek to capture constraints about desired performance (e.g., traditional "-ilities") for designs, they do not document and manage these requirements in a differential manner. Another break with the research is in the characterization of the requirements process. Several interviewees expressed their belief that requirements processes are indistinguishable from the design. While researchers have long asserted that requirements should presage efforts to determine how desired functionality could be achieved (i.e., the formal design), many participants felt that requirements questions are properly interspersed in the design process. Those interviewed emphasized the intensely iterative nature of requirements and design.[5] Even when the recognition of requirements as a formal, early phase of a design task was recognized, the designers did not mark distinctions in requirements activities, such as elicitation, specification, negotiation, validation, or verification. Thus, many of the classification schemes that demarcate discourses within requirements research remain unrecognized in contemporary practice.

A second key finding with respect to the practice of requirements is that the application of standardized and more formal methodologies might best be described as haphazard. Within the organizations represented, design professionals are seldom expected to adhere to explicit methodologies in the discovery, specification, or verification of design requirements.[6] Most projects advance through a patchwork of techniques at the discretion of project managers. Despite the generally idiosyncratic nature of the requirements processes, there were a number of contingencies for the application of methods. Project budgets, personnel needs, and the number of systems impacted are key considerations in determining whether or not a more rigid process is necessary, with the larger and integration-intensive efforts carrying the increased expectation of the use of formal methods. Interestingly, the application of formal methods throughout the design process and across different projects was repeatedly noted as a direction for future development and improvement. The following statement is characteristic:

---

[5] Note that while this iteration was often discussed, it was rarely in the context of agile development. Interviews were virtually devoid of discussions of agile methodologies, with only a couple of exceptions - a finding which may be a function of the highly complex development practices within our sample.

[6] It should be noted that validation efforts are an exception to this finding as formal mechanisms for phased advancement of design projects (in the form of sign-offs by business sponsors or other key stakeholders) were nearly universal.

"You have to become a lot more method [sic], a lot more rigor, a lot more science and gathering your requirements, qualifying them and then quantifying them in terms of financial [considerations]. Because at the end of the day, that's what we're in business for, to make money and pay dividends to our shareholders."

Another consistent finding from the study pertains to the management of conflict within designs. Despite the fact that researchers pay significant attention to negotiation and the management of disputes, conflict between stakeholders was viewed as largely unproblematic. Simple prioritization of requirements by a key sponsor or stakeholder acts as a primary mechanism for the management of conflicts. Frequently, the determination of such priorities is tied directly to the funding – i.e., whoever is perceived to be the primary source of funding sets the priorities. In this way, the valuation of requirements is transferred from the design team to the business stakeholders. However, the voice of IT stakeholders remains significant when the prioritization is subject to prior architectural decisions and constraints (see "Key Themes and Issues" section).

The participants experienced the most significant impediments to effective requirements processes in the interpersonal aspects of a design effort. In large part, these challenges reflect those often noted as key challenges throughout the requirements literature: stakeholders not knowing what they want, the inability of stakeholders to articulate what they want even when they do know it, and limitations in the communication skills of the design team. Interestingly, respondents noted very few impediments arising from available technical resources and formal methods. For example, no single participant felt that the absence of appropriate modeling approaches and tools set up a significant challenge to their requirements processes.

The study discovered a number of key findings concerning specific facets of the requirements processes. While the respondents themselves frequently failed to distinguish such requirements activities as discovery, specification, modeling, verification, and validation, the interview protocol helped glean insights regarding their approaches to the dimensions recognized in the protocol. By applying this established lens, we are able to discern linkages and discontinuities between current practice and past requirements research.

Discovery. With regard to discovery techniques, one of the most consistent observations regarding the process by which design teams explore and understand the needs of stakeholders is the relatively narrow range of techniques employed. Most organizations relied only on focus groups and other group-based discussions as a primary mechanism for requirements discovery. One-on-one interviews with stakeholders are also common. Although more intensive measures such as protocol analysis, direct observation of work practice, and ethnographic participation in the application domain were noted by a small number of respondents, traditional discursive techniques continue to dominate.[7] Also, we noted that discovery and validation often occurred simultaneously – frequently in the form of prototyping but also in other forms such as "blueprinting" sessions.

---

[7] It is worth noting that firms adopting less traditional discovery approaches are specifically recognized within design communities for their unorthodox perspective on requirements gathering.

A second key finding is the degree to which requirements articulation has been established as the responsibility of the relevant line-of-business or other stakeholder group. In several firms, sponsoring stakeholders are expected to engage the design team with a thorough statement of needs, extending beyond the business requirements to high-level functional requirements. In one case, a firm had started a training program in an effort to teach business unit managers to write system requirements more effectively. The discovery activity was also often outsourced to consultants or market research organizations (see "Key Themes and Issues" below). As a result, requirements discovery on the part of the design personnel has often become a matter of clarifying and assessing gaps rather than a comprehensive frontal elicitation effort.

Specification & Modeling. A central observation with respect to the specification of requirements is that the design professionals did not speak of specific modeling techniques employed. Rather they discussed modeling tools that their design teams use. Requirements management platforms such as IBM's Rational suite and Telelogic DOORS were more salient than the specific types of models developed. Use cases represent one important exception, however. Virtually all interviewees noted that their design teams engage in use case development as a central aspect of their specification activity. For example, one participant observed:

> "So a few of our more senior developers had really gotten on board with UML, use case modeling, and then are becoming fairly good with some of the software tools out there. Some of us use IBM's Rational suite. Some of them are working with a product we're evaluating called Rhapsody from I-Logix. But the intent there is if we can graphically present to the customer and do modeling to decompose a lot of those requirements, that it really helps in that review to catch anything that's missing."

Modeling was often described as "informal," and involved extensive use of natural language narratives, which is consistent with the widespread adoption of use cases. Beyond use cases, several participants reported the use of process or workflow models, as well as data models / E-R diagrams. While UML was implied by the application of Rational software, for example, only a handful of interviewees specifically indicated that some portion of their requirements process is based on UML.

Verification & Validation. None of the interviewees noted the adoption of formal approaches to verifications or requirements checking. Specifically, when asked about verifying correctness and assessing the consistency of requirements, the majority noted that this was accomplished through informal review and discussion among the design team. The following quote is characteristic of the responses to this inquiry:

> "Usually I have at least two developers that are familiar with all the systems. If we get new requirements in we'll all do a blind read and then we'll kind of mark it up, say where do you see some holes … and in some cases we'll say, 'Look at page two, item 3.1 and then look at page fifteen item 9.2, it sounds like you're saying A to Z here and you're saying Z to A there.' And sometimes they'll say you're right, maybe it was written by more than one person or they changed their mind and forgot to change in all the places. So we definitely try to go through the entire thing with more

than one set of eyes on our side looking for inconsistencies or omissions or things that look like they're definitely, at a minimum, confusing."

When moving from verification to validation, a greater degree of formality is observed. In most organizations validation efforts centered on explicit sign-offs by project sponsors and other stakeholders. The stakeholders are expected to review the requirements documentation and acknowledge their acceptance for the design effort to move forward. One challenge noted in this regard is that stakeholders frequently fail to review thoroughly the documentation that is presented to them (due to lack of time or perceived time-to-market demands), and therefore design efforts are allowed to move forward despite the potential presence of significant requirements errors. This phenomenon is exacerbated under conditions of multiple sign-offs because of the diffusion and ambiguity of responsibility.

In addition, the interviews noted frequent use of prototyping, user-interface mock-ups, and system walkthroughs as validation mechanisms. While none of the organizations represented was extensively involved in agile development, several emphasized iteration and prototype development:

"To be able to, very rapidly, hear what the requirements are and draft up a prototype, something they can see. Whether it would be, you know, there's varying levels of complexity and money that are associated with something like that right. I could do paper based prototyping or I can do systems based prototyping and having that kind of capability in place to help validate the requirement - is this what you asked for; is this what you would want to see."

Interestingly, a few of the firms indicated novel validation practices, such as validation of use case "personas," validation of time estimates, and stakeholder voting.

4.3   Key Themes and Issues

Beyond the state of current practices, we identified a number of recurring themes and issues that were inductively derived from the interview data. These are themes that tap into emerging trends or patterns in the requirements domain. Table 4 summarizes these themes.

It is important to note that these nine identified themes are not mutually exclusive. Indeed, there is significant conceptual affinity among several of the themes. However, the distinctions and level of detail that is presented emerged from the data through multiple rounds of coding among the authors, with a goal of deriving a parsimonious yet thorough representation of distinct themes and constructs in the data. In several cases, study participants proposed causal relationships between themes, but such causal assertions varied significantly and often suggested a recursive pattern (e.g., focus on integration leads to implementation of packaged software and the implementation of packaged software necessitates a focus on integration). Therefore, we will refrain at this stage from making any causal assertions with respect to the themes, but will discuss some of them critically in the discussion section. We will discuss each of these themes in detail after characterizing the general emerging pattern of requirements practices.

Table 5. Summary of Key Themes & Issues Associated with Design Requirements

Requirements Theme	Brief description
Business process focus	Requirements process focusing on the business process, and requirements for technological artifact driven by business process.
Systems transparency	Requirements driven by demand for a seamless user experience across applications.
Integration focus	Requirements efforts focus on integrating existing applications rather than development of new ones
Distributed requirements	In addition to diverse stakeholders, requirements process distributed across organizations, geographically, and globally.
Layers of requirements	Requirements iteratively developing across multiple levels of abstraction, design focus, or timing.
Packaged software	Purchase of commercial off-the-shelf (COTS) software rather then development – trend toward vendor-led requirements.
Centrality of architecture	Architectural requirements take a central role, and drive product and application requirements.
Interdependent Complexity	While some forms of complexity have been reduced, overall complexity has risen significantly.
Fluidity of design	Requirements process accommodates the continued evolution of the artifact after implementation.

We can describe the emerging practice of requirements as follows: *business process design* take precedence in the design of individual artifacts, and thereby represents a central source of complex socio-technical design requirements. The business process emphasis is driven by an increased demand for *transparency* across distinct systems and the corresponding focus on *integration* over more traditional isolated development efforts. As a result, the requirements process is *distributed* across functional, organizational, and geographic boundaries. The distributed nature of requirements underscores the existence of multiple *layers of requirements*, based on differences in abstraction, user-orientation, and timing. Such layering of requirements is illustrated in the marked emphasis on the use of *commercial-off-the-shelf (COTS)/packaged software* in most of the organizations represented. The heterogeneity of design artifacts within existing business environments in turn necessitates a focus on the adherence to established *information architectures* for all subsequent product and system design efforts. This emphasizes conformance to information *architectures* in guiding requirements, and channeling the implementation of COTS software, rather than developing from scratch. These increase the level of *interdependent complexity*, as well as *layering of requirements* across multiple dimensions. Finally, because of complexity and continuous change, designs are *fluid* as they continue to evolve after implementation, which stresses the importance for requirements to evolve. A more thorough exploration of each of these themes provides an original look into multiple factors that currently drive change in requirements practices.

Business Process Focus. One of the distinct insights to emerge from the study is a consistent shift from a focus on a particular application and its associated work practices to a focus on chains of work practices – or business processes – within which a given set of applications is situated. The comprehensive integration of information system components has become more prevalent, driven by the design focus on end-to-end business processes that utilize these technological resources. Accordingly, requirements for specific artifacts increasingly flow from the holistic understanding of the business process itself. This shift was prevalent in many interviews, but it is not as readily apparent in the research literature.[8] One informant describes this shift as follows:

> "The requirements are often based on the business process… Typically what you would do together with the requirements is you would define your business processes. You define the process flow and the main process steps at that point. You make the main activities. And then you can map the requirements to that."

More pointedly, one respondent mapped out the crucial role of business process management to requirements engineering efforts:

> "The rise of Business Process Management may largely affect the RE process in the near future. The [organization] is already running a pilot project which: Generates requirements (AS-IS / TO-BE modeling) through the Business Process Management practice; translates part of these requirements to specifications for the configuration of Workflow Management and Business Rule Management tools; refers to specific services of the SOA [service oriented architecture]. This way of working will not be applicable to all IS projects, but it seems suitable for particular types of applications."

In a similar vein, a trend that a number of informants identified suggested that the boundaries between business processes and IT are becoming increasingly blurred:

> "There's no such thing as an IT project, there's a business project where IT is part of it. And that's been helpful, but at some point, businesses are going to want IT leaders to in effect be innovative in relation to what the next kinds of solutions are. Some people have said that CIOs should become chief process officers."

The logic behind business investments in IT now emphasizes having organizational priorities and processes to drive IT development rather than letting IT capabilities determine the priorities of activities. Since the bursting of the dot com bubble, most organizations have heard this message loud and clear [141, 142].

Systems Transparency. The orientation toward the design of business processes implies a movement away from system boundaries based on arbitrary functional or divisional criteria. Business users and consumers alike demand transparency across

---

[8] An area of notable exception is the study of requirements engineering in workflow automation systems (e.g., [139, 140]).

software applications. Technologies are expected to converge so as to provide a seamless user experience and generate perceptions of a single service platform. As one participant noted, new solutions must be available anywhere, anytime, anyhow and are often expected to be device-independent. The concept of *systems transparency* highlights increased concerns of the users that emphasize the design of a seamless and uniform user experience. While a great deal of traditional requirements literature focuses on the notion of usability associated with a specific application, systems transparency calls attention to unified usability of a portfolio of applications. The requirements process is no longer about a user's interaction with a single application; rather, it is oriented toward the user's overall experience which is situated within an ecology of applications and design artifacts. One informant succinctly captured this idea of systems transparency, by emphasizing a trend toward personalization of applications:

> "The desire from the customer side is much more for applications to cross whatever artificial boundaries exist in terms of data sources and in terms of small systems coming together. I see a big change in what IT does for requirements focusing more on achieving user-centricity and giving people more of a personalized view of how to do their job and get the data going… a user shouldn't have to worry about what device they're using or what system it's in, just getting the information they need when they need that. That's really a major change in how systems are designed … inevitably the end user customers want a seamless integration, they want a common look and feel…"

In order to accomplish such "user-centricity," the requirements process must focus upon work roles and overall activity in order to provide systems that fit within the distinct daily practices of individuals. According to one interview, this focus "really changes IT people from being raw functional application creators, to being more of, you know, performance architects." Naturally, the seamless user experience must be enabled by linking applications which is addressed by the next theme.

Integration Focus. *Integration focus* denotes efforts associated with making user experiences possible through integrating applications and system components. While the bulk of the literature on requirements addresses the creation of new applications for specific purposes, many study participants downplayed the importance of *designing* individual artifacts, while emphasizing instead the criticality of integration across applications and capabilities. The focus on integration was one of the most pronounced themes across all interviews. The following statement is emblematic:

> "I'd say that the big difference that we've gone through over the five year period was a transition from one standalone system, which might have lived on individual desktops, or lived on a single network for delivery to departmental users, to now more integrated applications which are tied together via one way or another. Whether it's at a database level or whether it's a web services or whatever, we have applications now that need to share data between systems, between business units, and between our affiliated partners."

While this integration is driven by user considerations, there are host of other organizational drivers that shift requirements practices towards integration:

> "With the tighter integration of supply chains and the customer base, you can't have processes and systems that are not integrated… So that brings together the different applications, the platforms they're built on, the middleware and the data structures that you have to have in place to integrate this stuff. If you don't have it in place you design some islands of functionality that don't work together."

A primary implication of the trend toward integration is that the role of internal IT groups is changing rapidly. Many informants characterized their groups more as integrators than developers: the main proportion of their work is oriented toward assessing and maintaining interdependencies between systems rather than being involved with traditional functional requirements and subsequent design. As one participant put it, "for those of us in IT [the need for integration] changed us from application developers into systems integrators."

Distributed Requirements. A pattern that became apparent during the study is the increased distribution of requirements processes across functional, organizational, and geographic boundaries. Frequently, no single organization or functional unit is responsible for the development of the bulk of design requirements. Vendors, consultants, enterprise architects, development teams, business stakeholders, and individual users all play significant roles in articulating and implementing requirements. Furthermore, the widespread adoption of outsourcing has expanded the geographic spread of requirements discovery and development efforts.

Globalization has become a critical force behind much of this distribution, but the implications of globalization go beyond obvious growth in geographical and cultural distance. Distributed requirements bring with them business contexts and features that are tied to distinct locations and business environments. One informant illustrated this vividly:

> "[The organization] represents a large corporation and so you know, we have various initial conditions. So for example, a climate control module might have been supplied by supplier X sitting in Munich and they might have actually inherited along with the hard box, they might have inherited a whole lot of models. Those models would have had certain class definitions. Those class definitions would have been based on somebody else that the supplier was actually supplying some super system to. So we now have to incorporate those classes if we really wanted useful…if it has to be useful to my direct mainstream customer. So we can go create our own universal model here but our customer will have to go through a translation phase."

Not only are requirements distributed geographically, but they are spread across organizations as well. The prevalence of COTS applications (see discussion below) and the growth in industry wide standards results in the significant distribution of requirement's discovery and validation efforts among multiple independent organizations. While this is necessarily the case to an extent for all COTS, it is also amplified

by the complexity of packaged systems and the increasingly limited knowledge many organizations have in formulating solutions:

> "Now we are rolling out my [software system]. That product is way beyond the capabilities of my current team to implement because they haven't spent the months and months of learning how to use this new technology. And we don't have the months and months to wait to train people before we start doing requirements definition and process design."

With this emergence of the distributed nature of design, collaborative tools have become increasingly important for managing requirements that are drawn from increasingly diversified sources. One informant captured the new division of labor that results from distributed requirements as follows:

> "The significant advantage of that is people could be collaborating, not only synchronously but asynchronously … everybody gets to see what the other person is contributing … So for example you might want to have a group of people defining the technical aspect of it. Then you have another group which is defining the business aspect of it. And you have a third group working the external collaborators. And they all have parts and pieces to do it."

Distributed requirements also enhance parallelism in requirements processes. Given the traditional focus on singular individuals or teams managing requirements processes, there is notably little discussion in the literature about the implications of parallel requirements efforts. One area of exception in this regard is a recent focus on the impact of geographically distributed teams engaged in various facets of the requirements process [143-145].

Layers of Requirements. Contemporary design efforts generally entail multiple layers of requirements. These layers may be associated with differing levels of abstraction, design focus, user-orientation, or timing. This layering phenomenon includes the traditional transition through business, functional, and technical requirements [10]. It also includes organizing requirements based on the level of analysis. For example, the process for articulating and managing the requirements of enterprise architecture differ from those considered in the development of an individual application:

> "For example, in the situation that I'm currently in with one of my existing clients, we are not only building new applications for them, we're also building application architectures. So there's two sets of requirements; there's business requirements for the different applications that we're building and then in turn what we have to do is for those applications, what are the set of infrastructure requirements that our technology infrastructure team needs to build to be able to provide the framework that these applications will be built on. Whether that be a reporting architecture or a real-time architecture, batch architecture, online architectures, et cetera."

The volatility, or timing of requirements, is another key basis for layering. Requirements that are expected to persist over an extended period of time demand

distinct approaches from requirements that change rapidly, and in less-predictable ways. This phenomenon is relevant in the design of embedded systems and product lines because the requirements volatility (variability) for the embedded artifact in the product differs significantly from that of the underlying system, as the following statement illustrates:

> "In terms of being able to span the feature space if you will, cover it, there's a timeless element to it - people always want some aspect, they want a place to sit down and they want to be able to steer and drive. There's a piece that changes with very slow clock speed, that's the package and physical aspects of it. And then there's the fast cycle, fast clock speed, aspects. So there's really…there's a DC component, there's one that really changes very slowly and there's one that changes very fast."

The emergence of new bases for the layering of requirements has clear affinity with the distribution of requirements. Layers of requirements may be discovered, specified, and managed across distinct stakeholder groups or organizations. Increased challenges are created by shifts to build mechanisms that ensure consistency across different layers.

Packaged Software Orientation. For systems design efforts, the interviews depicted a clear preference for using commercial-off-the-shelf (COTS), or packaged, software over the development of separate, new applications. The following quote is a good representative of the sentiments espoused:

> "We made a decision that we were going to pick SAP and purchase it in advance of all these projects going live because, in a fact, we wanted to send a signal to the organization that we've picked SAP and it's the only solution you're going to use going forward."

This represents a major point of departure from much of the requirements research tradition, which, often implicitly, conceptualizes requirements practices as taking place in the context of a greenfield development where the end product is a new software system. Requirements for packaged software implementation projects are significantly different from those of traditional development. For example, software vendors and consultants have a great deal of involvement and often take the lead in requirements processes. The prevalence of packaged software creates a new dynamic for requirements processes. In contrast, to the traditional claim that requirements processes should focus on the "what" of a design effort without respect to "how" it will be achieved [38], the use of COTS implies that much of the "how" is already established at the outset of a requirements effort. In these cases, the requirements process begins more with a "gap" analysis between processes supported by the package and the desired work practices:

> "The easiest one is just to take the software as it comes out of the box, some type of a pre-configured solution that may be industry specific, may not be industry specific. And you run workshops where you sketch out the future processes, walk through the software itself in its current state, and identify any gaps with that process and the software and the requirements you have already defined."

While the prevalence of COTS applications was clear, many of the study partici-
pants did reflect on the drawbacks to packaged software – especially in terms of large
enterprise vendors, and their firms' dependence on such vendors as a major
requirements constraint. However, with the exception of truly unique and critical
applications, the benefits of COTS appear to outweigh these drawbacks in the minds
of our participants (e.g., lower cost, better predictable quality).

Centrality of Architecture. A consistent finding in the study was the growing
recognition of the importance of information architectures in establishing the context
for requirements processes. In many of the organizations represented, adherence to
established information architectures has become a critical concern and constraint for
all design efforts. In large part, the specification of formal and encompassing
enterprise architectures is driven by the need to address integration complexity and
need to maintain consistency in applications and organization wide process designs.
Therefore, architectures have become essential for requirements activity and set the
baseline constraints for all subsequent designs.

Because the study involved both functional IT units and product development or-
ganizations, two types of architectures were salient to design practices: enterprise
architecture and product architectures. Many participants indicated that enterprise
architectures, especially those associated with an organization's internal IT infrastruc-
ture, are becoming critical as organizations look to integrate extensive portfolios of
applications that have resulted from previous stove-piped legacy development. An-
other driver is organizational mergers and acquisitions. To move forward with devel-
opment projects, an enterprise-level justification and adherence to the established
architecture is essential. As a result, architectures precede and drive the requirements
of specific artifacts, rather than the requirements driving the development of models:

> "In fact we have a very strict architecture team and an architecture review
> board all focused in my area on, as projects are begun, to insure that those
> projects move forward with the long term architecture of [respondent's
> firm] in mind."
>
> ………
>
> "The architecture determines the scope of application functionality and re-
> quirements which you can do. But if you look at the sort of future evolu-
> tion it may be that you make currently the right architecture choices but
> maybe two years down the road another requirement emerges and you are
> stuck."

In some cases, the enterprise architecture represented significant constraints on the
requirements processes, while in other cases it just changed the structure of these
processes. As a large banking organization indicated:

> "The RE process is being tailored to the special needs of the aforemen-
> tioned architecture in various ways. For example, business analysts are
> aware of systemic calls to the core banking system and refer to them in de-
> tail when they design business functions. On the other hand, the bank is
> still on the process of adopting a service-oriented, multi-channel approach.
> The current, rather immature, situation (as far as multi-channeling is

concerned) generates a need for separating RE according to the service delivery channel. For example, the collection of requirements for implementing a new module on the core system is differentiated from the collection of requirements for the same module on the e-banking platform."

Similarly, organizations developing products focused increasingly on the development of formal product architectures to support the managed evolution of their offerings. This is particularly important whilst technologies change so rapidly that architects essentially "bet on" standards in order to accommodate unknown future changes in technologies and their demand.

Fluidity of Designs. Those interviewed showed an increased appreciation for the fluidity, or continued evolution, of design artifacts. While artifacts have always evolved after use, design teams have traditionally viewed a project as "complete" at some point – normally after software implementation. Informants indicated that this assumption about designs has begun to wane as they recognize that projects often form a single step in an iterative process: "You know as soon as we build a project and deliver it, the day after, we're in the enhancement phase." One strategy for dealing with this evolution was to limit the scope of projects intentionally, with planned and managed releases:

> "You have to set the expectation that what will be delivered will not be exactly what you're looking for. It's not going to be the end; it'll be a step along the way. We are going to build this and then we are going to expand on that capability in subsequent releases. You can't deliver to the end goal right out of the gate…"

Users appropriate artifacts in idiosyncratic ways. Many firms are therefore increasingly cognizant of this evolution and are not attempting to define all requirements ahead of time. Rather they seek to provide additional mechanisms to empower end users to personalize the artifacts. They may build open interfaces to allow evolution in requirements:

> "We're pushing the capability out to the end users and saying don't put us in the middle of it, if you want to figure this out here's the [tool] … the data definition is there, you can select the data that you want to put on the report, how you want it on the report, where you want to gather the data from, what kind of sort sequence, what kind of summary information you want. We're pushing that capability out to the end users so they can self serve just like, you know, companies are pushing capabilities out to their end customers so they can self serve and reduce the cost to serve overall."

In a product development, the challenge is to generate requirements that tolerate "fuzziness," as one product development manager indicated:

> "I don't really understand what the consumers actually prefer and since its change is faster than I can change my [design], how can I design in ways that somebody else can fiddle around with it? … These things are changing so fast it's invention in the hands of the owner, how you design your systems in a way that you make that possible."

Solutions to unknown user-led evolution involved increased reliance on interface standards and standardized "platforms" embedded into products. Rather than specifying specific functional requirements, as these can not be known, standard interfaces that may accommodate multiple add-ons have become the main object of requirements.

Interdependent Complexity. The final persistent theme was the perception of increased complexity. This was associated with varying technologies, requirements efforts, and design processes. While it has long been observed that complexity is an *essential* rather than an *accidental* property of systems development [146], most participants felt that the complexity they now encounter has increased significantly:

> "You know, certainly your ability to manage the sheer quantity of requirements and to be able to test those requirements. Usually, on these large complex systems where you're talking about hundreds and hundreds of different applications, your ability to test in an integrated fashion, all of those requirements is very, very hard and very costly."

However, the level at which such complexity emerges has also shifted – from the internal complexity of software development to the integrative complexity of interdependent systems:

> "I have not seen from my area, the complexity be nearly as mammoth as like when we did MRP back in the mid-1980s, where we had hundred and hundreds and hundreds of programs integrated into three hundred batch jobs and all synchronized to run in a 48-hour period for regenerative MRP once a week - not to count the dailys and weeklys. I don't see that the IT projects have that level of complexity in terms of tying things off. What I do see is that the complexity from systems integration and how to secure at the application level the appropriate details, has gotten a lot more complicated."

Despite the deployment of modular designs and architectures, complexity is now substantially greater because of the large number of interdependent systems, increased number of new systems that must be integrated with legacy infrastructure, and the sheer magnitude of integration-oriented requirements given all of the themes. Indeed, complexity in its various manifestations is perhaps the main fundamental motif that cuts across all the issues raised in this study.

5   Discussion

The findings from this study pose a series of engaging questions to researchers interested in design requirements phenomena. Introspectively, we must ask ourselves the degree to which the assumptions that underlie current research traditions have inhibited us from understanding and attending to the ways in which requirements work is actually accomplished. Turning to the observed processes and factors emerging in contemporary design practice, we may ask how best to make sense of the diverse forces that affect designers. Finally, we must consider the avenues that are opening up

for productive inquiry around emergent requirements themes and how these are related to existing threads within the research community.

5.1   Current Practice and the Research Tradition

Our results point to a great deal of progress associated with requirements practices, but they also signal a changing emphasis towards infrastructural, organizational, and environmental complexity. On the one hand, systems development organizations now employ many of the principles that researchers have been advocating for years, such as formal validation and sign-off, enterprise and functional modeling, user involvement, explicit risk management, and the use of CASE tools. Furthermore, many are looking to increase the degree of structure in their requirements practices. On the other hand, there appear to be a number of inconsistencies between the way practitioners view requirements processes and the way requirements topics are treated in the research community. For example, practitioners do not make many of the distinctions that researchers favor (e.g., phases of the requirements process and requirements types); they often don't refer to formal methodologies and their practices are not consistent with a singular methodology; and, practitioners de-emphasize formal modeling's role in discovery and validation/verification, betraying a continued preference for natural language to capture and communicate requirements.

While our findings reveal that academics may be leading the practitioner community in many respects, they also indicate that some of the assumptions reflected in the literature are not shared by design practitioners. Thus, we must ask ourselves a challenging question – has the practitioner community simply not yet caught up, or are many of our scholarly assumptions not relevant to requirements practice? One of the seemingly more problematic assumptions concerns the distinction between facets in the requirements process. While the research community has long acknowledged the importance of iteration and the interplay of processes in requirements efforts [3], most requirements texts outline a temporal sequence of requirements tasks or phases [e.g., 10, 12, 39]. Furthermore, the structure of the research discourse is clearly bounded by distinct phases, with researchers focusing largely on a single facet of the process (e.g., specification and modeling) in their research efforts. The activities we have labeled as "discovery," "specification," and "validation & verification" [39] have been framed a number of different ways in the literature, yet our interviews indicate that these distinctions are rarely made in practice. Discovery, specification, and validation/verification practices often happen simultaneously, and are largely mediated through natural-language and various artifacts. At the very least, they are so closely related that the practical distinctions between these tasks have become difficult to draw. Iteration continues throughout design, as discovery revolves and verification never really ceases, even after the delivery of a system. Throughout this process, interactions with user communities are conducted through natural language and design artifacts of different maturity.

A second key assumption concerns the estimated value of formal modeling techniques. Formal modeling remains squarely within the development community, and it does not appear to have been appropriated as a communication tool with users.[9]

---
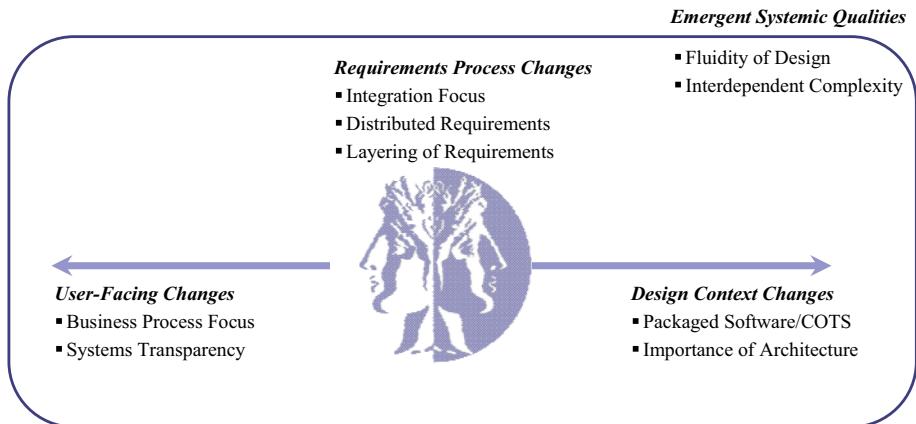
[9] One exception to this observation is business process flow diagrams that are widely used to mediate developer-user communication (in relevant applications).

Rather, to the extent that formal models are used, they are derived from natural language representations and created after requirements have been generated and approved. The academic literature on modeling seeks to make the natural language communication between users and developers more precise through formal models, but this approach does not appear to be followed by designers themselves. With the significant adoption of use cases, we find that greater precision in designer-user communication is indeed desired, but it is fulfilled through semi-structured natural language exchanges. Thus, we may question the assumption that formal modeling effectively supports interactions between distinct stakeholders or bridges the communicative gaps between the design team and other stakeholders [4, 80, 81, 93, 147]. Perhaps a more appropriate pursuit would be to augment formal models with well organized and structured natural language representations.

In our attempt to re-evaluate assumptions that are challenged by contemporary practice, it is important that we understand the emerging patterns of requirements processes within complex designs. Much of the academic literature is rooted in the paradigm of the 1970s and 1980s, where systems were developed from scratch (in distinct, typically "greenfield" projects) in order to support a specific set of operational activities. Two primary groups were involved: those that would use the artifact and those charged with the development of it. Within this context, it was commonly understood that requirements for the system were in the heads of the users, and it was up to the developers to elicit these requirements to guide further system design. As has been extensively illustrated, this assumption was rife with difficulty, as multiple stakeholder groups were affected by the system, and requirements were rarely easily accessible and had a tendency to change over time while systems evolved (e.g., [146]). In subsequent years, strategies have been put forward to overcome some of these challenges. Techniques such as prototyping [54] and ethnographic methods have been adopted to help designers move beyond the limitations of traditional methods [48, 53]. Yet, these approaches remain well within the traditional paradigm, as they are methods to improve the extraction of requirements from users for increased formalization by developers.

5.2   Understanding Emergent Forces in Requirements Practice

Our data suggest a number of different trends that present challenges to the traditional outlook. Systems are no longer created for a specific set of activities within an organization's functional silos. Systems are intended to support cross-organizational and inter-organizational business processes or to offer multiple functions over a product's life-cycle. Multipurpose, expandable devices for a wide array of applications abound. Systems increasingly connect to each other, become integrated, and system boundaries and associated connections are made invisible to users. Greenfield efforts are nearly extinct, and stakeholders are no longer a fixed or easily-identifiable set of individuals. Commercial-off-the-shelf (COTS) applications and modular architectures dominate design landscapes as firms look to buy rather than build due to lower cost and expectation of higher quality. Development never ends. When one level of complexity becomes black-boxed, additional layers of complexity emerge. No longer can we look at the requirements process as a dialogue where designers extract requirements from users. Rather, designers and design teams can be best viewed as reconciling multiple forces and perspectives: negotiating with an ever-expanding set of

Fig. 1. An Emerging Requirements Landscape

stakeholders, merging and connecting evolving architectures, addressing continuing technological changes, and mitigating the complexity associated with marrying these threads. Figure 1 illustrates one attempt to organize and structure the diverse forces that drive this emerging requirements landscape.

This new requirements landscape evokes again the Janus-faced nature of requirements practice. The designer is caught between two fluctuating worlds, where he or she is simultaneously looking backward towards the shifting sands of stakeholder needs, while looking forward to evolving platforms and technological architectures and the concrete design implications that they bear. Within this context, we observe themes that speak to the changing ways in which stakeholders encounter, and interact with, the software-intensive artifacts that populate their work and home environments. These "User-Facing Changes" reflect a shift in expectations on the part of individual users and groups. As they accept novel technologies into more and more facets of their lives, they expect the boundaries between artifacts to fade into the background, minimizing the cognitive effort required to transition between tools within their socio-technical ecology. We assert that the observed themes of *Business Process Focus* and *Systems Transparency* embody these changes in the users' experience.

At the other end of our Janus's line of sight, we observe significant changes in the design contexts within which design teams must maneuver. "Design Context Changes" reflect a fundamental shift in the baseline conditions for all contemporary software-intensive design efforts. The decline of traditional development activities and the critical of contextual constraints have dramatically altered the process of design itself. The rising emphasis on *Packaged Software/COTS* and the centrality of *Information Architectures* are two clear manifestations of this observed shift.

Between the changing expectations of users and the altered constraints on the broader design environment sits the Janus of requirements. In an effort to marry these diverse forces, the process of requirements has itself been transformed. Current requirements practices reflect a more heterogeneous and multi-faceted phenomenon than is often reflected in the treatment by the research community. A significant *Integration Focus*, the management of requirements from a varied set of sources

Table 6. Proposed Emergent Avenues for Requirements Research

Key Themes	Overall	Selected Topics from the Requirements Research Tradition		
		Discovery/ Elicitation	*Specification & Modeling*	*Validation & Verification*
User-Facing Changes				
Business Processes Focus	Understanding requirements processes when the technology fades into the business context	Assessing the effectiveness of discovery techniques in business process design efforts	Coordinating enterprise, business process, and functional models	Evaluating adherence to strategic business processes
Systems Transparency	Capturing benefits and challenges posed by the transparency of systems	Determining the sources of user expectations with respect to systems interoperability	Capturing the needs for transparency as a functional requirement in modeling methods	Evaluating new prototyping and simulation capabilities
Requirements Practice Changes				
Integration Focus	Determining who is responsible for functional requirements when traditional development is less relevant	Determining appropriate stakeholders for articulating integration-oriented requirements	Managing heterogeneous models from a variety of systems	Understanding the processes employed for ensuring satisfactory integration testing
Distributed Requirements	Effective management of requirements in a distributed cognitive process	Aggregation of requirements identified by multiple parties; Coordinating among different elicitation activities	Managing heterogeneous models from a variety of systems	Understanding validation and verification of aggregated requirements
Layers of Requirements	Identifying new bases for requirements layering in embedded systems and other contexts	Assessing differences in effectiveness of discovery approaches based on layers observed	Assessing what layers are most amenable to natural language vs. modeling methods	Developing mechanisms for requirements checking across multiple layers

Table 6. (*continued*)

Key Themes	Overall	Selected Topics from the Requirements Research Tradition		
		Discovery/ Elicitation	*Specification & Modeling*	*Validation & Verification*
Design Context Changes				
Packaged Software Orientation	Understanding the role of prognostication in requirements - who will be the winner in a given market?	Capturing the potential for knowledge gains through the use of vendors	Marrying current state and future state models to stated vendor preferences	Determining the degree to which COTS address platform-agnostic requirements
Centrality of Architecture	Identifying the ways in which architecture impacts requirements practice	Observing how architecture sets constraints on discovery processes	Models driving the requirements rather than requirements driving model development	Architecture as the arbiter of appropriateness and quality
Emergent Systemic Qualities				
Fluidity of Design	Requirements in the context of partial designs Run-time evolution of requirements	Evolution of stakeholder needs based on continued system use	Management of models over generation of design iteration	Managing stakeholder expectations and validation in fluid design contexts
Interdependent Complexity	Paradoxes of reducing and increasing complexity at different levels of analysis	Determining appropriate levels of stakeholder involvement in complex design efforts	Managing heterogeneous models from a variety of systems	Requirements testing methods for application in highly-interdependent environments

(i.e., *Distributed Requirements*), and the emergence of novel bases for the *Layering of Requirements* all embody the changes to be observed in modern requirements processes.

In addition to the changes observed in user experiences, design contexts, and requirements processes, there are broader contextual characteristics that have emerged from, and in turn engendered, the other themes we have highlighted here. These "Emergent Systemic Qualities" call our attention to important new areas for exploration and rigorous inquiry. The rise of *Interdependent Complexity* is a force that can be seen at all levels of the design process – from the expectations of users to the practical, technical demands reflected in novel artifacts. As designers have struggled to control complexity at one level of work, it has re-emerged at another level. Complexity has become an intrinsic part of design and affects both stakeholder behaviors and other extrinsic forces. Similarly, the recognition of the *Fluidity of Design* in the organizations that participated in this study suggests a new maturity of understanding with respect to the impermanence and evolutionary potential that is central to modern software-intensive systems design.

A shifting focus toward integration and evolution rather than elicitation and documentation highlights the increasingly creative role that designers must play in actively co-producing requirements and artifacts, rather than simply charting out needs that are "out there" *a priori*. This observation has multiple implications for design research and calls for an expansion of the realm of requirements research to address broader organizational aspects of design and the requirements processes.

5.3 New Avenues for Requirements Research

Perhaps most importantly for the present discussion, the observations and phenomena presented in this study call our attention to a wide array of new avenues for requirements research. Each of the key findings reflects an issue that warrants additional exploration. To close the research-practice gap within the requirements arena, some of the more counter-intuitive (and contra-prescriptive) findings from the assessment of current practice should be investigated. Similarly, each of the key themes that emerged from the analysis should be thoroughly examined to improve our understanding of how these forces are shaping today's design environments. The current volume is intended to initiate just such an agenda-setting perspective. Several of the key themes noted in this study are explored in greater depth and nuance in the chapters of this book. For example, a focus on business processes, the fluidity of contemporary design, and the challenges of interdependent complexity are considered repeatedly throughout the volume. Other facets of the emergent requirements landscape have yet to be approached. For example, the distributed nature of requirements processes and sources, the role of centralized architectures in both addressing and driving requirements efforts, and the demands for greater systems transparency promise fertile ground for research in the coming years.

The challenge, of course, is determining how we can draw upon the research tradition while remaining open to the new phenomena at hand. We have suggested that some of the fundamental assumptions that undergird the requirements literature may need to be reconsidered as we look to the future, but how can we effectively leverage the work that has come before this point?

In the opinion of the research team, the framework provided by the extant research may still provide a useful lens for directing the attention of researchers. While distinctions between facets of the requirements process have blurred, the fundamental concerns upon which they are built remain: What does the design need to have (Discovery)? How can we render an unspoken vision in an explicit form around which multiple individuals and groups can gravitate (Specification)? How can we as designers know when we are on the right track and persuade others of the same (Validation & Verification)? Each of these high-level conceptual challenges offers a perspective that can be fruitfully applied to the emergent phenomena observed among practicing design teams. In Table 5 (provided in the Appendix), we illustrate how traditional requirements research foci and the key themes outlined in this study can be combined to open up a wide range of prospective channels for requirements research in the coming years. Clearly, the issues presented in the table are far from exhaustive, as they are intended merely to illustrate the types of inquiries that are suggested by the framework.

6   Conclusion

In this study, we have reflected upon the degree to which the literature on requirements appropriately reflects current design practices across a variety of organizations. We find that recent decades have seen a significant amount of progress in orienting designers to many critical requirements-based considerations, but we also observe a number of issues where current practices are less than consistent with the assumptions of the academic literature. Moreover, we identify a number of macro-level emerging trends across a variety of modern requirements practices. We conclude with a characterization of complex large-scale requirements efforts as an exercise in balancing constraints and opportunities in multiple directions at the same time. Designers, like the Roman god Janus, must simultaneously look to the often-ambiguous needs of stakeholders and attend to the practical demands of the design environment. In so doing, they have changed the face of requirements practice, and ushered in a period of expanding complexity and evolutionary dynamics in design. Contemporary designers construct requirements in relation to existing systems and practices, rather than simply eliciting them as much of the literature implies.

Using this empirical research as a backdrop, we now embark on an effort to make sense of these issues. In a series of two workshops over two years, some of the world's top thinkers on requirements issues will be assembled to discuss the implications of our findings, to address issues we have not yet considered, and to broadly assess the direction of requirements research going forward. During this time we plan to assess the current practices of both the research and the practitioner communities in light of emerging trends in requirements activity, technologies, and organizational environments, with an eye to the following questions: Is the way researchers think about requirements adequate going forward? Do our assumptions about requirements practices need to change? Where should research into requirements focus or expand to capture emerging trends in system design? Thus, the current study is intended as food for thought. We are confident that tremendous insights lie ahead.

References

1. Simon, H.: The Sciences of the Artificial. The MIT Press, Cambridge (1996)
2. Norman, D.A.: The Design of Everyday Things. Basic Books (2002)
3. Nuseibeh, B., Easterbrook, S.: Requirements Engineering: A Roadmap. In: Proceedings of the conference on The future of Software engineering, pp. 35–46. ACM Press, New York (2000)
4. Ross, D.T., Schoman Jr., K.E.: Structured Analysis for Requirements Definition. IEEE Transactions on Software Engineering 3, 6–15 (1977)
5. Guinan, P.J., Cooprider, J.G., Faraj, S.: Enabling Software Development Team Performance During Requirements Definition: A Behavioral Versus Technical Approach. Information Systems Research 9, 101–125 (1998)
6. Mumford, E.: Effective Systems Design and Requirements Analysis: The ETHICS Approach. Macmillan, Basingstoke (1995)
7. Montazemi, A., Conrath, D.: The Use of Cognitive Mapping for Information Requirements Analysis. MIS Quarterly 10, 45–56 (1986)
8. Davis, G.: Strategies for Information Requirements Determination. IBM Systems Journal 21, 4–30 (1982)
9. Yadav, S., Ralph, R., Akemi, T., Rajkumar, T.: Comparison of analysis techniques for information requirement determination. Communications of the ACM 31, 1090–1097 (1988)
10. Wiegers, K.: Software Requirements. Microsoft Press, Redmond (1999)
11. Crowston, K., Kammerer, E.: Coordination and Collective Mind in Software Requirements Development. IBM Systems Journal 37, 227–246 (1998)
12. Sommerville, I., Sawyer, P.: Requirements Engineering: A Good Practice Guide. John Wiley & Sons, Inc., New York (1997)
13. Kotonya, G., Sommerville, I.: Requirements Engineering: Processes and Techniques. John Wiley & Sons, New York (1998)
14. Hoffer, J., George, J., Valacich, J.: Modern Systems Analysis and Design. Prentice Hall, Upper Saddle River (2001)
15. Whitten, J.L., Bentley, L.D.: Systems Analysis and Design Methods. McGraw-Hill/Irwin, New York (2007)
16. Ulrich, K., Eppinger, S.: Product design and development. McGraw-Hill, New York (1995)
17. van Lamsweerde, A.: Requirements Engineering in the Year 00: A Research Perspective. In: Proceedings of the 22nd international conference on Software engineering, pp. 5–19 (2000)
18. Zave, P.: Classification of research efforts in requirements engineering. ACM Computing Surveys 29, 315–321 (1997)
19. Bell, T., Thayer, T.: Software requirements: Are they really a problem? In: Proceedings of the 2nd international conference on Software engineering, pp. 61–68 (1976)
20. Boehm, B.: Software Engineering Economics. Prentice Hall PTR, Upper Saddle River (1981)

21. Boehm, B., Papaccio, P.: Understanding and controlling software costs. IEEE Transactions on Software Engineering 14, 1462–1477 (1988)
22. Scharer, L.: Pinpointing Requirements. Datamation 27, 139–151 (1981)
23. Orr, K.: Agile requirements: opportunity or oxymoron? Software, IEEE 21, 71–73 (2004)
24. Bergman, M., King, J., Lyytinen, K.: Large-Scale Requirements Analysis Revisited: The need for Understanding the Political Ecology of Requirements Engineering. Requirements Engineering 7, 152–171 (2002)
25. Lindquist, C.: Fixing the Requirements Mess. CIO Magazine, 52–60 (2005)
26. Group, T.S.: The CHAOS Report. West Yarmouth, MA (1995)
27. Aurum, A., Wohlin, C.: Requirements Engineering: Setting the Context. In: Aurum, A., Wohlin, C. (eds.) Engineering and Managing Software Requirements, pp. 1–15. Springer, Berlin (2005)
28. Leffingwell, D., Widrig, D.: Managing Software Requirements: A Unified Approach. Addison-Wesley Professional, Reading (1999)
29. Hickey, A., Davis, A.: Elicitation technique selection: how do experts do it? In: Proceedings of 11th IEEE International on Requirements Engineering Conference, pp. 169–178 (2003)
30. Keil, M., Cule, P.E., Lyytinen, K., Schmidt, R.C.: A framework for identifying software project risks. Communications of the ACM 41, 76–83 (1998)
31. Kaindl, H., Brinkkemper, S., Bubenko Jr., J.A., Farbey, B., Greenspan, S.J., Heitmeyer, C.L., Leite, J.C.S.P., Mead, N.R., Mylopoulos, J., Siddiqi, J.: Requirements Engineering and Technology Transfer: Obstacles, Incentives and Improvement Agenda. Requirements Engineering 7, 113–123 (2002)
32. Zowghi, D., Coulin, C.: Requirements Elicitation: A Survey of Techniques, Approaches, and Tools. Engineering and Managing Software Requirements (2005)
33. Siddiqi, J., Shekaran, M.C.: Requirements Engineering: The Emerging Wisdom. IEEE Software 13, 15–19 (1996)
34. Berry, D.M., Lawrence, B.: Requirements Engineering. IEEE Software 15, 26–29 (1998)
35. Zave, P., Jackson, M.: Four Dark Corners of Requirements Engineering. ACM Transactions on Software Engineering and Methodology 6, 1–30 (1997)
36. Wiegers, K.E.: Read My Lips: No New Models! IEEE Software 15, 10–13 (1998)
37. Davis, A.M., Hickey, A.M.: Requirements Researchers: Do We Practice What We Preach? Requirements Engineering 7, 107–111 (2002)
38. Davis, A.M.: Software Requirements: Objects, Functions, and States. Prentice-Hall, Upper Saddle River (1993)
39. Loucopoulos, P., Karakostas, V.: System Requirements Engineering. McGraw-Hill, Inc., New York (1995)
40. Hull, E., Jackson, K., Dick, J.: Requirements Engineering. Springer, London (2005)
41. Jackson, M.: Software Requirements & Specifications: A Lexicon of Practice, Principles and Prejudices. ACM Press/Addison-Wesley Publishing Co., New York (1995)
42. Windle, D.R., Abreo, L.R.: Software Requirements Using the Unified Process: A Practical Approach. Prentice Hall PTR, Upper Saddle River (2002)
43. Wieringa, R.J.: Requirements Engineering: Frameworks for Understanding. John Wiley & Sons, Inc., New York (1996)
44. Vessey, I., Conger, S.A.: Requirements Specification: Learning Object, Process, and Data Methodologies. Communications of the ACM 37, 102–113 (1994)
45. Dorfman, M.: Software Requirements Engineering. In: Thayer, R.H., Dorfman, M. (eds.) Software Requirements Engineering, pp. 7–22. IEEE Computer Society Press, Los Alamitos (1997)

46. Ghezzi, C., Jazayeri, M., Mandrioli, D.: Fundamentals of Software Engineering. Prentice Hall, Englewood Cliffs (1991)
47. Boehm, B.W.: Verifying and validating software requirements and design specifications. IEEE Software 1, 75–88 (1984)
48. Goguen, J., Linde, C.: Techniques for Requirements Elicitation. Requirements Engineering 93, 152–164 (1993)
49. Agarwal, R., Tanniru, M.T.: Knowledge Acquisition Using Structured Interviewing: An Empirical Investigation. Journal of Management Information Systems 7, 123–140 (1990)
50. Wright, G., Ayton, P.: Eliciting and modelling expert knowledge. Decision Support Systems 3, 13–26 (1987)
51. Byrd, T., Cossick, K., Zmud, R.: A Synthesis of Research on Requirements Analysis and Knowledge Acquisition Techniques. MIS Quarterly 16, 117–138 (1992)
52. Beyer, H., Holtzblatt, K.: Apprenticing with the customer. Communications of the ACM 38, 45–52 (1995)
53. Viller, S., Sommerville, I.: Social Analysis in the Requirements Engineering Process: From Ethnography to Method. In: International Symposium on Requirements Engineering. IEEE, Limerick, Ireland (1999)
54. Alavi, M.: An Assessment of the Prototyping Approach to Information Systems Development. Communications of the ACM 27, 556–563 (1984)
55. Glass, R.L.: Searching for the Holy Grail of Software Engineering. Communications of the ACM 45, 15–16 (2002)
56. Hickey, A.M., Davis, A.M.: A Unified Model of Requirements Elicitation. Journal of Management Information Systems 20, 65–84 (2004)
57. Boland, R.J.: The process and product of system design. In: Management Science, IN-FORMS, Institute for Operations Research, vol. 24, p. 887 (1978)
58. Ropponen, J., Lyytinen, K.: Components of software development risk: how to address them? A project manager survey. IEEE Transactions on Software Engineering 26, 98–112 (2000)
59. Jørgensen, M., Sjøberg, D.I.K.: The Impact of Customer Expectation on Software Development Effort Estimates. International Journal of Project Management 22, 317–325 (2004)
60. Cook, S., Brown, J.: Bridging Epistemologies: The Generative Dance between Organizational Knowledge and Organizational Knowing. Organization Science 10, 381–400 (1999)
61. Stewart, D., Shamdasani, P.: Focus Groups: Theory and Practice. Sage Publications, Newbury Park (1990)
62. Boehm, B.: A spiral model of software development and enhancement. ACM SIGSOFT Software Engineering Notes 11, 22–42 (1986)
63. Boehm, B.: Verifying and validating software requirements and design specifications. Software risk management table of contents, 205–218 (1989)
64. Garfinkel, H.: The origins of the term 'ethnomethodology'. Ethnomethodology, 15–18 (1974)
65. Lynch, M.: Scientific practice and ordinary action: ethnomethodology and social studies of science. Cambridge University Press, Cambridge (1993)
66. Siddiqi, J.: Challenging universal truths of requirements engineering. IEEE Software 11, 18–19 (1994)

67. Shekaran, C., Garlan, D., Jackson, M., Mead, N.R., Potts, C., Reubenstein, H.B.: The Role of Software Architecture in Requirements Engineering. In: First International Conference on Requirements Engineering, pp. 239–245. IEEE Computer Society Press, San Diego (1994)

68. Baldwin, C.Y., Clark, K.B.: Design Rules, The Power of Modularity, vol. 1. MIT Press, Cambridge (2000)

69. Palmer, J.D.: Traceability. In: Thayer, R.H., Dorfman, M. (eds.) Software Requirements Engineering. IEEE Computer Society Press, Los Alamitos (1997)

70. Ramesh, B.: Factors influencing requirements traceability practice. Communications of the ACM 41, 37–44 (1998)

71. Wieringa, R.J.: An Introduction to Requirements Traceability. Faculty of Mathematics and Computer Science, Vrije Universiteit 1995

72. Hsia, P., Davis, A.M., Kung, D.C.: Status Report: Requirements Engineering. IEEE Software 10, 75–79 (1993)

73. van Lamsweerde, A.: Formal Specification: A Roadmap. In: Conference on the Future of Software Engineering, pp. 147–159. ACM Press, Limerick (2000)

74. Balzer, R., Goldman, N., Wile, D.: Informality in Program Specifications. IEEE Transactions on Software Engineering 4, 94–103 (1978)

75. Ackerman, M., Hadverson, C.: Reexamining organizational memory. Communications of the ACM 43, 58–64 (2000)

76. Reubenstein, H.B., Waters, R.C.: The Requirements Apprentice: Automated Assistance for Requirements Acquisition. IEEE Transactions on Software Engineering 17, 226–240 (1991)

77. Gervasi, V., Nuseibeh, B.: Lightweight validation of natural language requirements. Software Practice and Experience 32, 113–133 (2002)

78. Goldin, L., Berry, D.M.: AbstFinder, A Prototype Natural Language Text Abstraction Finder for Use in Requirements Elicitation. Automated Software Engineering 4, 375–412 (1997)

79. Ryan, K.: The role of natural language in requirements engineering. In: IEEE International Symposium on Requirements Engineering, San Diego, CA, pp. 240–242 (1993)

80. Borgida, A., Greenspan, S., Mylopoulos, J.: Knowledge Representation as the Basis for Requirements Specifications. IEEE Computer 18, 82–91 (1985)

81. Mylopoulos, J.: Information Modeling in the Time of the Revolution. Information Systems 23, 127–155 (1998)

82. Checkland, P.: Systems Thinking, Systems Practice. John Wiley & Sons, Inc., New York (1981)

83. Checkland, P., Scholes, J.: Soft Systems Methodology in Action. John Wiley & Sons, Inc., New York (1990)

84. Rumbaugh, J., Jacobson, I., Booch, G.: The Unified Modeling Language Reference Manual. Addison-Wesley Longman, Essex (1998)

85. Scholz-Reiter, B., Stickel, E.: Business Process Modelling. Springer, Secaucus (1996)

86. Scheer, A.-W.: ARIS–Architecture of Integrated Information Systems. Springer, Heidelberg (1992)

87. Leppänen, M.: An Ontological Framework and a Methodical Skeleton for Method Engineering: A Contextual Approach. University of Jyväskylä (2005)

88. Machado, R.J., Ramos, I., Fernandes, J.M.: Specification of Requirements Models. In: Aurum, A., Wohlin, C. (eds.) Engineering and Managing Software Requirements, pp. 47–68. Springer, Berlin (2005)

89. Mylopoulos, J., Chung, L., Yu, E.: From Object-Oriented to Goal-Oriented Requirements Analysis. Communications of the ACM 42, 31–37 (1999)
90. Brand, D., Zafiropulo, P.: On Communicating Finite-State Machines. Journal of the ACM 30, 323–342 (1983)
91. Peterson, J.L.: Petri Nets. ACM Computing Surveys 9, 223–252 (1977)
92. Harel, D.: Statecharts: A Visual Formalism for Complex Systems. Science of Computer Programming 8, 231–274 (1987)
93. Ross, D.T.: Structured Analysis (SA): A Language for Communicating Ideas. Transactions on Software Engineering 3, 16–34 (1977)
94. DeMarco, T.: Structured Analysis and System Specification. Prentice-Hall, Englewood Cliffs (1979)
95. Yourdon, E., Constantine, L.L.: Structured Design. Prentice-Hall, Englewood Cliffs (1979)
96. Sutcliffe, A.G.: Object-oriented systems development: survey of structured methods. Information and Software Technology 33, 433–442 (1991)
97. Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W.: Object-Oriented Modeling and Design. Prentice-Hall, Inc., Englewood Cliffs (1991)
98. Jacobson, I., Christerson, M., Jonsson, P., Overgaard, G.: Object-Oriented Software Engineering: A Use Case Driven Approach, vol. 524. Addison-Wesley, Reading (1992)
99. Bonabeau, E.: Agent-Based Modeling: Methods and Techniques for Simulating Human Systems. Proceedings of the National Academy of Sciences 99, 7280–7287 (2002)
100. Axelrod, R.M.: The Complexity of Cooperation: Agent-Based Models of Competition and Collaboration. Princeton University Press, Princeton (1997)
101. Carley, K., Krackhardt, D.: Cognitive inconsistencies and non-symmetric friendship. Social Networks 18, 1–27 (1996)
102. Carley, K.M.: Computational and mathematical organization theory: Perspective and directions. Computational & Mathematical Organization Theory 1, 39–56 (1995)
103. Epstein, J.M., Axtell, R.: Growing Artificial Societies: Social Science from the Bottom Up. Brookings Institution Press, Washington (1996)
104. van Lamsweerde, A.: Goal-Oriented Requirements Engineering: A Guided Tour. In: Proceedings of the 5th IEEE International Symposium on Requirements Engineering, pp. 249–263. IEEE, Toronto (2001)
105. Dardenne, A., van Lamsweerde, A., Fickas, S.: Goal-Directed Requirements Acquisition. Science of Computer Programming 20, 3–50 (1993)
106. van Lamsweerde, A., Darimont, R., Letier, E.: Managing conflicts in goal-driven requirements engineering. IEEE Transactions on Software Engineering 24, 908–926 (1998)
107. Mylopoulos, J., Chung, L., Nixon, B.: Representing and using nonfunctional requirements: a process-oriented approach. IEEE Transactions on Software Engineering 18, 483–497 (1992)
108. Miller, J.A., Sheth, A.P., Kochut, K.J.: Perspectives in Modeling: Simulation, Database and Workflow. In: Chen, P.P., Akoka, J., Kangassalu, H., Thalheim, B. (eds.) Conceptual Modeling. LNCS, vol. 1565, pp. 154–167. Springer, Heidelberg (1999)
109. Menzel, C., Mayer, R.J.: The IDEF Family of Languages. Handbook on Architectures of Information Systems 1, 209–241 (1998)
110. Booch, G., Rumbaugh, J., Jacobson, I.: The Unified Modeling Language User Guide. Addison Wesley Longman, Redwood City (1999)
111. Kobryn, C.: UML 2001: a standardization odyssey. Communications of the ACM 42, 29–37 (2001)

112. Welke, R.J., Kumar, K.: Method engineering: a proposal for situation-specific methodology construction. In: Cotterman, Senn (eds.) Systems Analysis and Design: A Research Agenda, pp. 257–268. Wiley, Chichester (1992)

113. Harmsen, F., Brinkkemper, S., Oei, H.: Situational Method Engineering for Information System Project Approaches. IFIP Transactions A: Computer Science & Technology, 169–194 (1994)

114. Gray, J., Bapty, T., Neema, S., Tuck, J.: Handling crosscutting constraints in domain-specific modeling. Communications of the ACM 44, 87–93 (2001)

115. Ledeczi, A., Bakay, A., Maroti, M., Volgyesi, P., Nordstrom, G., Sprinkle, J., Karsai, G.: Composing domain-specific design environments. Computer 34, 44–51 (2001)

116. Rossi, M., Ramesh, B., Lyytinen, K., Tolvanen, J.-P.: Managing Evolutionary Method Engineering by Method Rationale. Journal of the AIS 5 (2004)

117. Pohl, K.: Requirement Engineering: An overview. Encyclopedia of Computer Science and Technology. Marcel Dekker, New York (1996)

118. Easterbrook, S.: Handling Conflict between Domain Descriptions with Computer-Supported Negotiation. Knowledge Acquisition 3, 255–289 (1991)

119. Grünbacher, P., Seyff, N.: Requirements Negotiation. In: Aurum, A., Wohlin, C. (eds.) Engineering and Managing Software Requirements, pp. 143–162. Springer, Berlin (2005)

120. Berander, P., Andrews, A.: Requirements Prioritization. In: Aurum, A., Wohlin, C. (eds.) Engineering and Managing Software Requirements, pp. 69–94. Springer, Berlin (2005)

121. Regnell, B., Höst, M., Dag, J.N.: An Industrial Case Study on Distributed Prioritisation in Market-Driven Requirements Engineering for Packaged Software. Requirements Engineering 6, 51–62 (2001)

122. Feather, M.S., Fickas, S., Finkelstein, A., van Lamsweerde, A.: Requirements and Specification Exemplars. Automated Software Engineering 4, 419–438 (1997)

123. Robinson, W., Volkov, V.: Supporting the Negotiation Life Cycle. Communications of the ACM 41, 95–102 (1998)

124. Robinson, W.N.: Negotiation behavior during requirement specification. In: Proceedingsof the 12th International Conference on Software Engineering, pp. 268–276. IEEE Computer Society Press, Nice (1990)

125. Fisher, R., Ury, W.: Getting to Yes: Negotiating without Giving In, Boston, MA (1991)

126. Boehm, B.W., Egyed, A.: WinWin Requirements Negotiation Processes: A Multi-Project Analysis. In: 5th International Conference on Software Processes (1998)

127. Grünbacher, P., Briggs, R.O.: Surfacing Tacit Knowledge in Requirements Negotiation: Experiences using EasyWinWin. In: 34th Hawaii International Conference on System Sciences. IEEE, Los Alamitos (2001)

128. Robinson, W.N., Fickas, S.: Automated Support for Requirements Negotiation. In: AAAI 1994 Workshop on Models of Conflict Management in Cooperative Problem Solving, pp. 90–96 (1994)

129. Kotonya, G., Sommerville, I.: Requirements Engineering with Viewpoints. Software Engineering Journal 11, 5–18 (1996)

130. Leite, J.C.S.P., Freeman, P.A.: Requirements Validation through Viewpoint Resolution. IEEE Transactions on Software Engineering 17, 1253–1269 (1991)

131. Knight, J.C., Myers, E.A.: An Improved Inspection Technique. Communications of the ACM 36, 51–61 (1993)

132. Yourdon, E.: Structured Walkthroughs. Yourdon Press, Upper Saddle River (1989)

133. Soni, D., Nord, R., Hofmeister, C.: Software Architecture in Industrial Applications. In: 17th International Conference on Software Engineering, pp. 196–207. ACM Press, New York (1995)

134. Glaser, B.G., Strauss, A.L.: Discovery of Grounded Theory: Strategies for Qualitative Research. Aldine Publishing Company, Chicago (1967)
135. Eisenhardt, K.: Building Theories from Case Study Research. Acad. Manage. Rev. 14, 532–550 (1989)
136. Lyytinen, K., Mathiassen, L., Ropponen, J.: Attention Shaping and Software Risk - A Categorical Analysis of Four Classical Risk Management Approaches. Information Systems Research 9, 233–255 (1998)
137. Kruchten, P.: The Rational Unified Process: An Introduction. Pearson Education, Boston (2003)
138. Vessey, I., Sravanapudi, A.P.: CASE tools as collaborative support technologies. Communications of the ACM 38, 83–95 (1995)
139. Stohr, E., Zhao, J.: Workflow Automation: Overview and Research Issues. Information Systems Frontiers 3, 281–296 (2001)
140. Becker, J., Rosemann, M., von Uthmann, C.: Guidelines of Business Process Modeling. In: van der Aalst, W., Desel, J., Oberweis, A. (eds.) Business Process Management - Models, Techniques, and Empirical Studies, pp. 30–49. Springer, Heidelberg (2000)
141. King, W.R.: IT Capabilities, Business Processes, and Impact on the Bottom Line. In: Brown, C.V., Topi, H. (eds.) IS Management Handbook. Auerbach Publications, Boca Raton (2003)
142. Brynjolfsson, E., Hitt, L.M.: Beyond Computation: Information Technology, Organizational Transformation and Business Performance. In: Malone, T.W., Laubacher, R., Scott Morton, M.S. (eds.) Inventing the organizations of the 21st century, pp. 71–99. MIT Press, Cambridge (2003)
143. Jarke, M., Pohl, K.: Requirements Engineering in 2001 (Virtually) Managing a Changing Reality. Software Engineering Journal 9, 257–266 (1994)
144. Damian, D., Eberlein, A., Shaw, M., Gaines, B.: An exploratory study of facilitation in distributed requirements engineering. Requirements Engineering 8, 23–41 (2003)
145. Grünbacher, P., Braunsberger, P.: Tool Support for Distributed Requirements Negotiation. Cooperative methods and tools for distributed software processes, 56–66 (2003)
146. Brooks, F.P.: No Silver Bullet: Essence and Accidents in Software Engineering. IEEE Computer 20, 10–19 (1987)
147. Greenspan, S., Mylopoulos, J., Borgida, A.: On Formal Requirements Modeling Languages: RML Revisited. In: Proceedings of the 16th International Conference on Software Engineering (ICSE-16), Sorrento, Italy, pp. 135–147 (1994)

Section 1: Fundamental Concepts of Design

Kalle Lyytinen

The concept of design and the concept of requirements are intricately related and conceptually interdependent. Requirements are the means for capturing the "intent" or "constraints" of the design. What gets designed, how it gets designed, and why it is designed, are all shaped by our notion of requirements. Traditionally, most of the software engineering and system design literature has approached requirements as necessary inputs into the design process that may be revised based on the feedback obtained during the process (see Hansen et al. 2008, and Cheng and Atlee 2008, this volume). Requirements provide necessary and sufficient knowledge bases upon which to build a system, and to validate whether the design process was successful in providing a solution that conforms to the requirements. Most requirements are treated, accordingly, as fixed statements of desirable, either functional or emergent, system properties, or as sets of constraints to which the delivered system must conform. This relatively axiomatic view of requirements as "a set of design axioms" or goals that precedes the "proof" of design has widely dominated the research discourse in requirements engineering. It has also dominated much of the practices of software development as reflected in published standards of requirements discovery and specification and reviewed in Hansen et al. 2008.

If some of the underlying ideas related to design context, system scope, and the nature of systems to be designed is changing, then the prevailing understanding of the requirements is also in need of revision. In this part of the book we will review some of the emerging, revised views of design, with an emphasis on how these views may shape our understanding of requirements: their role in design and how they are discovered during the design. To this end, in this section on "fundamental concepts of design" we have included four articles that scrutinize the concept of design in terms of the nature of the design target, the definition of the design and what needs to be included in it, the organization and process of generating designs, and the specific nature of some of the things which are designed and included in information systems.

The first article by Isabelle Reymen and Georges Romme titled "The Evolution of Design Requirements in the Trajectory of Artificiality: A Research Agenda" is an ambitious attempt to develop and relate existing views of software design and requirements into a broader movement of "design theory". They describe the evolution of design and requirements thinking based on Krippendorf's recent review of design theory. Accordingly, they argue that Krippendorff's trajectory of artificiality shows an increasing dematerialization and human-centeredness of artifacts to be designed – designers move from the creation of things to the creation of social conditions. Based on a review of the design literature, that covers two major design journals, they observe that current "theorizing" and "advancing" the design of socio-technical systems tends to be situated on the level of multi-user systems and networks, whereas more complex artifacts like projects (e.g. like designing a Product Life-cycle Management system) and discourses (e.g. Web 2.0 platforms) receive limited attention in current requirements thinking. This has significant implications for what types of

requirements receive attention in the literature, why certain requirements get accepted, and how one designs with them.

In the second article, Paul Ralph and Yair Wand make "A Proposal for a Formal Definition of the Design Concept." They justify this endeavor by noting that most attempts to formulate design have been haphazard and not systematically developed. Simply, we lack a common base for research on design as we do not have good "ontology" of what design is and what it is not. We also have difficulties in differentiating between requirements concepts and design concepts. Their definition incorporates seven elements: agent, object, environment, goals, primitives, requirements and constraints. Accordingly, design projects are temporal trajectories of work systems that include human agents, who work to design systems for stakeholders, and use resources and tools to accomplish this task. They demonstrate how the design definition can be useful how to classify design knowledge and how the derived conceptual model of design can be used to classify design approaches, which have very different ways to handle and recognize requirements.

The third article by Raghu Garud, Sanjay Jain, and Philipp Tuertscher titled "Incomplete by Design and Designing for Incompleteness" addressed the challenge of completeness in design which is typically translated as "system requirements need to be complete" in the literature. They argue that this idea is a reflection of scientist understanding of design, which is adequate if you assume that the reality you deal with remains the same and is covered by unchanging laws (like computing the values of a sin function). The situation changes, however, if we enter a dynamic environment that changes continually or a situation where the system, by being embedded into the environment, changes the behaviors of the environment. In such a situation we deal with incomplete designs and incomplete requirements. Overall, we need to develop concepts of design that assume and accept incompleteness, and offer means to deal with such incompleteness. They propose a pragmatic view of design where design has a generative element in that one cannot separate the design outcomes from the process, and where design is both the outcome and medium of stakeholder action.

In the fourth article by Sal March and Gove Allen titled "Challenges in Requirements Engineering: An Ontology of the Artificial for Conceptual Modeling" poses a challenge to traditional view of design. They claim that the domains for which information systems are developed deal primarily with social constructions—conceptual or symbolic objects and attributes created by human intentions and for human purposes. Accordingly, information systems play an active role in these domains: they document the creation of new conceptual objects, record and ascribe values to their attributes, initiate actions within the domain, track activities performed and infer conclusions based on the application of rules that govern how the domain is affected when socially-defined and identified *causal events* occur. Therefore, high level designs and their representations in various conceptual models deal with a different type of reality when compared with a design of a machine or a cup- they shape language, thought and affect cognition, and organizational action. They do not have similar physical characteristics as the design of things. Therefore conceptual modeling grammars aimed at representing the requirements or designs for information systems must include and recognize conceptual objects, socially-defined events, and the rules pertaining to them. This poses new challenges to research on conceptual modeling and poses an ontology of the artificial as a step toward meeting them.

Overall, these articles show how both the research and practice of the design of software intensive systems need to grapple in the future with questions like: 1) what are the necessary and constitutive elements of software design, 2) how design by definition can deal with incompleteness and fluidity, and 3) what are the targets of the design and do different targets matter, and if so, how? They also show that we may need to formulate more distinctive notions of design, design contexts, and design processes than the prevailing simple conceptualizations of design allow.

The Evolution of Design Requirements in the Trajectory of Artificiality: A Research Agenda

Isabelle Reymen and Georges Romme

Eindhoven University of Technology, P.O. Box 513,
5600 MB Eindhoven, The Netherlands
`i.m.m.j.reymen@tue.nl, a.g.l.romme@tue.nl`

Abstract. Managing design requirements of complex socio-technical designs in heterogeneous and rapidly-changing environments demands new approaches. In this chapter we use the framework described by Krippendorff [1] to describe the evolution of requirements thinking and subsequently develop a research agenda. Krippendorff's trajectory of artificiality shows an increasing dematerialization and human-centeredness of artifacts. He distinguishes six kinds of artifacts, namely material products; goods, services, and identities; interfaces; multi-user systems and networks; projects; and finally, discourses. Based on a review of the design literature, involving two major design journals, we find that the design of socio-technical systems currently tends to be situated on the level of multi-user systems and networks. Projects and discourses hardly get any attention in requirements thinking. We therefore develop an agenda for future research directed toward advancing requirements thinking at the level of projects and discourses as artifacts of design.

Keywords: Socio-technical system, project design, discourse design, role of requirements, meta-requirement.

1 Introduction
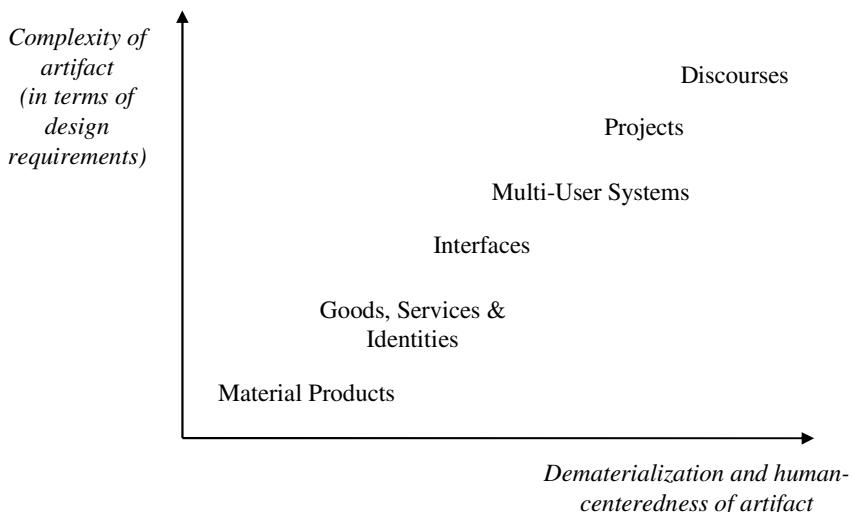
The field of requirements identification, capture, verification, and management for complex socio-technical designs is in need of a vision and agenda for future research. Managing design requirements in heterogeneous and rapidly-changing environments demands new approaches. In this chapter, we contribute to the design requirements debate by describing the evolution of requirements thinking, using the framework described by Krippendorff [1], and deriving a research agenda for requirements research. The argument starts by introducing Krippendorff's framework. Subsequently, the design requirements literature is reviewed and assessed by means of this framework. Based on the evolution observed in this literature, we then discuss some implications for the design of complex socio-technical systems. Finally, a future research agenda is outlined.

2 The Trajectory of Artificiality

Krippendorff [1] describes a trajectory of artificiality, involving a cumulative progression of six major kinds of artifacts, each adding to what designers can do. According t

Krippendorff, this trajectory is not intended to "describe irreversible steps but phases of extending design considerations to essentially new kinds of artifacts, each building up and rearticulating the preceding kinds and adding new design criteria, thus generating a history in progress" (p. 6, [1]). The six kinds of artifacts are material products; goods, services, and identities; interfaces; multi-user systems and networks; projects; and finally, discourses. Figure 1 provides an overview, suggesting that artifacts become increasingly complex (in terms of design requirements) along this trajectory. We now turn to each of these six kinds of artifacts.

   Products are seen here as the material artifacts that producers produce. Designing products thus implies adopting manufacturers' (design) requirements: for example, cost price, utility, functionality, and universal aesthetics [1]. The primary value of products therefore arises from their material functionality from a manufacturer's point of view. The value of products for those who buy and use them plays a secondary role. The emphasis on material products prevailed in the supply economy of the 1950s and 1960s in the USA, Europe and elsewhere, in which resources were scarce and for almost all products demand was structurally higher than what the manufacturing industry could supply.

Fig. 1. The trajectory of artifacts in design (adapted from [1])

   Goods, services, and identities are artifacts that are fundamentally different from products. They are distinguished from products by Krippendorff [1] as follows. Goods are fabricated to be traded and sold, not merely used; their primary function is their role in the marketplace and serve, at best, as sales arguments. Moreover, services need to be designed to be recognizable and trustworthy so that customers return and develop loyalty to the service provider. Identities, whether they are encoded in logos, brand names, or corporate images, are deliberately crafted to create various kinds of commitments. Goods, services, and identities are thus only products in a metaphorical

sense, as Krippendorff suggests. In designing these artifacts, designers are concerned with marketability, symbolic diversity, folk and local aesthetics [1].

Following Krippendorff, the next type of artifact is one that mediates between complex technological devices and their users: *human-machine interfaces*. Examples of human-machine interfaces are the cockpit screen of an airplane pilot or the interface of a particular computer game. The design of interfaces shifts the attention of designers from a concern for the internal configuration and appearance of technology to what mediates between single users and technology – that is, between how individual users make sense of the technology and how designers try to create a technology that supports, enhances or changes users' practices. Important concerns here are natural interactivity, understandability, reconfigurability, and adaptability [1].

Multi-user systems and networks facilitate the coordination of many human activities across space and time, for example sign systems, information systems, or communication networks [1]. An example is an airport check-in system or an e-commerce system for ordering and paying books (e.g., Amazon). Unlike in the design of interfaces, the design of multi-user systems deals with the information that multiple participants can (simultaneously) receive and send through such systems. In the case of multi-user systems and networks, design requirements may involve for example informativeness, connectivity, and accessibility [1]. Whereas human-machine interfaces are artifacts at the level of interaction between individual users and their devices, multi-user systems need to be capable to deal with, and coordinate, a large number of different users with a variety of preferences and needs.

Projects are one of the most complex artifacts one can imagine – complex in the sense of design requirements (cf. Figure 1). A project typically arises around particular desires to change something – a purpose or objective, however vague it may be at the outset – to develop a technology, for example, and create something that is useful for people other than those directly involved [1]. Following Krippendorff, projects have several characteristics. As artifacts, projects are realized in particular communicative practices among participants. Moreover, projects are designed to achieve cooperation among people without losing sight of what the project is about. As artifacts, projects are mainly processed in language, in narratives of what has to change, needs to be done, how, by whom, and at which time. A project can rarely be designed and executed single-mindedly by an individual. The very openness to details energizes a project and motivates its contributors to perform and deliver. Projects are socially viable organizations, temporary in nature but lasting long enough to leave something behind. Important design requirements involve considerations such as social viability, directionability, and commitment [1].

Finally, *discourses* are organized ways of talking, writing, and acting accordingly [1]. There are many distinct discourses: for example, professional, political, religious, and economic discourse. A discourse may be supported by a multi-user system, but its viability does not depend on this system. As an artifact, discourse may be enacted in a project, but does not require a (deliberately crafted) common purpose. Discourses involve communities of people who collaborate in enacting what constitutes their collective identity, thereby creating everything that matters to their sense of community [1]. Design requirement and considerations here involve generativity, rearticulability, and solidarity [1]).

This trajectory of artificiality shows an ongoing dematerialization of artifacts [1]. That is, artifacts become increasingly fluid, indeterminable, immaterial (or virtual), and embedded in language. Products are the most material in nature. Projects and discourses are almost entirely immaterial; they may involve material components, due to other artifacts (e.g., goods and interfaces) playing a role in realizing a project or initiating a discourse (note that the trajectory is cumulative in nature). Along the trajectory, the human-centeredness of artifacts also increases. The least human centered are the products manufactured in a supplier-driven economy (e.g., in 1950s), whereas projects and discourses tend to highly depend and focus on human participation, creativity and values (cf. [1]).

3   Evolution of Design Requirements

The trajectory of artificiality previously discussed is used in this section to assess the literature on design requirements and to describe its evolution. We selected two representative journals in the design field, namely *Design Studies* and *Research in Engineering Design*. In these journals, we searched for all articles that have "requirements", "specifications", "demands", and "criteria" in the abstract, title, or key words. The search period begins with the first volume of each journal (Design Studies 1979; Research in Engineering Design 1989) and ends with the last issue available in 2007. The resulting set of articles involves either studies of requirements for the design of something (an artifact) or methodological papers about requirements elicitation, validation and management for a certain kind of artifact.

We categorized, according to the closed coding approach of Strauss and Corbin [2], each article according to one of the 6 kinds of artifacts as introduced by Krippendorff, namely products; goods, services, and identities; interfaces; multi-user systems/networks; projects; and discourses. This categorization follows from the kind of artifact the requirements-specifications-demands-criteria were related to. The result is shown in Table 1.

Table 1. Categorization according to type of artifact discussed in articles on requirements in two design journals

Artifact	# Articles in Design Studies (n=100)	# Articles in Research in Engineering Design (n=43)
products	47	8
goods, services, and identities	8	0
interfaces	35	29
multi-user systems/networks	7	4
projects	3	2
discourses	0	0
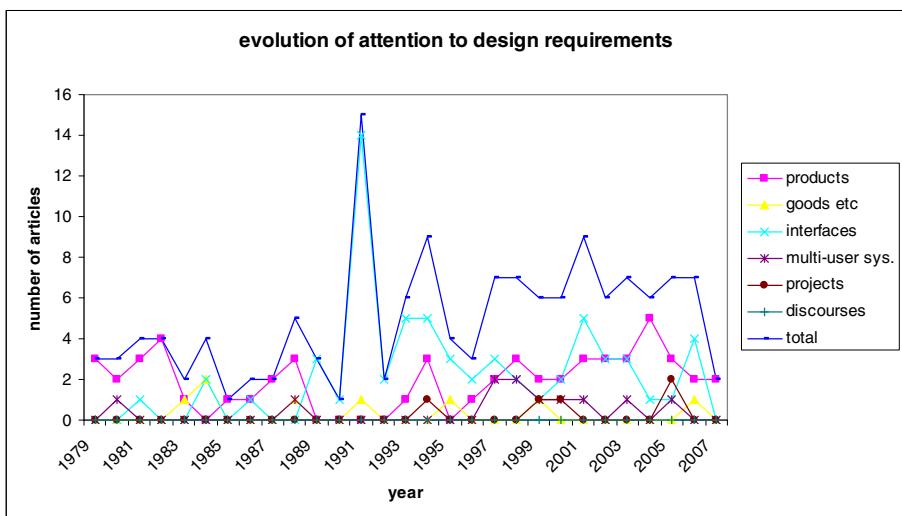
The evolution of attention to design requirements for each of the artifacts in the two design journals is depicted in Figure 2. The results in Table 1 and Figure 2 suggest a prevailing focus on the product and interface level. The attention for artifacts conceptualized as goods, services and identities is rather underdeveloped, as are multi-user systems and networks. Projects hardly get any attention in requirements thinking. The five studies at the level of projects are: Eppinger et al. [3]; Lauche [4]; Pons and Raine [5]; Reid et al. [6]; Smith and Morrow [7]. Finally, there were no articles in any of the two journals addressing discourses as artifacts (see Table 1). These general patterns are similar for both journals, with the exception of the data for goods, services and identities (see Table 1).

To illustrate the kind of requirements defined and used for each type of artifact, we now turn to several representative papers in each category. This results in the following list:

- For products, *functionality* [8] and *just-in-time production* [9] are important requirements.
- For goods, services, and identities, Foque and Lammineur [10] derive *user-oriented and user-friendly, functionality and emotion* as important requirements for service design. Wright [11] argues that for designing the documentation that explains how IT works, there is a requirement for an initial involvement of the technical writers during product development. This requirement thus concerns *links between writer and system designer*. There is a subsequent requirement for an ability to think about the reader's needs (*links between writer and reader*). And there is a final requirement for means of evaluating the usability of the documentation (*links between writer and text*).
- For interfaces, requirements are then, for example, need to be *extensible, reusable, multi-agent,* and *concurrent* [12] and *functional* and *accessible* as well as exhibiting short *interactive response times* [13].
- For multi-user systems and networks, an interesting requirement involves the *distributive characteristics of collaborative product design*. These characteristics involve, for example, extended time; multiple places, cultures, practices, policies and behaviors; multiple languages and tools; interchangeable interaction methods; and usability and adaptability to workers with different levels of education [14]. In addition, the *evolutionary nature of the environment* (e.g., group evolution and learning, supporting variable-term relationships) is defined as an important requirement [14]. For interactive video game design, for example, Jacobs and Ip [15] discuss *multi-user functional requirements* such as graphics, sounds, technical realism, structure and challenge, multiplayer features, and online features.
- For projects, Eppinger et al. [3] derived the following *information transfer requirements* of a technical project (to improve the product development process): 1. Documenting existing procedures for scrutiny of the team, 2. Resequencing and regrouping design tasks to reduce complexity, 3. Sharing engineering data earlier and/or with known confidence, 4. Redefining critical tasks to facilitate overall project flow, 5. Exposing constraints and conflicts due to task interactions, 6. Helping design managers to place emphasis on task coordination, 7. Allowing design iteration to be strategically planned.

Lauche [4] determined the following requirements for job design: control over the design process, availability and clarity of design-relevant information, feedback on results, and management support. Pons and Raine [5] discuss how to deal with requirements when designing projects. They do so in terms of constraints from incomplete and qualitative specifications, using subjective processes. Furthermore, they subsequently negotiate with others to relax constraints, as the design space may be over-constrained. This negotiation involves interaction with others, and adds behavioral factors to the design process. As such, decision-making during design processes needs to be able to accommodate multiple viewpoints, cope with uncertainty of analysis (incompleteness of knowledge), propagate uncertain variables, and accommodate varying degrees of information abstraction.

Other authors, for example Reid et al. [6], see design coordination and team integration as the most important dilemmas for project management by design teams. Reid and co-authors argue that successful project management requires project leaders to continuously steer an acceptable path through these dilemmas. They suggest these problems can be addressed by adopting a flexible, dynamic approach to team coordination in which moment-to-moment demands are met by appropriate management actions.

Fig. 2. Evolution of attention to requirements in the community of design researchers

Given the fact that projects and discourses hardly get any attention in requirements thinking in the selected journals, we also searched other journals. In this respect, three journals that were created recently are explicitly human-centered in their aim and scope. These are *Journal of Design Research* (e.g., the work of Sebastian [16]), *Co-Design* (International Journal of CoCreation in Design and the Arts) (e.g., [17]), and *The Design Journal* (of the European Academy of Design) (e.g., [18]). These journals may actually publish more work on projects and discourses as artifacts than the two

journals we have studied in this paper.[1] This implies that our focus on *Design Studies* and *Research in Engineering Design* may produce a bias in our findings – in particular the relatively low number of studies regarding projects and discourses in Table 1.

4   Design Requirements of Complex Socio-technical Systems

The trajectory of artificiality involves an evolution in the kind of artifacts being designed. In this respect, requirements elicitation, validation and management should coevolve in parallel. This means that for each type of artifact, the corresponding requirement (management) techniques must be considered. As such, requirements thinking needs to be congruent with the kind of the artifact being designed.

Given the results of the literature review and interpretation in the previous section, requirements thinking has recently developed up to the level of multi-user systems (cf. Figure 1). In this respect, the design of socio-technical systems currently tends to be largely framed in terms of multi-user systems and networks (e.g., [19],[20]). However, to create effective socio-technical systems in heterogeneous and rapidly changing environments, their design requirements must also be viewed from a project and discourse perspective.

More particularly, software systems can be understood and designed as advanced socio-technical systems. In this respect, the trajectory of artificiality suggests that software (systems) can be framed as an artifact at each level of this trajectory. This is evident for software systems as material products (e.g., software delivered on cd-rom), services (e.g., a 24x7 service desk), interfaces (e.g., a knowledge management interface with search functions) and multi-user systems and networks (e.g., an airport check in system). An example of a software-related project is the introduction of the Capability Maturity Model in a company (e.g., [21]). An example of a software-enabled discourse is Wikipedia. Software can also be supportive for designing projects and discourses. Moreover, people may think they are designing software, although in fact they are (implicitly) designing a project or discourse.[2]

Evidently, the role of design requirements changes along the trajectory of artificiality. In this respect, in moving from left to right on the trajectory outlined in figure 1, more attention needs to be given to social and semantic instead of technical and material aspects. Moreover, along the trajectory of artificiality, system boundaries will become more diffuse and complex and system design becomes more context dependent. Additionally, moving from products to discourses as artifacts, the process of defining and managing requirements becomes more participative, dynamic and flexible. Each type of artifact can be designed in a more or less participative manner. Stakeholder involvement in the design process can be positioned on a continuous scale between the designer as an expert, i.e., with minimal stakeholder feedback, and stakeholder-driven design, i.e., the designer and stakeholders co-designing. The desired level of stakeholder interaction of the artifact itself may however differ among the different types of artifacts and result in different requirements. Described from an

---

[1] The low number of published volumes of these journals severely constrains opportunities to study papers on design requirements and compare the findings with the two design journals used in this paper.

[2] This paragraph is based on comments of Kalle Lyytinen on a draft version of the paper.

interaction perspective, products and identities involve communicating a message about functionality, quality and other aspects to the stakeholders in their environment; interfaces support communication between stakeholders (users) and technology; multi-user systems support interactions between multiple stakeholders/users; projects organize how a team can perform a task collaboratively; discourses consist mainly of communication between participants. Designing these artifacts means (in the trajectory of artificiality) increasingly designing support for communication and interaction between stakeholders. Both requirements about the desired level of stakeholder involvement in the design process and those about the desired level of stakeholder interaction of the artifact are important and should be defined and managed. Especially when many stakeholders are involved, conflicting interests need to be discussed and managed.

For products, identities, interfaces and multi-user systems, requirements can be communicated – in two directions – between designers and stakeholders during the artifact design process. For projects and discourses, this can also take place during the actual "use" of the designed artifact, i.e., during the course of the project or discourse. For these types of artifacts, it is less clear when the design of the artifact is finished, i.e., when the design is "frozen" and produced, like in a product or a software system. Projects and discourses are more open-ended and are not produced at a certain moment; they can continuously be redesigned, importantly also by the stakeholders during the use of the artifact. This means also that their requirements are continuously adapted and should thus be more dynamic and flexible. This influences how the requirements are represented and communicated. In some cases, IT systems can be used to support the communication processes.

5   Future Research Agenda: Designing Projects and Discourses

The previous two sections identified several emerging patterns in the evolution of requirements thinking. We argued that requirements thinking needs to be congruent with the kind of artifact being designed. In this respect, in addressing complex sociotechnical systems, designers and design researchers will increasingly have to explore design requirements at the level of projects and discourses as the artifacts being designed and created. We suggest the following questions and challenges for future research.

5.1   What Are Design Requirements for Projects and How Should They Be Managed?

Projects as the artifacts of design constitute an enormous challenge for requirements researchers. An interesting starting point can be found in agile developments and methods like SCRUM [22]. The agile development approach sets up the design process in a pragmatic way with team-based, bottom-up design processes. The agile perspective on organizing design projects might be useful for informing the design of other projects as well. One particular question here is whether the requirements formulated for agile projects can be codified, and to what kind of projects they can (not) be applied.

5.2 What Are Design Requirements for Designing a (viable) Discourse and How Should These Be Managed?

An example of the deliberate attempt to design a discourse is branding "Poland" [23]. Another interesting example is the way the Bush government in the USA has been redesigning the political discourse toward the key idea: 'if you criticize the Bush government, you criticize our soldiers in Iraq.' As previously mentioned, Wikipedia is another example, closer to the design of socio-technical systems. Garud et al. [24], studied the design of the Wikipedia online encyclopedia. They highlight that deliberate incompleteness is part of a pragmatic approach to design. This suggests that design requirements for designing discourse should thus be able to deal with, and incorporate, incompleteness and fluidity.

Krippendorff (pp. 10-11, [1]) argues that projects are "realized in particular communicative practices among participants" while discourses "reside in communities of people who collaborate in enacting what constitutes their community". Therefore, to start building a body of knowledge about design requirements for projects and discourses, the literature on communities of practice (e.g., [25]) provides an important point of departure. In this volume, Fischer distinguishes between communities of practice and communities of interest [26]. This suggests future research needs to address how the nature of the community affects its discourse, and vice versa.

Moreover, designing socio-technical systems at the project and discourse level may be more about the design of the ecosystem than of the artifact itself. For example, an information system may be intended to shape an organizational discourse (e.g., around knowledge, competences, and client needs) as an ecosystem. The design requirements should then largely be defined on the level of the ecosystem. The idea of ecosystem and ecology corresponds to the "meta-design" concept introduced by Fischer et al. [27] and Fischer and Giaccardi [28]. The idea here is to use, as a designer, your own creativity to create socio-technical environments in which other people can be creative. The only thing you do as a designer is provide a context where users provide the content (e.g., web 2.0, open source, Wikipedia). In these examples, designing occurs simultaneously to using. Interesting in this regard is also the SER model of Fischer and Ostwald [29], which describes a cycle of seeding, evolutionary growth, and reseeding. Requirements should thus be defined for open, adaptive, and adaptable systems, which make possible that the systems can emerge and continuously evolve in interaction between stakeholders. To some extent, thus, requirements for shaping a language are needed.

5.3 What Is the Role of Requirements in Designing Projects and Discourses?

The human-centered as well as participative nature of projects and discourses evidently affects the role of requirements in the design process, but how? Human-centeredness implies that requirements are not only of a 'technical' nature, but also include communication specifications, co-ordination specifications, and so forth. The participative nature of projects and discourses implies that requirements (but also the solutions and outcomes) are not only determined by the designer and user, but by all stakeholders involved. This means that requirements elicitation, validation and management fundamentally changes, because it will be done along the way by a continually changing population of participants. Given the co-evolution of problem and

solution in socio-technical design projects [30], requirements should also co-evolve with the solution. This means they cannot be fixed at the beginning of the process and may (need to) change rapidly. In general, requirements will not loose their importance if they are able to adapt and respond as an open, evolving system. If requirements are fixed at the outset and cannot change, they will become obsolete and irrelevant to how the project or discourse evolves and matters to the people engaging in it.

5.4 How Does Requirements Management Differ for Complex Socio-technical Systems, Framed as Different Types of Artifacts?

This is a bold question that can be specified in a number of directions. First, we need to understand how and why software designers (are able to) think through the levels of the trajectory of artificiality and determine which levels are relevant for particular contexts. Second, future research needs to explore how requirements differ for these different types of artifacts, for example in terms of scale, flexibility, dynamic behavior, and growth pattern. Finally, the question arises as to whether there are differences in the manner in which requirements emerge for a particular type of artifacts.

5.5 Can Meta-requirements of the Requirements Linked to Each Type of Artifact Be Developed?

At the product level, for example, the field of systems engineering has already defined meta-requirements (e.g., [31]). At the project and discourse level, possible 'meta-requirements' can be derived from the discussion above: for example, the ability to deal with incompleteness and fluidity; the focus on ecology; and an emphasis on human-centered as well as participative values. To avoid confusion and build a straightforward requirements 'language', these meta-categories are perhaps better labeled demands (for requirements).

Acknowledgement

References

1. Krippendorff, K.: The Semantic Turn: A New Foundation for Design. Taylor & Francis, New York (2006)
2. Strauss, A., Corbin, J.: Basics of qualitative research: Grounded theory procedures and techniques. Sage Publications, London (1990)
3. Eppinger, S., Whitney, D., Smith, R., Gebala, D.: A model-based method for organizing tasks in product development. Research in Engineering Design 6, 1–13 (1994)
4. Lauche, K.: Job design for good design practice. Design Studies 26, 191–213 (2005)

5. Pons, D., Raine, J.: Design mechanisms and constraints. Research in Engineering Design 16, 73–85 (2005)
6. Reid, F., Culverhouse, P., Jagodzinski, A., Parsons, R., Burningham, C.: The management of electronics engineering design teams: linking tactics to changing conditions. Design Studies 21, 75–97 (2000)
7. Smith, R., Morrow, J.: Product development process modeling. Design Studies 20, 237–261 (1999)
8. Owen, C.: Evaluation of complex systems. Design Studies 28, 73–101 (2007)
9. Whitney, D.: Nippondenso Co. Ltd.: A case study of strategic product design. Research in Engineering Design 5, 1–20 (1993)
10. Foque, R., Lammineur, M.: Designing for patients: a strategy for introducing human scale in hospital design. Design Studies 16, 29–49 (1995)
11. Wright, P.: Designing the documentation that explains how IT works. Design Studies 7, 73–78 (1984)
12. Cartwright, A.: Interactive prototyping—a challenge for computer based design. Research in Engineering Design 9, 10–19 (1997)
13. van Elsas, P., Vergeest, J.: New functionality for computer-aided conceptual design: the displacement feature. Design Studies 19, 81–102 (1998)
14. Reich, Y., Konda, S., Subrahmanian, E., Cunningham, D., Dutoit, A., Patrick, R., Thomas, M., Westerberg, A.: Building Agility for Developing Agile Design Information Systems. Research in Engineering Design 11, 67–83 (1999)
15. Jacobs, G., Ip, B.: Establishing user requirements: incorporating gamer preferences into interactive games design. Design Studies 26, 243–255 (2005)
16. Sebastian, R., Arch, B.: Multi-architect design Collaboration on Integrated Urban Complex Development in the Netherlands. J. of Design Research 3 (2003)
17. Marshall, J., Pengelly, J.: Computer technologies and transdisciplinary discourse: critical drivers for hybrid design practice? CoDesign 2, 109–122 (2006)
18. Chung, W., Wang, C.: Knowledge-based design and decisions: Accommodating multi-disciplined product development teams. The Design Journal 7 (2004)
19. Mumford, E.: The story of socio-technical design: reflections on its successes, failures and potential. Information Systems Journal 16, 317–342 (2006)
20. de Moor, A., Weigand, H.: Formalizing the evolution of virtual communities. Information Systems 32, 223–247 (2007)
21. Paulk, M., Weber, C., Garcia, S., Chrissis, M., Bush, M.: Key Practices of the Capability Maturity Model, Software Engineering Institute. CMU/SEI-93-TR-25 (1993)
22. Schwaber, K., Beedle, M.: Agile Software Development with Scrum. Prentice Hall PTR, Upper Saddle River (2001)
23. Olins, W.: National branding in Europe. Business at Oxford 7, 4–6 (2005)
24. Garud, R., Jain, S., Tuertscher, P.: Incomplete by Design and Designing for Incompleteness. Organization Studies 29, 351 (2008)
25. Lave, J., Wenger, E.: Situated Learning: Legitimate Peripheral Participation. Cambridge University Press, Cambridge (1991)
26. Fischer, G.: Communities of Interest: Learning through the Interaction of Multiple Knowledge Systems. In: 24th Annual Information Systems Research Seminar In Scandinavia (IRIS 24), Ulvik, Norway, pp. 1–14 (2001)
27. Fischer, G., Giaccardi, E., Ye, Y., Sutcliffe, A., Mehandjiev, N.: Meta-design: a manifesto for end-user development. Communications of the ACM 47, 33–37 (2004)

28. Fischer, G., Giaccardi, E.: Meta-Design: A Framework for the Future of End User Development. In: Lieberman, H., Paternò, F., Wulf, V. (eds.) End User Development: Empowering People to Flexibly Employ Advanced Information and Comm., pp. 427–457. Kluwer Academic Publishers, Dordrecht (2006)
29. Fischer, G., Ostwald, J.: Seeding, Evolutionary Growth, and Reseeding: Enriching Participatory Design with Informed Participation. In: Proceedings of the Participatory Design Conference (PDC 2002), Malmö University, Sweden, pp. 135–143 (2002)
30. Dorst, K., Cross, N.: Creativity in the design process: Co-evolution of problem–solution. Design Studies 22, 425–437 (2001)
31. IEEE Standard: Guide for developing System Requirements Specifications. IEEE, New York (1998)

A Proposal for a Formal Definition of the Design Concept

Paul Ralph and Yair Wand

Sauder School of Business
University of British Columbia
Canada
paulralph@gmail.com, yair.wand@ubc.ca

Abstract. A clear and unambiguous definition of the design concept would be useful for developing a cumulative tradition for research on design. In this article we suggest a formal definition for the concept *design* and propose a conceptual model linking concepts related to design projects. The definition of design incorporates seven elements: agent, object, environment, goals, primitives, requirements and constraints. The design project conceptual model is based on the view that projects are temporal trajectories of work systems that include human agents who work to design systems for stakeholders, and use resources and tools to accomplish this task. We demonstrate how these two suggestions can be useful by showing that 1) the definition of design can be used to classify design knowledge and 2) the conceptual model can be used to classify design approaches.

Keywords: design, information systems design, software design project, requirements, goals, science of design.

1 Introduction

There have been several calls for addressing design as an object of research. Freeman and Hart [1] call for a comprehensive, systematic research effort in the science of design: "We need an intellectually rigorous, formalized, and teachable body of knowledge about the principles underlying software-intensive systems and the processes used to create them," (p. 20). Simon [2] calls for development of a "theory of design" and gives some suggestions as to its contents (p.134). Yet, surprisingly, it seems no generally-accepted and precise definition of design as a concept is available.[1]

 A clear understanding of what design means is important from three perspectives. From an instructional perspective, it seems obvious that any designer's education ought to include providing a clear notion of what design is. Furthermore, better understanding what design is will inform what knowledge such education could include.

 From a research perspective, in any theoretical or empirical work in which design is a construct, a clear definition will help ensure construct validity. Furthermore, a clear understanding of the meaning of design will facilitate developing measures of

---

[1] As an anecdotal note – we have asked colleagues in several conferences to suggest a definition for "design" (in the software and IS context) and often the responses indicated IS academics did not have a well-defined notion of the concept.

design-related constructs, such as design project success. Moreover, building a cumulative tradition of design research can benefit from having a well-defined, the alternative being different theories define design differently, or not defining it explicitly.

From a (software design) practitioner's perspective, a clear definition of design can help organize, share and reuse design knowledge. Such sharing can enhance software project success and software development productivity. Furthermore, understanding the elements of design would be useful in determining the issues and information that need to be considered in the process of design and in planning this process.

Given the potential value of a clear definition of design, our objective here is to suggest such a definition. We first seek to answer the question: what are the important elements of design as a phenomenon? We then seek to situate design in a network of related concepts.

We begin our discussion by making a distinction between *the science of design* and *the design science research paradigm* as elucidated by Hevner et al. [3]. In their view, design science research "builds and evaluates constructs, models, methods and instantiations" with "design intent" ([4], p. 256). In contrast, Freeman and Hart [1] call on the community to theorize and justify theories *about* design – what March and Smith [4] call "natural science intent" (p. 256). *Design science* is a research paradigm, like experimentalism. *Science of design* is a field of inquiry, like psychology. Here we seek to primarily address issues related to the science of design.

The paper is organized as follows. First, we synthesize a definition of design by applying concepts and suggestions in existing literature (§2). We then evaluate the proposed definition in Section 3. Section 4 situates our view of design in a conceptual model of software design projects. In Section 5, we demonstrate how the proposed definition of design can be applied to indexing design knowledge for reuse and by using the conceptual model of software design projects to classify design approaches. Finally, we discuss the implications of our definition of design for current themes in software design and requirements research (§6).

2   Proposing a Formal Definition of Design

2.1   Design in the Literature

We have conducted a review of existing definitions of the concept "design" in the literature. A list of definition we examined is provided in the *Appendix* (Table 9). We analyzed the definitions in three ways: first, we identified concepts that appeared common to several definitions (Table 1). We then analyzed each definition as to whether it appeared to have errors of omission or inclusion by testing them with respect to a set of examples. We have found that all definitions included errors of either kind or both. The detailed analysis is provided also in Table YY (Appendix). Finally, we have identified four main areas of disagreement among the definitions.

While most of the areas of agreement seem reasonable, a few appear problematic. First, some definitions confuse design with *good* design, adding desirability criteria to the definition, as evidenced by words like "optimally" [5] and "optimizing" [6]. Designs might be suboptimal, but we still call them designs. Second, organizing does not necessarily constitute design, for example, when someone returns books to their

Table 1. Frequency of Common Concepts in Analyzed Definitions

Concept	Frequency
Design as a *process*	11
Design as creation	11
Design as *planning*	7
Design as a *physical* activity (or as including implementation)	7
System (as the object of the design)	7
Design as being *deliberate*, or having a *purpose, goal* or *objective*	7
Design as an *activity*, or a collection of activities	7
Design as occurring in an environment (or domain/situation/context)	7
Artifact, as the object of the design	5
Needs or *requirements*	5
Design as a *human* phenomenon	5
Design as *organizing*	4
Parts, components or elements	4
Constraints or *limitations*	3
Process (as the object of design)	2
Design as *creative*	2
Optimizing	2
Design as a *mental* activity	2
Resources	2

proper shelves in a library, one is organizing the books into a pre-designed arrangement rather than actively performing a design task. Third, four definitions state or imply that design is strictly a human phenomenon. However, machines can also design objects (e.g., Bradel and Stewart [7] report on the design of processors using genetic algorithms).[2] Fourth, while many designers are surely creative, not all design need involve creativity. For example, design might involve relatively minor modifications to a previously created design.

Finally, we mention the four areas of disagreement we have identified. First, different objects of design arise: *system*, *artifact* and *process*. Second, disagreement exists concerning the scope of design: where or when a design begins and ends. Third, some definitions indicate that design is a physical activity, others a mental activity. Fourth, some disagreement concerns the outcome of design: is it a plan, an artifact, or a solution?

2.2   Suggesting a Definition of Design

In this section, we develop our proposed definition of design. First, Eekels [8] differentiates between the subject of the design and the object of design. The subject of the design is the (often human) **agent** that manifests the design. The **design object** is the thing being designed. Design outcomes such as an artifact, a system or a process that appear in some existing definitions are encompassed here by the more general term, design object.[3]

---

[2] Some research indicates this might also be the case for animals (see [9] and [10]).

[3] Note: often the object is called an artifact, when designed by humans. The more general term object allows (in principle) for non-human agents such as animals and computers.

Some definitions mention parts, components or elements of which the design object is, or is to be, composed. Obviously, all artificial physical things are made from other things. These other things might be given, or also are composed of components. We term the lowest level of components ***primitives***. Similarly, but perhaps less obviously, if we assume that atomic conceptual things, such as a single thought or idea, are not designed (but discovered or just *are available*), then all conceptual things that are designed are made from other conceptual things. Therefore, all design involves components, or primitives, which are, or can be, assembled or transformed to create the design object.[4] March and Smith [4] note that "Technology includes...materials, and sources of power" (p. 252). Materials and sources of power would be included in the set of primitives.

The outcome of a design effort is not necessarily the design object itself, but may be a plan for its construction, as pointed out by the definitions that characterize design as planning rather than building. The common factor here is that the agent specifies properties of the design object: sometimes as a symbolic representation, as in an architectural blueprint, sometimes as a mental representation, as in the picture in the painter's mind, and sometimes as the artifact itself, as in a hand-carved boomerang. We call the specified properties of the design object a ***specification***. More specifically, *a specification is a detailed description of a design object's structural properties*, namely, what primitives are assembled or modified and, if more than one component is used, how primitives are linked together to make the artifact.[5]

Practically speaking, a specifications document might include desired behaviors as well as structural properties. From the perspective of this paper, these desired behaviors are requirements – they are not strictly part of the specifications. The object's behavior *emerges* from the behavior of the individual components and their interactions. By behavior we mean the way the object responds to a given set of stimuli from its environment (including agents who interact with the artifact).

The specification may be purely mental, provided in a symbolic representation, presented as a physical model, or even manifested as the object itself.

Churchman [11] points out that "Design belongs to the category of behavior called teleological, i.e., "goal seeking" behavior," (p. 5). Many of the definitions we surveyed also included the concepts of goal, purpose or objective. It is possible the goal is not explicit or not well-defined. However, a design effort is always intentional. For example, a social networking web application can be designed, without having an articulated explicit goal, based only on the vague idea that it would be useful (and fun) to have an online space where people could connect. We would still say the web application was designed. On the other hand, accidental or unintentional discoveries are not really designed. Thus, ***goals*** are inherent to design insofar as a designer must

---

[4] What the set of available primitives is can be a relative issue. A designer might be given a set of components, or component types, where each might be in turn composed from lower level components. We consider primitives the set of component-types available to the designer, independent of whether they are natural, or the outcome of previous design. Furthermore, even if the components are not yet available, a designer might proceed assuming they will be available. The assumptions made about these components will become requirements for their design.

[5] This notion of specification agrees with that of Bourque and Dupuis ([14], p. 1-3), that design is the *activity that produces "a description of the software's internal structure"*.

have intentionality. However, this should not be interpreted as a requirement that a design goal is or can be explicitly specified and articulated.

Many definitions characterize the design process as occurring within an environment, domain, situation or context. Design involves *two different environments*: the environment of the design object, and the environment of the design agent. As pointed out by Alexander [12] "every design problem begins with an effort to achieve fitness between two entities: the form in question and its context." Clearly, the design process or activity also occurs within some **environment**, even if that environment is difficult to characterize. March and Smith [4] mention the "organizational setting" (p. 252) and Hevner et al. [3] refer to "organizational context" (p. 77). For instance, the software created by a developer is intended to operate in a different environment than the developer. For the environment of the artifact the qualifier "organizational" is not always valid because, for some design objects, the environment does not have to be an organization (e.g. the environment of a pacemaker is a human body).

Many definitions also mention needs or requirements and limitations or constraints. The issue of requirements requires a clarification. If we interpret requirements strictly as a *formal* requirements document or as a set of mathematically expressible functions (as in [13]) the system is to perform, then requirements are not absolutely necessary. The primitive hunter who fashions a spear from a branch specified the spear's properties by creating it – without an explicit reference to formal requirements (let alone mathematically definable functions). However, in the sense that every designer expects or desires of the design object to possess certain properties or exhibit certain behaviors, **requirements** are inherent to design. Requirements are a major construct in requirements engineering and software design (see, for example [15] and [16]).

Similarly, all design must involve **constraints**. Even if the design agent had infinite time and resources, physical design is still constrained by the laws of physics, virtual design by the speed and memory of the computational environment, and conceptual design by the mental faculties of the design agent. Constraints are a major construct in engineering design (see [2] and [17]). However, we note that, as for goals and requirements it is possible constraints are not stated or perceived explicitly.
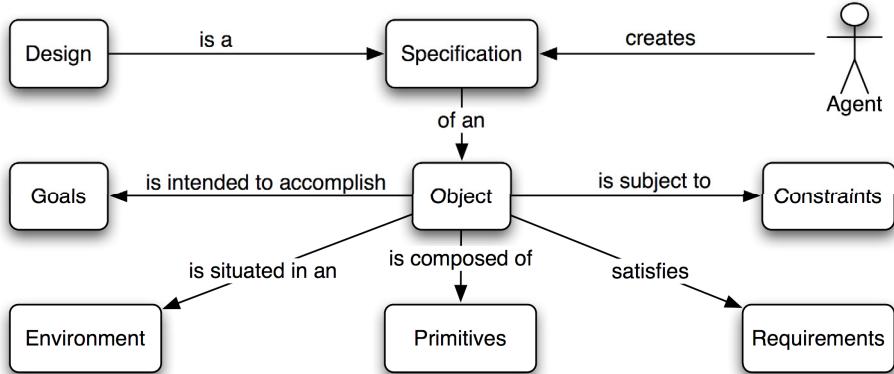
The above analysis leads to the following suggestion for the definition of design (modeled in Figure 1). Table 2 further describes each concept in the definition.

Considering design as a process (depicted in Figure 2), the outcome is the specification of the design object. The goals, environment, primitives, requirements and constraints are, in principle, the inputs to the design process; however, often knowledge of these may emerge or change during the process. Nevertheless, the design process must begin with some notion of the object's intended environment, the type of object to design and some initial intentions. By initial intentions, we simply mean that design cannot be accidental – the design agent must have intentionality. Finally, if the type of design object changes significantly (e.g., from a software system to a policy manual), the existing design effort is no longer meaningful and a new design effort begins. The possibility of changing information is related to the possibility that the design process involves exploration. It also implies that the design might evolve as more information is acquired.

Design

(noun) a *specification* of an *object*, manifested by some *agent*, intended to accomplish *goals*, in a particular *environment*, using a set of *primitive components*, satisfying a set of *requirement*s, subject to some *constraints*;
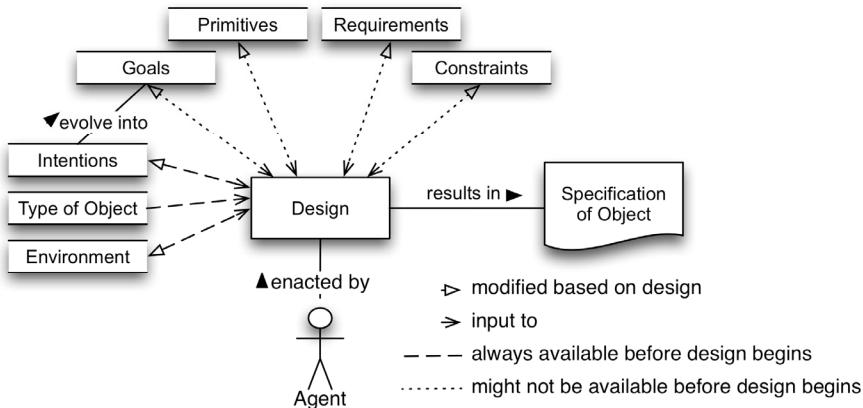
*(verb, transitive)* to create a design, in an environment (where the designer operates)

Fig. 1. Conceptual Model of Design (as a noun)

Table 2. Definitions of Design Concepts

Concept	Meaning
Design Specification	A specification is a detailed description of an object in terms of its structure, namely the components used (out of the set of possible *types* of primitives) and their connections.
Design Object	The design object is the entity (or class of entities) being designed. Note, this entity does not need to be a physical object.
Design Agent	The design agent is the entity or group of entities that specifies the structural properties of the design object.
Environment	The object environment is the context or scenario in which the object is intended to exist or operate (used for defining design as the specification of an object). The agent environment is the context or scenario in which the design agent creates the design (used for defining design as a process).
Goals	Goals are what the design object should achieve; goals are optative (i.e. indicating a wish) statements that may exist at varying levels of abstraction [18]. Since the designed object exists and/or operates in an environment, goals are related to the impact of the artifact on its environment.
Primitives	Primitives are the set of elements from which the design object may be composed (usually defined in terms of *types* of components assumed to be available).
Requirements	A requirement is a structural or behavioral property that a design object must possess. A structural property is a quality the object must posses regardless of environmental conditions or stimuli. A behavioral requirement is a required response to a *given* set of environmental conditions or stimuli. This response defines the changes that might happen in the object or the impact of these changes on its environment.
Constraints	A constraint is a structural or behavioral restriction on the design object, where "structural" and "behavioral" have the same meaning as for requirements.

Fig. 2. Context-level Conceptual Model of Design (as a Verb)

2.3 What Can Be Designed and Examples of Design Elements

"What can be designed?" is a difficult ontological question, one we are not sure we can answer completely. However, we have identified at least six classes of design object:

- **physical artifacts,** both simple, such as boomerangs (single-component), and composite, such as houses (made of many *types* of components)
- **processes**, such as business workflows
- **symbolic systems**, such as programming languages
- **symbolic scripts**, such as essays, graphic models, and software (which, in turn, prescribe the behavior of other artifacts, i.e. computers)
- **laws, rules and policies**, such as a criminal code
- **human activity systems**, such as software development projects, committees, schools, hospitals, and artistic productions (e.g. operas)

Clearly, the nature of a specification depends on the class of design object since the structure and components of; for example, a law would be very different from those of a physical object.[6] For simple artifacts, such as a one-piece racket or a metal blade, the specification would include structural properties such as shape, size, weight and material. For a composite physical artifact, such as a desk, the specification would include the primitive components and how they are connected. Since a process is 'a set of partially ordered activities aimed at reaching a goal' [19], a specification of a process might identify the activities and their order (although other approaches are possible – e.g. using, Petri Nets [20] or states and events [21]). For a symbolic sys-tem, the specification might include syntax, denotational semantics and (for a spoken language) pragmatics. A symbolic script can be specified by symbols and their

---

[6] It is of interest to see how some of the concepts can be applied to non-physical artifacts such as a law. Although a law clearly is a designed (albeit conceptual) artifact, the notion of a "be-havior" of a law might not be clear. One possibility would be the conditions under which it is invoked. Likewise, constraints with respect to laws can be a constitution or cultural values that limit the types of laws that can be enacted.

arrangement. A policy or law can be specified in some (possibly formal) language. The specification of a human activity system might include the various roles and tasks and their relationships and interactions.

Furthermore, all of the elements from the definition of design might vary depending on the object type. Table 3 provides examples of each design element for each type of design object.

2.4   Scope of Design

According to the perspective on design expressed in this paper, design (as a verb) is the act of specifying the structural properties of an object, either in a plan, or in the object itself. Because design is an activity, rather than a phase of some process, it may not have a discernable end point. Rather, it begins when the design agent begins specifying the properties of the object, and stops when the agent stops. Design may begin again if an agent (perhaps a user) changes structural properties of the specification or design object at a later time. This defines the scope of the design activity.

Our definition does not specify the process by which design occurs. Thus, how one interprets this scope of activities in the design process depends on the situation. If a designer encounters a problem and immediately begins forming ideas about a design object to solve the problem, design has begun with problem identification. If requirements are gathered in reaction to the design activity, design includes requirements gathering. In contrast, if a designer is given a full set of requirements upfront, or gathers requirements before conceptualizing a design object, requirements gathering is not part of design. Similarly, if the construction agent refines the specification (a possible occurrence in software development), construction is part of design, but if the designer creates a complete specification on paper that the construction agent follows, construction is not part of design. Any activity, including during testing and maintenance, that involves modifying, or occurs within an effort to modify, the specification is part of design. Therefore, design practice may not map cleanly or reliably into the phases of a particular process, such as the waterfall model [22].

This distinction has particular bearing for software design, where a significant debate over the scope of design exists. On the narrow-scope side, Bourque and Dupuis [14], for example, define design as:

> *the software engineering life cycle activity in which software requirements are analyzed in order to produce a description of the software's internal structure that will serve as the basis for its construction*, (p. 3-1).

On the broad-scope side, Freeman and Hart [1], for example, argue that:

> *Design encompasses all the activities involved in conceptualizing, framing, implementing, commissioning, and ultimately modifying complex systems—not just the activity following requirements specification and before programming, as it might be translated from a stylized software engineering process*, (p. 20).

One way of interpreting this debate is as follows. Proponents of a narrow scope of the design process posit that all inputs to design (goals, environment, primitives,

Table 3. Examples of Design Elements

Object Type	Process	Symbolic system	Law/policy	Human activity system	Physical artifact	Symbolic Script
Object	loan approval	a special purpose programming language	criminal code	a university course	office building	a software system
Agent	loan officer	person or persons who create the language	legal experts and lawmakers	instructor	Architect	programmer
Goals	accurately estimate risk level of loan	provide a means of expressing software instructions	provide a legal framework for dealing with crimes	facilitate learning and development of students in a given area	provide office space for a business	support management of customer information
Environment	bank administrative system	computing environment on which code will execute	national legal and constitutional system	university (with all resources available)	business district of a given city	personal computers and a specific operating systems
Requirements	provide a decision with justification; generate audit trail for decision process	be easily readable, minimize coder effort, fit certain applications	define crimes and punishments clearly; be unambiguous	learning objectives	include open floor plan of offices, be energy efficient	maintain customer information, identify customers with certain characteristics
Primitives	various actions that need to be taken, e.g., assessing the value of a collateral	the c programming language instructions	English words as used in legal documents	various common teaching actions (presentations, laboratory sessions, tests)	building materials, interior decoration materials	the instructions in the symbolic system (programming language)
Constraints	bank approval rules and risk policies (e.g. debt-service ratio allowed)	cannot violate some programming languages related standards	must not violate the country's constitution and international laws	prior knowledge students have, number of class and laboratory hours available	comply with building code, cost less than a given budget	must be able to run on a given hardware configuration with maximum delay X.

requirements and constraints) are fully defined before any property of the object has been decided. Furthermore, the design phase results in a full specification of all relevant object properties before coding begins. In contrast, proponents of a broad scope of design recognize that properties of the object are often defined during requirements elicitation or, at the opposite end - during coding. Moreover, design might not begin with a complete knowledge of all information needed. And the process might include obtaining additional information. Which side of this debate better reflects software design practice is an empirical question; the proposed definition of design is compatible with either.

3   Evaluating the Proposed Definition of Design

In this section we evaluate our definition of design, based on the degree to which it:

- Satisfies a set of four definition evaluation criteria (Appendix, Table 8)
- Incorporates areas of agreement in existing definitions (Tables 1 and 4)
- Resolves disagreements in existing definitions (§2.1)
- Appears usable and useful.

3.1   Definition Evaluation Criteria

Coverage. Whether a definition has proper domain coverage (i.e. can account for all phenomena in the domain to which it applies) is an empirical question, akin to a universal hypothesis. Therefore, the definition cannot be proven to be correct; however, it could be shown to have coverage problems by a counter example. Thus, we evaluated the definition by testing it against a diverse set of examples (such as those in Table 3). We found that the examples could be described in terms of the seven proposed aspects of design.

Meaningfulness. This refers to the requirement that all terms used have clear meaning. We have defined explicitly all terms having imprecise everyday meanings in Table 2.

Unambiguousness. This refers to the requirement that all terms used have unique meaning. All terms with potentially ambiguous meanings are defined. All terms not explicitly defined are intended in the everyday sense, that is, as defined in the dictionary. Where terms have multiple definitions, the intention should be clear from the context.

Ease of Use. The proposed definition is presented in natural language, and is segmented into clearly distinct elements, to ensure clarity for both practitioners and researchers. It is consistent with everyday notions of design and differentiates design from related terms such as invention, decision-making, and implementation. Table 3 provides examples of the elements of design to facilitate use of the definition.

3.2 Areas of Agreement

The relationship of each area of agreement to the proposed definition is analyzed in Table 4. Aspects of design mentioned in the literature that we demonstrated should not be included are marked "discounted." As can be seen in the table, all areas are accommodated explicitly or implicitly.

Table 4. Incorporation of Areas of Agreement

Concept	Consistency with Proposed Definition
Design as a *process*	implicit in the verb form of the proposed definition
Design as creation	explicit in the verb form of the proposed definition
Design as *planning*	encapsulated by the design 'specification;' however, planning may be lightweight, especially where specification occurs simultaneously with creating the object
System (as the object of the design)	included in the more abstract term, *design object*
Design as being *deliberate*, or having a *purpose, goal* or *objective*	explicitly included as *goals*
Design as an *activity*, or a collection of activities	implicit in the verb form of the proposed definition
Design as occurring in an environment (or domain/situation/context)	explicitly included as *environment*
Artifact, as the object of the design	included in the more abstract term, *design object*
Needs or *requirements*	explicitly included as *requirements*
Design as *organizing*	Discounted
Parts, components or elements	explicitly included as *primitives*
Design as a *human* phenomenon	Discounted
Constraints or *limitations*	explicitly included as *constraints*
Process (as the object of design)	included in the more abstract term, *design object* and listed as one of the main categories of design objects
Design as *creative*	Discounted
Optimizing	Discounted
Resources	implicit in *primitives* and the verb form of the proposed definition (since creating something always uses resources)

3.3 Areas of Disagreement
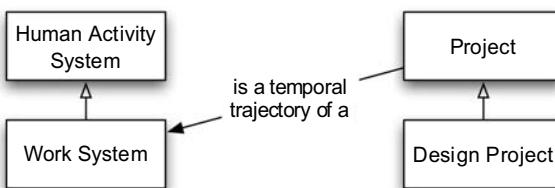
The proposed definition address each of the four areas of disagreement among existing definitions described in §2.1. First, different objects of design arise: system, artifact and process. We addressed this by using the more general term, *design object* and suggesting major categories of such objects. Second, disagreement exists concerning the scope of design: where or when a design begins and ends. We discussed this issue in §2.4. Third, disagreement exists as to whether design is a physical or mental activity. Clearly, design (for humans) is mental activity, albeit one that may be supported by physical activities (such as drawing diagrams or constructing physical models). The fourth disagreement, concerning what can be designed, was addressed in §2.3.

3.4 Usefulness and Usability

We suggest that the proposed definition of the design concept can inform practice in several ways. First, the elements of the definition (excluding agent) suggest a framework for *evaluating* designs: specification – is it complete? object – did we build the right thing? goals – are they achieved? environment – can the artifact exist and operate in the specified environment? primitives – have we assumed any that are not available to the implementers? requirements – are they met, i.e., does the object possess the required properties? constraints – are they satisfied? Second, the breakdown of design into elements can provide a checklist for practitioners. Each element should be explicitly identified for a design task to be fully explicated. For example, a project team might not be able to provide consistent and accurate estimates of design project costs if crucial elements are unknown. Third, a clear understanding of design can prevent confusion between design and implementation activities. Such confusion might lead to poor decisions and evaluation practices. For example, a manager who needs to hire team members for a project might view programmers as implementers only (not understanding the design involved in programming) and thus look for the wrong sorts of skills in applicants. Fourth, the elements of design can also be used to specify and index instances of design knowledge for reuse. This is demonstrated in §4.
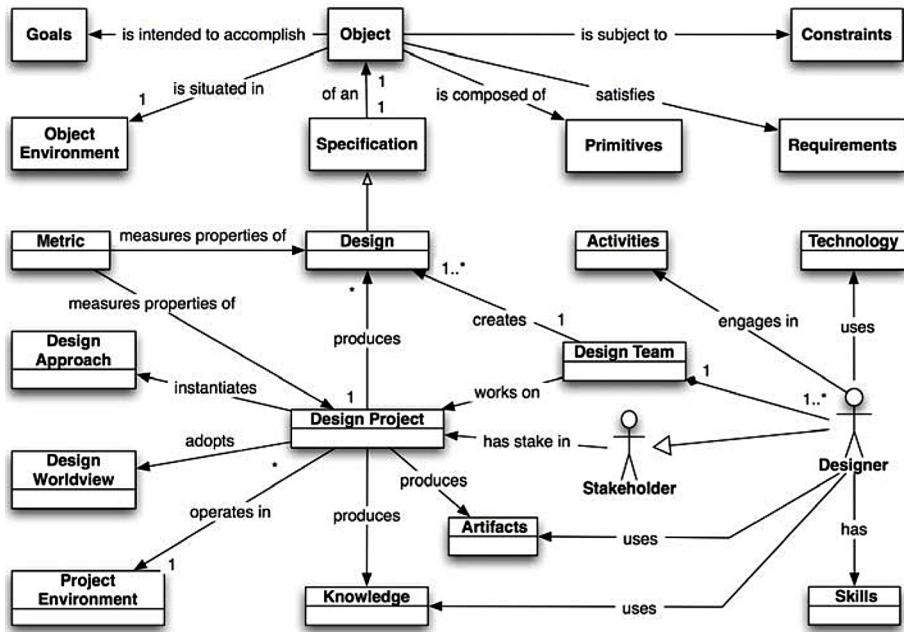
4 A Conceptual Model for the Design Project

We now propose a conceptual model (a set of concepts and their relationships) for design-related phenomena.[7] Here, we limit our discussion to design within the information systems field. Specifically, we view design as a *human activity that occurs within a complex entity, which can be thought of as a human activity system.* Alter [23] defines a *work system* as "a system in which human participants and/or machines perform work using information, technology, and other resources to produce products and/or services for internal or external customers," (p. 11). Expanding on this concept, we suggest that a *project* is *a temporal trajectory of a work system toward one or more goals*; the project ceases to exist when the goals are met or abandoned. Following this, we define a *design project* as a *project having the creation of a design as one of its goals*. This relationship is shown in Figure 3.

Fig. 3. Design Project Generalization Relationship. Shaded arrow indicates relationship; unshaded arrow indicates generalization.

---

[7] We note that to define a conceptual model of a domain, one needs to define the concepts used to reason about the domain (and their relationships). Such a conceptual structure is an ontology. Hence, we view our proposal as a conceptual model and as an ontology of concepts.

Fig. 4. Design Project Conceptual Model. Shaded arrows indicate reading direction, unshaded arrows indicate generalization, shaded diamonds indicate composition; all relationships many-to-many unless otherwise indicated.

The design project is the central concept of our conceptual model (depicted in Figure 4). Each concept is defined and each relationship is discussed in the following section (except the concepts from the definition of design, defined in Table 2).

Notes. 1) The relationships between the definition-of-design elements (e.g. constraints) and the other design project conceptual model elements (e.g., knowledge) are omitted to maintain readability. 2) The relationships between design approach and elements other than design project are unclear at this time and left for future work. 3) All shown concepts are implicitly part of the work system within which the design project takes place. 4) Creates is shown in this diagram as a relationship between design team and design, whereas Fig. 1 depicted creates as a relationship between agent and specification. In a design project, the design team is the agent. Furthermore, since the design project conceptual model includes the design concept, the model shows that the design team creates the design, which is a specification.

4.1   Discussion of Concepts

Alter [23] identifies nine elements of a work system:

- Work practices
- Participants
- Information

- Technologies
- Products and services the work system produces
- Customers for those products and services
- Environment that surrounds the work system
- Infrastructure shared with other work systems
- Strategies used by the work system and the organization

Since a design project is a trajectory of a work system, it should share all of these elements. Furthermore, since a design project is particular type of project, it should have properties not necessarily shared by other projects and work systems. Here, we discuss each element of the conceptual model, the relationships among elements, and the correspondence between elements of the conceptual model and elements of a work system. The conceptual model includes all the work system elements and, in addition, several elements specific to design projects, such as *design approach*.

Activities. Activities include the specific behaviors engaged in by participants in the design project. These may include interviewing stakeholders, modeling requirements, evaluating proposed design, etc. Activities exist at differing levels of granularity; for instance, modeling can be further divided into sub-activities such as writing scenarios, drawing entity relationship diagrams and then comparing the data models with the scenarios.

Participants and Stakeholders. Alter [23] defines participants as the "people who perform the work," (p. 13). Because individual participants vary among projects, we use the generic label, s*takeholder.* A stakeholder [24] is a person or entity with an interest in the outcome of the project. Design projects may have different types of stakeholders we specifically indicate the *designer* type for obvious reasons.

Designer. A designer is an agent that uses his or her skills to directly contribute to the creation of a design. This concept is specific to design projects.

Knowledge. Stakeholders may have and use knowledge during their involvement with the design project. In our interpretation, *knowledge* includes the kinds of information and knowhow used by stakeholders in a design project. To define knowledge, we extend the definition suggested by Bera & Wand [25]: given the states of the agent and the environment, knowledge is the information that enables an agent to select actions (from those available to the agent) so as to change the current state of affairs to a goal state. The design project can create knowledge as it proceeds – a tenant of the design science research paradigm [3].

Skill. A *skill* is a combination of mental and/or physical qualities that enable an agent to perform a specific action. Skills differ from knowledge as the latter enable one to *select* actions.

Technologies. *Technologies* are artificial, possibly intangible, tools and machines. Technologies can be used by the design team to create the design.

Design. The *design*, defined above, is the product that the design project aims to produce. This concept is specific to design projects.

Environment and Infrastructure. Fig. 4 combines Alter's *environment* and *infrastructure* constructs because both represent aspects of the project that are outside its

scope. Checkland [26] argues that, to properly model a system, the analyst must first model the system it serves. This wider system served by a design project is its environment. Alter [23] argues, "the work system should be the smallest work system that has the problems or opportunities that are being analyzed," (p. 22). Following this, then, the *environment* is the smallest coherent system served by the design project.

The environment construct is a potential source of confusion because *Design Project* and *Design* both have environments. The design project's environment is the work system in which the project occurs; the design's environment is the context in which in the object is to operate.

Design Approach and Strategy. A *design approach* is a set of beliefs about how design (and related activities) *should* be done. Examples include The Unified Software Development Process [27], and the Systems Development Lifecycle [28], [29]. According to Alter, "Strategies consist of the guiding rationale and high-level choices within which a work system, organization, or firm is designed and operates" (p. 14, [23]). As a design approach contains rationale and is implemented as choices, it corresponds to Alter's *strategy* construct. A design project may explicitly instantiate a formal design approach by using some or all of its elements. If a broad scope of design is taken (§2.4), a design approach can refer to the entire development process from problem identification to implementation and maintenance.

We have adopted the more general term, design approach, in lieu of design process or design methodology because what is referred to as "design process" often contain much more than a set of activities. Moreover, methodology is an overloaded concept used both as a formal word for 'method' and as the systematic study of methods. The design process concept is specific to design projects.

Design Team. All designers involved in a project comprise the design team. The design team engages in activities and uses technologies to create the design and other, intermediate artifacts. This concept is specific to design projects.

Artifacts. In this model, *artifact* is used in the broad, anthropological sense of any object manufactured, used or modified by agents in the design project. Examples include conceptual models, software development environments, whiteboards, and e-mails.[8]

Metric. A metric is a way or standard of taking a measurement, where measurement refers to a process of assigning symbols (often numbers) to an attribute of an object or entity (see [30], [31], [32]), and also the symbols assigned. In the case of a design project, metrics are used for evaluating specifications, designed objects, or the design project, among other things.

Design Worldview. Worldview is a way of translating the German word "Weltanschauung" meaning a way of looking onto the world. It is sometimes used in social sciences to indicate a set of high level beliefs through which an individual or group experiences and interprets the world. A precise definition of this concept is elusive. In Table 5 we suggest some possibilities for classifying Worldviews in the design context. Weltanschauungs are not mutually exclusive, i.e., a project could adopt several.

---

[8] This is not to be confused with the artifact that is the object of design.

Table 5. Identified Design Weltanschauung

Weltanschauung	Description	Proponents / Examples
Problem Solving	Design can be seen as an attempt to solve a known problem, a view characterized by the beliefs that a problem exists and is identifiable and that the success of a design is related to how well it solves the problem.	[2], [3], much of the design science and engineering literature.
Problem Finding	Design can be seen as an attempt to solve an unknown problem, implying that understanding the problem is part of the design process.	[33], much of the requirements engineering literature
Epistemic	Design can be seen as a learning process where actions that can lead to improvements to the current situation (in the eyes of stakeholders) are discovered.	[26]
Inspiration	Design can be seen as a result of inspiration, i.e., instead of beginning with a problem, design begins with an inspiration of the form 'wouldn't it be great if....'	the design of Facebook [34]
Growing	Design can be seen as growing an artifact, progressively improving its fit with its environment and purpose.	[4], [35]

Some design projects may explicitly adopt one or more design Weltanschauung. However, even without such an explicit view, every project participant brings a view of design to the project, and the combination of these views comprises the project's collective Weltanschauung. This concept is not necessarily common to all work systems.

4.2 Evaluation of the Conceptual Model of Design Projects

To evaluate the set of concepts underlying the proposed conceptual model, we use evaluation techniques suggested for ontologies. Ontology evaluation can proceed in several ways. The competency questions approach [36] involves simultaneously demonstrating usefulness and completeness by analytically proving that the ontology can answer each competency question in some question set. The ontology is then considered complete with respect to that question set. In contrast, Noy and Hafner [37] suggest two dimensions of ontology quality: coverage and usefulness. Coverage can be demonstrated by comparing an ontology to a reference corpus: terms in the corpus that do not fit into the ontology indicate lack of coverage. They further point out that "An important way of evaluating the capabilities and practical usefulness of an ontology is considering what practical problems it was applied to" (p. 72).

Since the proposed "ontology" is not intended to answer particular questions, evaluation with respect to coverage and usefulness seems preferable. Assessing the conceptual model's coverage is beyond the scope of this paper; however, a possible approach is evident. By surveying a range of design approaches, e.g. The Rational Unified Process, Agile Methods, The Waterfall Model, The Spiral Model, etc., A list of design concepts can be generated and compared to the proposed conceptual model.

Coverage can be measured by the extent to which these revealed concepts match the proposed concepts (usually as instances of the generic concepts suggested above).

We address usefulness in section (§5.2) by demonstrating how the conceptual model can be applied *in principle* to the practical problem of classifying and contrasting design approaches.

5   Potential Applications

In this section we discuss possible applications of the proposed definition of design and of the design project conceptual model. First, we suggest the use of the elements of the definition of design to classify and index design knowledge. Second, we discuss the use of the design project conceptual model for comparing and classifying approaches to software design.

Application 1: Design Knowledge Management System

The importance of reuse in software development has been widely recognized. For example, Mili, et al. [38] state that software reuse "is the (only) realistic opportunity to bring about the gains in productivity and quality that the software industry needs" (p. 528). Ambler [39] suggests a number of reuse types in software engineering, divided into two broad categories: *code reuse* and *knowledge reuse*.

Code reuse includes different approaches to organize actual code and incorporate it into software (e.g., libraries of modules, code fragments, or classes) and the use of off-the-shelf software. Code repositories can be considered design knowledge bases. Though some authors (e.g., [35]) argue that the best mechanism to communicate design is the code itself, sharing design is not the same as sharing design *knowledge*. Even well-commented code does not necessarily communicate design knowledge such as the rationale for structural decisions (e.g., why information was stored in a certain structure).

Knowledge reuse refers to approaches to organizing and applying knowledge about software solutions, not to organizing the solutions themselves. It includes algorithms, design patterns and analysis patterns.[9] Perhaps the most successful attempt to codify software design knowledge is the design patterns approach. A design pattern is an abstract solution to a commonly occurring problem. The design pattern concept was originally proposed in the field of architecture [40] and became popular in software engineering following the work by Gamma et al. [41].[10]

Despite the apparent benefits of sharing design knowledge, it has been observed that it is difficult to accomplish. Desouza et al. [42] claim that "Experts and veterans continue to shun reuse from public knowledge spaces" and that when the needed

---

[9]   Other approaches to organizing software development knowledge include Architectural Patterns, Anti-Patterns, Best Practices and development methods. As well, standards and templates (e.g., for documentation) can be considered organized knowledge.

[10] The Portland Pattern Repository (http://c2.com/ppr/) is an example of a design pattern repository that could be called a design knowledge base.

artifact "was not found in their private space … it was also less costly for them to recode the desired artifact than to conduct a global search for one" (p. 98). This indicates the difficulties of locating needed design knowledge (or other software artifacts). One way to facilitate search is to classify and index design knowledge on meaningful dimensions. We now demonstrate by example how the proposed definition of design can provide such dimensions and thus help index instances of design knowledge.

An Example. In programming, an iterator object traverses a collection of elements, regardless of how the collection is implemented. Iterators are especially useful when the programmer wants to perform an operation on each element of a collection that has no index. The iterator design pattern is a description of how best to implement an iterator. Table 6 shows how the design knowledge represented by the iterator design pattern might be indexed using the elements of the proposed definition of design. Note that, in this application the goals, requirements, etc. are properties of the iterator, not of the design pattern. The goal of the design pattern, for instance, is to explain how to implement an iterator (and not to traverse a collection).

Table 6. Example of Design Knowledge Indexing

Object Type	Symbolic Script
Object	Iterator
Agent	application programmer
Goals	access the elements of a collection of objects
Environment	object-oriented programming languages
Primitives	primitives and classes available in object-oriented programming languages
Requirements	have a means of traversing a collection, be implementable with respect to a variety of collections, etc.
Constraints	must not reveal how the objects in the collection are stored, etc.

By classifying design knowledge according to these dimensions, a designer can ask questions of the form 'are there any design patterns (*object*) for traversing a collection (*requirement*) in an object-oriented language (*environment*)?' We suggest that such classification can help organize and share design knowledge and thus help improve designers' effectiveness and efficiency in locating and applying useful design knowledge.

Application 2: Design Approach Classification Framework

Classifying design approaches is important for several reasons. First, practitioners need guidance in selecting appropriate design approaches for their situations. Second, such classification can facilitate comparative research on approaches. Third, it can guide the study of the methods employed by experienced developers (which, in turn, can inform research on software design and software processes).

At least two types of classifications of design approaches are possible. First, a classification can be based on the actual elements (e.g. steps, interim products) that comprise a design approach or process. This can be termed a "white-box" approach.

Second, a classification can be based on the environment that "surrounds" a design approach. For example, specific objectives of the approach, the view of design it embeds, and the roles of stakeholders. This can be termed a "black-box" approach.

We suggest that the proposed design project conceptual model can be used to create a black-box classification scheme for design approaches. In the following we demonstrate how this can be done by examples. Using dimensions derived from the design project conceptual model, Table 8 classifies three design approaches: the Soft Systems Methodology [26], Extreme Programming [35] and the Rational Unified Process [16]. We chose these three because they are each prominent in the literature and represent significantly different perspectives.

6   Discussion and Implications for Software Design Research

6.1   Completeness, Design Agency and Software Architecture

For years, researchers have argued that informal specifications may suffer from incompleteness (e.g., [43]). Above, we defined a specification as a detailed description of an object in terms of its structure, namely the components used and their connections. This allows a more precise characterization of incompleteness. We suggest that a design specification is complete when the *relevant* structural information that has been specified is sufficient for generating (in principle) an artifact that meets the requirements.[11]

Based on the notion of completeness we have defined above, we can now identify three forms of incompleteness. First, relevant components or connections may be missing. For example, the specification for a bicycle may be missing the qualification that the tires be *attached* to the rims. Second, a particular component or connection may be insufficiently described. For example, it may not be clear from the specifications *how* the tires should be attach to the rims or which tire to use. (Please note, here we are not distinguishing here between incompleteness and ambiguity.) Third, a component may not be part of the set of primitives but can be designed based on existing primitives or other components. The design will not be complete until specifications exist for all such components.

Completeness is not an end state for a design specification. Future changes in the set of primitives may render a previously-complete specification incomplete. Furthermore, many researchers now agree on the importance of "the fluidity, or continued evolution, of design artifacts," ([44], p. 36). In situations where future conditions are difficult or impossible to predict, one response is to focus on the evolvability and adaptability of the design object [2], [45]. The characterization of design advanced here provides important implications for design fluidity. First, specification completeness does not imply constancy. A design specification can clearly be evolved over time by its original creator, the design object's users, or others, to respond to changing conditions. Furthermore, the elements of the proposed definition enumerate classes of

---

[11] Since it is impossible to list all of the properties of any object, we limit our discussion to "relevant" properties, i.e., a sufficient subset of properties to allow a "generating machine" (e.g., a human being or a manufacturing robot) to deterministically assemble the object.

Table 7. Example Classification of Design Approaches

	Soft Systems Methodology (SSM)	Extreme Programming	Rational Unified Process (RUP)
Object	human activity systems	software	software
Weltanschauung	epistemic	growing	problem solving
Metrics	situation dependent "measures of performance;" the 5 E's: efficacy, efficiency, effectiveness, ethicality, elegance	advocated, but none provided; differentiates internal and external quality	defines metrics as part of the process; fundamental quality measure: 'does the system do what it is supposed to?'
Nature of Specification	action items, i.e., some action that can be taken to improve the situation, in the eyes of the stakeholders	source code	UML models (use cases and diagrams); source code
Activities	semi-structured interviews, analysis, modeling, debate	coding, testing, listening, designing (refactoring)	broadly: requirements gathering, analysis and design, implementation, testing, deployment, configuration and change management, project management (each with sub activities)
Artifacts	interview guides and transcripts, collections of notes, rich pictures	prototypes, test suites	stakeholder requests, vision, business case, risk list, deployment plan, analysis model, etc.
Users	owner, actor, customer	programmers/developers, clients	RUP users take on one or more of six role categories: analysts, developers, managers, testers, production and support, and additional.
Stakeholders	stakeholders is an explicit concept in SSM	divided into "business" and "development"	"stakeholder" is a "generic role" that refers to "anyone affected by the outcome of the project" (p. 276)
Tools	rich pictures, interview guides, debates and group discussions	story cards, diagrams, an integration machine, several development workstations	IBM Rational Suite

possible changing conditions, in response to which the design object or specification might need to evolve. For example, the specification might be modified in response to changes in the environment. Finally, the set of requirements might contain stipulations for a design object's evolvability by end-users or others.

This raises questions of who exactly, in a typical software project, is the design agent? We have defined the design agent as the entity or group of entities that specifies the structural properties of the design object. When users are involved in design, whether a user is part of the design agent depends on the nature of his or her involvement. Simply providing information, such as requirements, does not make a user part of the design agent, nor does testing and giving feedback. To share in design agency, the user must *make at least one structural decision regarding the design object*. As a complete discussion of this issue would require incorporating the vast literature on authority and organizational power (e.g., [46], [47]), here we simply point out that official authority to make a structural decision does not necessarily coincide with the practical reality of who makes a decision. The key to identifying the design agent is in separating those individuals (or groups) who provide information about constraints, primitives and the other design elements, and those that decide on structural properties.

Another theme currently gaining significant attention is software architecture [44]. Software architecture is the level of design concerned with "designing and specifying the overall system structure," ([48], p.1). This presents a possible difficulty: if a specification is a description of the components of a design object and their relationships, which components and relationships are parts of the software architecture? How does one distinguish high-level components and relationships from low-level ones? A design specification for a complex system might exist simultaneously at many levels of abstraction. Alternatively (and perhaps more likely) high-level components are defined in terms of lower-level components and these are defined in terms of even lower-level components, etc., until everything is defined in terms of primitive components. In this multilevel view of design, the software architecture concept is a threshold above which is architecture, and below which is "detailed design." Is this threshold arbitrary? At this time, we can only suggest these fundamental questions about software architecture as topics for future research.

6.2 Implications for Research

The proposed characterization of design also gives rise to several implications for design research. To date, much design research has been prescriptive, addressing practical recommendations and guidance for software development; yet little theoretical, and even less empirical, treatment of software design exists [49]. This has led to many calls for field research in this area (e.g., [1], [49]). Defining design as the process by which one specifies an object's structural properties raises several important research topics:

1. How is software designed in practice?
2. To what extent is each element of the proposed definition (requirements, primitives, etc.) known when design begins?

3. Can a single theory explain all of the diverse behaviors involved in software design?
4. How do designers discover each kind of information?
5. What factors influence design project success?

Put another way, academic treatment of software design may involve developing and testing interdependent *process* and *causal* theories of design. Process theories can be used to explain *how* design occurs.[12] Causal theories deal with effects of some variables on others and can be used to suggest how to design better.

6.3   Goals Versus Requirements in Information Systems Development

The notion of *goal* is considered essential in requirements engineering as the concept that captures the motivation for developing a system ("why") and the way to define objectives at various level of abstraction [18]. Our definition of design includes both goals and requirements. We now describe briefly how these two concepts are related within this context.

We start by observing that in the information systems context, a design object is an artifact situated[13] in an environment termed the *application domain* and designed to support activities of the *application domain*. Typically, the application domain is an organizational setting such as a business or a part of a business. The application domain itself operates within an *external environment*. For example, a business is embedded within a business environment comprising customers, suppliers, competitors, service providers, and regulatory bodies. The application domain and the external environment interact: the environment generates *stimuli* that invoke *actions* in the domain. The actions of the domain can *impact* its environment. Similarly, the artifact is situated in the domain. The domain and the artifact interact: the domain creates external stimuli which invoke actions in the artifact. The actions of the artifact can impact the domain. Once the artifact is embedded a change occurs: the domain now includes the artifact. Now the modified domain (with the included artifact) interacts with the external environment. This view is depicted in Figure 5.
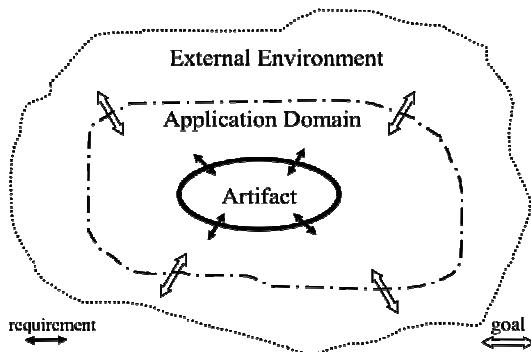
We define *domain goals*, or simply *goals*, as the intended impact of the actions in the domain on the external environment.[14] The purpose of the artifact is to enable the domain to accomplish these goals more effectively and efficiently. The artifact does this by responding to *stimuli* from the domain is ways that will support the domain in accomplishing the goals. Accordingly, requirements can be defined as the *properties*

---

[12] According to Van de Ven and Poole [50] "a process theory [is] an explanation of how and why an organizational entity changes and develops."

[13] The word "situated" should not be taken literally in the physical sense, but in sense that the artifact acts in a role of a component of the domain, and interacts with other components.

[14] To demonstrate, consider, for example, profitability, which might appear related to the business rather than to its environment. However, profitability is the outcome of exchanges between a business and its environment, and the business should act in a way these exchanges create the desired outcome.

Fig. 5. Separate Domains of Goals and Requirements

that the artifact should possess in order to accomplish its purpose. These requirements can be of two types:

1. *Structural requirements* are intended to assure the artifact can match well with the other components of the domain, or those of the external environment it might interact with.
2. *Behavioral requirements* define the *desired responses* of the artifact to stimuli from the domain (or from the environment) generated when the domain is working to accomplish its goals. These responses, in turn, affect the domain (and, directly, or indirectly, the environment).

The *Requirements definition* process can be viewed as identifying what properties (structural and behavioral) the artifact should possess in order to support the domain in accomplishing the goals. Design can be viewed as the way to assemble available types of components in order to accomplish an artifact that meets the requirements.

7 Conclusion

The work we describe here is motivated by the observation that a clear, precise and generally accepted definition of the concept of design can provide benefits for research, practice and education. Our literature study indicated that such a definition was not available. We therefore undertook to propose a definition of the design concept. The definition views the design activity as a *process*, executed by an *agent*, for the purpose of generating a *specification* of an *object* based on: the *environment* in which the object will exist, the *goals* ascribed to the object, the desired structural and behavioral properties of the object (*requirements*), a given set of component types (*primitives*), and *constraints* that limit the acceptable solutions. As one possible application of our definition we demonstrate how it can be used to index design knowledge to support reuse of this knowledge.

As a second step, we situate the design concept in a network of related concepts appropriate to the information systems and software development domain by

proposing a conceptual model for design projects. The intent of this conceptual model is to facilitate study of design projects by identifying and clarifying the main relevant concepts and relationships. We demonstrate the usefulness of this conceptual model by using it to compare several approaches to system and software design.

Finally, we link our proposed definition of design to current themes in design research, in particular, the notion of requirements as used in system development.

One purpose of this work is to facilitate theoretical and empirical research on design phenomena. We hope this paper will contribute to clarifying understanding and usage of design and related concepts and encourage scientific research on design. Another purpose is to create a set of concepts that can guide practices and education in the domain of information systems and software design.

This article includes examples of design from diverse areas such as prehistoric hunters, artists, and architects. The reader may question whether such a broad perspective on design is useful for studying software development. It will be of interest to find out if software designers are more similar to engineers or to artists, or perhaps are a class on their own. This can only be answered by *observing* the behaviors of a wide range of those who are engaged in software design (elite and amateur, engineers and "hackers", formally trained and self-taught). Having a well-defined set of concepts to describe phenomena related to design and to design projects and to reason about these phenomena can provide guidance for such undertaking.

References

1. Freeman, P., Hart, D.: A science of design for software-intensive systems. Communications of the ACM 47(8), 19–21 (2004)
2. Simon, H.A.: The Sciences of the Artificial, 3rd edn. MIT Press, Cambridge (1996)
3. Hevner, A.R., March, S.T., Park, J., Ram, S.: Design science in information systems research. MIS Quarterly 28(1), 75–105 (2004)
4. March, S.T., Smith, G.F.: Design and natural science research on information technology. Decision Support Systems 15(4), 251–266 (1995)
5. Accreditation board for engineering and technology, Inc. Annual report for the year ending September 30, 1988. New York, USA (1988)
6. van Engers, T.M., Gerrits, R., Boekenoogen, M., Glassée, E., Kordelaar, P.: Power: using uml/ocl for modeling legislation - an application report. In: ICAIL 2001: Proceedings of the 8th international conference on Artificial intelligence and law, pp. 157–167. ACM Press, New York (2001)
7. Bradel, B., Stewart, K.: Exploring processor design using genetic programming. In: ECE1718 Special Topics in Computer Hardware Design: Modern and Emerging Architectures. University of Toronto, Toronto (April 2004)
8. Eekels, J.: On the fundamentals of engineering design science: The geography of engineering design science. part 1. Journal of Engineering Design 11, 377–397 (2000)
9. Breuer, T., Ndoundou-Hockemba, M., Fishlock, V.: First observation of tool use in wild gorillas. PLoS Biol. 3(11) (2005)

10. Mulcahy, N., Call, J.: Apes save tools for future use. Science 312(5776), 1038–1040 (2006)
11. Churchman, C.W.: The design of inquiring systems: Basic concepts of systems and organization. Basic Books, New York (1971)
12. Alexander, C.W.: Notes on the synthesis of form. Harvard University Press (1964)
13. Gero, J.S.: Design prototypes: A knowledge representation schema for design. AI Magazine 11(4), 26–36 (1990)
14. Bourque, P., Dupuis, R. (eds.): Guide to the software engineering body of knowledge (SWEBOK). IEEE Computer Society Press, Los Alamitos (2004)
15. Siddiqi, J., Shekaran, M.: Requirements engineering: The emerging wisdom. IEEE Software, 15–19 (March 1996)
16. Kruchten, P.: The Rational Unified Process: An Introduction, 3rd edn. Addison-Wesley Professional, Reading (2003)
17. Pahl, G., Beitz, W.: Engineering Design: A Systematic Approach. Springer, London (1996)
18. Lamsweerde, A.v.: Goal-oriented requirements engineering: a guided tour. In: Proceedings of the Fifth IEEE International Symposium on Requirements Engineering, pp. 249–262 (August 2001)
19. Hammer, M., Champy, J.: Reengineering the corporation: A manifesto for business revolution. Business Horizons 36(5), 90–91 (1993),
http://ideas.repec.org/a/eee/bushor/v36y1993i5p90-91.html
20. van der Alast, W.M.P.: Workflow verification: Finding control-flow errors using petri-net-based techniques. In: van der Aalst, W.M.P., Desel, J., Oberweis, A. (eds.) Business Process Management. LNCS, vol. 1806, pp. 161–183. Springer, Heidelberg (2000)
21. Soffer, P., Wand, Y.: Goal-driven analysis of process model validity. Advanced Information Systems Engineering, 521–535 (2004)
22. Royce, W.: Managing the Development of Large Software Systems. Proceedings of IEEE WESCON 26 (1970)
23. Alter, S.: The Work system method: Connecting people, processes, and IT for business results. Work System Press (2006)
24. Freeman, R.: Strategic Management: A stakeholder approach. Pitman, Boston (1984)
25. Bera, P., Wand, Y.: Conceptual Models for Knowledge Management Systems. Working paper, University of British Columbia (2007)
26. Checkland, P.: Systems Thinking, Systems Practice. John Wiley & Sons, Ltd., Chichester (1999)
27. Jacobson, I., Booch, G., Rumbaugh, J.: The unified software development process. Addison-Wesley Longman Publishing Co., Inc., Boston (1999)
28. Department of Justice. The department of justice systems development life cycle guidance document
29. Manlei, M., Teorey, T.: Incorporating behavioral techniques into the systems development lifecycle. MIS Quarterly 13(3), 257–274 (1989)
30. Fenton, N.: Software measurement: A necessary scientific basis. IEEE Trans. Softw. Eng. 20(3), 199–206 (1994)
31. Finkelstein, L.: A review of the fundamental concepts of measurement. Measurement 2(I), 25–34 (1984)
32. Roberts, F.: Measurement Theory with Applications to Decision Making, Utility and the Social Sciences. Addison Wesley, Reading (1979)
33. Polya, G.: How to Solve It: A New Aspect of Mathematical Method, 2nd edn. Princeton University Press, Princeton (1957)

34. Kessler, A.: WSJ: Weekend interview with Facebook's Mark Zuckerberg

35. Beck, K.: Extreme programming eXplained: embrace change. Addison-Wesley, Reading (2000)

36. Grüninger, M., Fox, M.: Methodolgy for the design and evaluation of ontologies. In: Proceedings of the IJCAI Workshop on Basic Ontological Issues in Knoweldge Sharing, Menlo Park CA, USA. AAAI Press, Menlo Park (1995)

37. Noy, N., Hafner, C.: The state of the art in ontology design. AI Magazine, 53–74 (Fall 1997)

38. Mili, H., Mili, F., Mili, A.: Reusing software: Issues and research directions. IEEE Transactions on Software Engineering 21(6), 528–562 (1995)

39. Ambler, S.: A realistic look at object-oriented reuse. Software Development 6(1), 30–38 (1998)

40. Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I., Angel, S.: A Pattern Language: Towns, Buildings, Construction. Oxford University Press, Oxford (1977)

41. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design patterns: Elements of reusable object-oriented software. Addison-Wesley, Boston (1995)

42. Desouza, K.C., Awazu, Y., Tiwana, A.: Four dynamics for bringing use back into software reuse. Commun. ACM 49(1), 96–100 (2006)

43. Reubenstein, H., Waters, R.: The requirements apprentice: Automated assistance for requirements acquisition. IEEE Trans. Softw. Eng. 17(3), 226–240 (1991)

44. Hansen, S., Berente, N., Lyytinen, K.: Requirements in the 21st century: Current practice and emerging trends. In: The Design Requirements Workshop, Cleveland, Ohio, USA, June 3-6 (2007)

45. Gregor, S., Jones, D.: The anatomy of a design theory. Journal of the Association for Information Systems 8, 312 (2007)

46. Aghion, P., Tirole, J.: Real and Formal Authority in Organizations. Journal of Political Economy 105, 1–29 (1997)

47. Pfeffer, J.: Managing with Power: Politics and Influence in Organizations. Harvard Business School Press, Cambridge (1992)

48. Garlan, D., Shaw, M.: An introduction to software architecture. In: Ambriola, V., Tortora, G. (eds.) Advances in software engineering and knowledge engineering, pp. 1–39. World Scientific, Singapore (1993)

49. Wynekoop, J., Russo, N.: Studying system development methodologies: an examination of research methods. Information Systems Journal 7, 47–65 (1997)

50. Van de Ven, A., Poole, M.: Explaining Development and Change in Organizations. Acad. Manage. Rev. 20, 510 (1995)

51. Hinrichs, T.R.: Problem-solving in open worlds: a case study in design. PhD thesis, Atlanta, GA, USA (1992)

Appendix: Analysis of Existing Definitions of Design

We have identified at least 33 definitions of design and sub-types of design (such as "software design" and "urban design") in the literature. Though design has several meanings, we have focused on the meaning involving plans for an object and planning or devising as a process.

We employed judgment sampling and snowball sampling, i.e., we made educated guesses as to where to look, and then investigated promising references. This strategy was consistent with our goal of identifying as many relevant definitions as possible.

To evaluate the definitions we applied a set of four main criteria: coverage, meaningfulness, unambiguousness and ease of use (see Table 8). The first three are derived from the evaluation criteria for good theories mentioned, for example, by Casti (1989, p.44-45). The fourth is a pragmatic criterion. We do not claim that these are the *best* criteria, but, in the absence of a guiding theory for evaluating definitions, that they are reasonable and have face validity.

To give the reader a sense of the thought process behind the analysis, we discuss two representative examples of the definitions encountered. The first example is by Engers et al. [6] who define design as "the creative process of coming up with a well-structured model that optimizes technological constraints, given a specification." This definition has both meaningfulness and coverage problems. First, the meaning of 'optimizes technological constraints' is unclear. In optimization techniques, one optimizes the characteristics of an object subject to constraints, not the constraints themselves. Second, the use of "well-structured" paints an idealistic portrait of design. This confounds the notion of design with measures for design quality. For example, an inexperienced computer science student can design a personal organizer application. The application might not be "well-structured", but is nonetheless *designed*. Thus, this definition omits activities that are clearly design. The second example is that of Hinrichs [51] who defines design as "the task of generating descriptions of artifacts or processes in some domain," (p. 3). This also has coverage problems. "My chair is grey" is a description of an artifact in a domain, but is clearly not a design. The problem here is that the definition relates to previously-designed artifacts. Thus, this definition includes phenomena that are not design.

The complete analysis of existing definitions is presented in Table 9. Of the 33 definitions identified, we have found that all seem to have coverage problems, at least 12 have meaningfulness problems and at least three have some form of ambiguity.

Table 8. General Definition Evaluation Criteria

	Criterion	Definition	Example of Error
Necessary	Coverage	Proper coverage means including all appropriate phenomena (completeness), and only appropriate phenomena. If a definition has improper coverage, it excludes at least one phenomenon that it should include or includes at least one phenomenon it should not.	Defining "human communication" to include only speech, will not address non-verbal communication (e.g. body language).
	Meaningfulness	Each term comprising a definition must have a commonly accepted meaning in the given context or must have been pre-defined. Each combination of terms must be directly understandable from the meaning of terms, or have been predefined.	Defining a zombie as 'the living dead' is inappropriate because, even though 'living' and 'dead' have commonly accepted meanings, their juxtaposition forms an oxymoron.
	Unambiguousness	Each term comprising a definition must have exactly one meaning in the given context; furthermore, the definition as a whole must have only one valid interpretation.	Defining political oratory as 'oral rhetoric related to politics' is inappropriate because 'rhetoric' is a contronym, i.e., has two contradictory meanings.
Optional	Ease of Use	Ideally, a definition should be easy to understand and remember, applicable in disparate situations, and readily differentiate between included and excluded phenomena. Simplicity, parsimony and concreteness are all aspects of Ease of Use. These aspects are at least in part subjective and depend on who uses the definition.	Defining the Natural Numbers as 'the smallest set satisfying the two properties: A) 1 is in N; and B) if n is in N, then $n + 1$ is in N" while clearly correct, would score poorly on Ease of Use in a low-level mathematics class.

Table 9. Analysis of Existing Definitions

Source	Definition	Criticism
Accreditation Board for Engineering and Technology (1988)	"Engineering design is the process of devising a system, component, or process to meet desired needs. It is a decision making process (often iterative), in which the basic sciences, mathematics, and engineering sciences are applied to convert resources optimally to meet a stated objective."	*Coverage* – the definition is idealistic and unnecessarily limiting in its use of "optimally." E.g., the building in which I work is far from optimal, but it was still designed. *Meaningfulness* – it is not clear what "desired needs" are.
Alexander (1964)	"The process of inventing physical things which display new physical order, organization, form, in response to function."	*Coverage* – this definition excludes the design of intangible things, such as processes. *Unambiguousness* – it is not clear whether thing must display new physical order, organization AND form, or new physical order, organization OR form.
Archer (1979)	"Design is, in its most general educational sense, defined as the area of human experience, skill and understanding that reflects man's concern with the appreciation and adaptation in his surroundings in the light of his material and spiritual needs."	*Coverage* – design is an activity, not an "area of human experience..." One can design with little or no experience, skill and understanding. E.g., the application programmer who designs a graphical user interface without experience in, skill in or understanding of the principles of interface design.
Beck (2000)	"Designing is creating a structure that organizes the logic in the system"	*Coverage* – excludes forms of design that organize things other than logic, e.g., urban planning organizes space.
Blumrich (1970)	"Design establishes and defines solutions to and pertinent structures for problems not solved before, or new solutions to problems which have previously been solved in a different way."	*Coverage* – Unnecessarily limits design to solutions not previously solved. Excludes independent invention and finding new ways to solve old problems. E.g., by this definition, new cars are not designed because we already have cars.
Bourque & Dupuis (2004)	"Design is defined in [IEEE610.12-90] as both "the process of defining the architecture, components, interfaces, and other characteristics of a system or component" and "the result of [that] process." Viewed as a process, software design is the software engineering life cycle activity in which software requirements are	*Coverage* – even within the software domain, this definition is far too restrictive. If someone simply writes software without creating an intermediate description of its structure, this is still design. Design is, furthermore, not limited to the phase of the software engineering life cycle between requirements analysis and construction; it is in no way clear that these phases can be practically distinguished

Table 9, (*continued*)

Source	Definition	Criticism
	analyzed in order to produce a description of the software's internal structure that will serve as the basis for its construction."	in all situations.
Buchanan (2006)	"Design is the human power to conceive, plan and realize all of the products that serve human beings in the accomplishment of their individual or collective purposes."	*Coverage* – Design is not an ability ("power") but an activity. E.g., drawing blueprints for a house, by this definition, is not design. *Unambiguousness* – it is not clear what "products" are – does this include processes and strategies as well as consumer goods?
Complin (1997)	"'design' is used to refer to the abstract description of the functional architecture of both real or possible systems."	*Coverage* – Excludes design of simple things, such as boomerangs. *Meaningfulness* – it is not clear what "functional architecture" entails
Engers et al. (2001)	"the creative process of coming up with a well-structured model that optimizes technological constraints, given a specification."	*Coverage* – excludes all suboptimal artifacts. *Meaningfulness* – the meanings of "specification" and model are unclear.
Eckroth et al. (2007)	"Design (as a verb) is a human activity resulting in a unique design (specification, description) of artifacts. Therefore, what can be designed varies greatly. However, common to all design is intention: all designs have a goal, and the goal is typically meeting needs, improving situations, or creating something new. Thus, design is the process of changing an existing environment into a desired environment by way of specifying the properties of artifacts that will constitute the desired environment; in other words, creating, modifying, or specifying how to create or alter artifacts to meet needs. In addition, it is best communicated in terms of a particular context, as previous knowledge, experience, and expectations play a strong role in designing and understanding designs."	*Coverage* – excludes independently invention of previously created artifacts and design starting from a hypothetical situations
FitzGerald and Fitz-	"design means to map out, to plan, or to arrange the	*Coverage* – this excludes artifacts that satisfy only some of their ob-

Table 9. (*continued*)

Source	Definition	Criticism
Gerald (1987)	parts into a whole which satisfies the objectives involved."	jectives. E.g., Enterprise-Resource Planning software does not always satisfy its stated objectives (Trunick 1999), but surely it as still designed.
Freeman and Hart (2004)	"design encompasses all the activities involved in conceptualizing, framing, implementing, commissioning, and ultimately modifying complex systems—not just the activity following requirements specification and before programming, as it might be translated from a stylized software engineering process."	*Coverage* – simple systems and non-systems can also be designed, e.g. an oar, a boomerang. *Meaningfulness* – the activities are not defined or clearly explained; furthermore, enumerating the tasks *encompassed by* design does not necessarily capture the *meaning of* design.
Gero (1990)	"a goal-oriented, constrained, decision-making, exploration and learning activity which operates within a context which depends on the designer's perception of the context."	*Coverage* – The problem here is subtle. Not all design is a decision making activity; some designers, such as sculptors, may proceed fluidly without discrete decisions. It could be argued that their decisions are implicit, but then this definition would include activities such as public speaking. Decision-making is a perspective on design, not inherent to it. Furthermore, the idea of designing as leading to a new or changed artifact is missing.
Harris (1995)	"A collection of activities designed to help the analyst prepare alternative solutions to information systems problems."	*Coverage* – excludes design for non problems outside information system. *Meaningfulness* – use of "designed" is circular
Hevner et al. (2004)	"design is the purposeful organization of resources to accomplish a goal."	*Coverage* – includes organization tasks that do not constitute design, e.g., alphabetizing books. *Meaningfulness* – resources is undefined; e.g., what are the resources organized to create a military strategy? What are the resources that are being organized in graphics design? *Unambiguousness* – usage of "organization;" is it physical organization of resources, or mental?
Hinrichs (1992)	"the task of generating descriptions of artifacts or	*Coverage* – includes descriptions of artifacts that are not, e.g., "the chair is

Table 9. (*continued*)

Source	Definition	Criticism
	ing, and analyzing facts about a particular [information system] and the environment in which it operates. *Systems design* then is the conception, generation and formation of a new system, using the analysis results."	a complete specification (e.g., of a bridge) rather than a system. *Meaningfulness* – this definition hinges on undefined terms "conception, generation and formation"
Jobs (2000)	"Design is the fundamental soul of a man-made creation that ends up expressing itself in successive outer layers of the product or service."	*Coverage* – excludes designs not involving a product or service and designs that are not "man-made" *Meaningfulness* – the meaning of "fundamental soul" is unclear
Love (2002)	"'Design' — a noun referring to a specification or plan for making a particular artefact or for undertaking a particular activity. A distinction is drawn here between a design and an artifact — a design is the basis for, and precursor to, the making of an artefact." "'Designing' —human activity leading to the production of a design."	*Coverage* – 1) the strict time sequencing implied by this definition is unnecessarily limiting; e.g., in software engineering simultaneous design and creation is arguably the preferred approach (see Martin, 1991 and Beck, 2000), 2) Design is not strictly a human activity *Meaningfulness* - "Artefact" is undefined, so the scope is unknown.
Merriam-Webster (2006) [verb]	(verb) "*transitive senses* 1 : to create, fashion, execute, or construct according to plan : DEVISE, CONTRIVE 2 a : to conceive and plan out in the mind <he *designed* the perfect crime> 4 a : to make a drawing, pattern, or sketch of b : to draw the plans for"	*Coverage* – t would include drawing a diagram of a tree (not design), but not collaboratively writing a new search algorithm (design). *Meaningfulness* – circular reference to 'design'
Merriam-Webster (2006) [noun]	"1 a : a particular purpose held in view by an individual or group <he has ambitious *designs* for his son> b : deliberate purposive planning <more by accident than *design*> 2 : a mental project or scheme in which means to an end are laid down 4 : a preliminary sketch or outline showing the main features of something to be executed : DELINEATION 5 a : an underlying scheme that governs functioning, developing, or un-	*Coverage* - Overall, this definition does not provide a unifying notion of the minimum requirements to call something a design, and does not separate designing from planning. *Meaningfulness* – circular reference to 'designs'

Table 9. (*continued*)

Source	Definition	Criticism
	complishing something (as a scientific experiment); *also* : the process of preparing this 6 : the arrangement of elements or details in a product or work of art 7 : a decorative pattern 8 : the creative art of executing aesthetic or functional designs"	
Miller's (2005)	"Design is the thought process comprising the creation of an entity,"	*Coverage* – design can encompass more than just a thought process; e.g., drawing diagrams. Thought processes cannot create physical things.
Nunamaker et al. (1991)	"Design ... involves the understanding of the studied domain, the application of relevant scientific and technical knowledge, the creation of various alternatives, and the synthesis and evaluation of proposed alternative solutions."	*Coverage* – if a person has a breakthrough idea and implements a single, innovative artifact, without considering any alternatives, this would still be design. Depending on how one defines "scientific knowledge," many designers throughout history would be excluded by this definition.
Papenek (1983)	"Design is a conscious and intuitive effort to impose meaningful order.... Design is both the underlying matrix of order and the tool that creates it."	*Coverage* – Would include all ordering activities, such as alphabetizing books *Meaningfulness* – 'underlying matrix of order' is undefined. *Ease of use* – unclear how to operationalize "matrix of order"
Partners of Pentagram (1978)	"A design is a plan to make something; something we can see or hold or walk into; something that is two-dimensional or three-dimensional, and sometimes in the time dimension. It is always something seen and sometimes something touched, and now and then by association, something heard."	*Coverage* – This definition excludes design of an incorporeal thing, e.g., a philosophy, society or strategy.
Pye (1964)	"Invention is the process of discovering a principle. Design is the process of applying that principle. The inventor discovers a class of system – a generalization – and the designer prescribes a particular embodiment	*Coverage* – Designing need not comply with principles; e.g., one might design a software interface with absolutely no knowledge of any principles regarding interface design. The interface is no less designed by someone.

Table 9. (*continued*)

Source	Definition	Criticism
Richardson (1984)	"Design is a general term, comprising all aspects of organization in the visual arts."	*Coverage* – excludes design in architecture, engineering, etc.
Schurch (1999)	"Therefore, urban design might be more clearly defined as "giving physical design direction to urban growth, conservation, and change..." (Barnett, 1982, p. 12) as practised by the allied environmental design professions of architecture, landscape architecture and urban planning and others, for that matter, such as engineers, developers, artists, grass roots groups, etc."	*Coverage* – Though design intuitively may give direction, not all instances of giving direction are design; e.g., the mere command "give the castle a moat" gives direction, but is clearly not design *Meaningfulness* – 'physical design direction' undefined
Simon (1996)	"Design is devising courses of action aimed at changing existing situations into preferred ones."	*Coverage* – excludes designs beginning from hypothetical situations, e.g., when a national defense agency designs a contingency plan for a nuclear attack, and designing imagined system.
Stumpf and Teague (2005)	"Design is a process which creates descriptions of a newly devised artifact. The product of the design process is a description which is sufficiently complete and detailed to assure that the artifact can be built."	*Coverage* – includes describing an artifact that already exists, e.g. 'the cruise ship is big;' excludes partially designed objects and design of imaginary objects.
Urban Design Group (2006)	"Urban design is the process of shaping the physical setting for life in cities, towns and villages. It is the art of making places."	*Coverage* – This definition confuses design as planning a setting with the physical process of implementing that plan; e.g., by this definition, planning the park is not designing, but laying the sods is.
Walls et al. (1992)	"The design process is analogous to the scientific method in that a design, like a theory, is a set of hypotheses and ultimately can be proven only by construction of the artifact it describes."	*Coverage* – While a design may imply a set of hypotheses, saying the design *is* the like saying being hungry *is* making a sandwich. *Ease of Use* – representing a design as a set of hypotheses may be difficult.

Incomplete by Design and Designing for Incompleteness

Raghu Garud[1], Sanjay Jain[2], and Philipp Tuertscher[3]

[1] Smeal College of Business Pennsylvania State University,
University Park, Pennsylvania USA 16802
[2] College of Business, San Francisco State University,
San Francisco, California USA 94132
[3] E&I Institute for Entrepreneurship and Innovation, Vienna University of
Economics and Business Administration, Vienna, Austria

Abstract. The traditional scientific approach to design extols the virtues of completeness. However, in environments characterized by continual change, there are challenges in adopting such an approach. We examine Linux and Wikipedia as two exemplary cases to explore the nature of design in such a protean world. Our observations highlight a pragmatic approach to design in which incompleteness is harnessed in a generative manner. This suggests a change in the meaning of the word "design" itself – from one that separates the process of design from its outcome, to one that considers design as both the medium and outcome of action.

Keywords: Platform, emergence, design, innovation community.

1 Introduction

Historically, much of the discourse on design has extolled the virtues of completeness. Completeness allows for the pre-specification of a problem, the identification of pre-existing alternatives and the choice of the most optimal solution. Such a scientific approach to design pervades much of management thinking, education and research [1: 24].[1]

For instance, this approach is evident in the design of traditional organizations at the turn of the 19th century. Organizations enhanced the efficiency of their operations by systematically applying principles of scientific management to discover "the one best way" to organize [3]. Interchangeable parts, division of labor, routinization – each of these were features of an organizational design capable of mass producing "any color car as long as it was black" [4: 72].

For such an approach to work, however, there needs to be a clear and stable boundary between the entity being designed and the context for which it is being designed. Such a boundary makes it possible to fix the purpose of a design based on a stable set

---

[1] In this paper, we make a distinction between a "scientific approach" to design that applies analytic thinking to address clearly defined problems to discovery an optimal solution (following the natural sciences as a role model) and a "pragmatic approach" that applies synthetic thinking to address ill-structured problems. Others have used the terms "science" vs. "design" [1] and "decision science" vs. "design science" [2] to make such a distinction.

of user preferences and performance expectations. Clear boundaries, stable preferences and fixed goals – these form the cornerstones of the scientific approach to design as articulated by Simon [5].

But how does such an approach to design hold up in environments characterized by continual change? What if there are multiple designers, each with their own representation of the problem? What if users of a design are also its designers? To further complicate matters, what if the process of discovering new and potentially better states only takes place through a process of participation, and the unfolding of the process itself changes the problem?

This is the new frontier in which we find ourselves. There is no clear separation between the inside and the outside, text and context. Rather, there is only an evolving and emerging network of associations [6: 267]. Problems are ill defined, preferences are fluid and solutions emerge in action. In such situations, an emphasis on completeness is likely to result in the creation of designs that foreclose future options.

It is useful to consider the dual meaning of the word "design" within this context. As a verb, "to design" refers to the process of developing a plan for a product, structure or component. As a noun, "a design" is used to connote the outcome of the process.[2] In traditional settings, these two meanings of design have been separated from one another. One would engage in a process of design (the verb) so as to emerge with a design (the noun) for a specific context. In contemporary settings, however, designs are more appropriately viewed as being simultaneously noun and verb, with every outcome marking the beginning of a new process.[3] Put differently, designs are like dynamic jigsaw puzzles in which multiple actors assemble pieces within templates that change as a result of the actors' engagement.

It is this proposition that we develop in the paper. We suggest that, rather than a scientific approach that tends to separate the two meanings of design, we must embrace a pragmatic approach to design that simultaneously embraces both process and outcome. Given this dual connotation, designs, by definition, have to deal with incompleteness. However, rather than pose a threat, incompleteness acts as a trigger for action. Even as actors try and complete what has been left incomplete, they generate new problems as well as new possibilities that continually drive the design. In this way, incompleteness is both a cause and consequence of the dynamics of organizing in contemporary environments.

We begin by providing a brief overview of the scientific approach to design and then highlight the challenges that one confronts in applying this within contemporary environments characterized by continual change. To empirically locate our observations, we examine two exemplary designs that appear to be always in-the-making – the Linux operating system and the Wikipedia online encyclopedia. We find that, rather than one group designing for another's consumption, designs emerge through situated use as actors co-theorize across multiple settings, and, in the process, create new options. These dynamics produce self-perpetuating processes that further drive continual change.

---

[2] Dewey's [7] approach to pragmatism informed our understanding of design as noun and verb, an understanding reinforced by the entry on design in Wikipedia
(http://en.wikipedia.org/wiki/Design)

[3] This is clearly derived from a structurational perspective [8] where structure is both medium and outcome of action.

2   To Design or Not to Design?

In his book, "The Sciences of the Artificial" [5], Simon suggested that the design of artificial systems – meaning man-made as opposed to natural – is contingent upon the goals of the designer and the purposes for which the system is designed. A key initial task for designers involves the specification of system boundaries. As Simon pointed out, "An artifact can be thought of as a meeting point – an 'interface' in today's terms – between an 'inner' environment, the substance and organization of the artifact itself, and an 'outer' environment, the surroundings in which it operates. If the inner environment is appropriate to the outer environment, or vice versa, the artifact will serve its intended purposes" [5: 6].[4]

Once an interface has been specified and the problem has been defined in terms of its context, form and goals, it is possible to proceed using the organizing principles to be found in the natural world. For instance, Simon [10] offered the principle of 'near decomposability' as a concept possessing clear evolutionary advantages for both natural and artificial systems (see also [11]). Decomposability refers to the partitioning of a system in such a way that the interactions of elements within a subassembly are greater than the interactions between them. Such decomposition reduces the complexities confronted by boundedly rational human beings in their efforts to design artifacts.

To illustrate this point, Simon offered a parable of two watchmakers, Tempus and Hora. Tempus organized his work in a manner that if he had "one (watch) partly assembled and had to put it down – to answer the phone, say – it immediately fell to pieces and had to be reassembled from the elements." Consequently, every time Tempus was interrupted and forced to set aside his work, the entire unfinished assembly fell to pieces. In contrast, Hora first built stable subassemblies that he then put together in a hierarchic fashion into a larger stable assembly. Thus, when Hora was interrupted, only the last unfinished subassembly fell apart, preserving most of his earlier work.

According to Simon, the differential cost of incompleteness that the watchmakers confront is a specific case of a more general challenge that individuals confront when addressing complex problems. The differential cost can be explained by the interplay between the short and long-term memories. When individuals address complex problems, transactions are carried out in their short-term memory. Given the limits to short-term memory, any interruption to a task can exact a toll. This is because any intermediate outcome that might have been accomplished before the interruption is lost. While it is possible to store intermediary outcomes in long-term memory, this too exacts a toll as the transfer between long and short term memory often requires considerable effort.

---

4   Specifying the boundaries is by no means a trivial task. Alexander's insightful analysis [9] of the redesign of a tea kettle illustrates this point. At first blush, such a redesign seems simple given that the kettle is a clearly defined object and the boundaries between the kettle and its environment are obvious. However, Alexander goes on to demonstrate that by changing the nature of the problem to be addressed – for instance, by asking if it is the method of heating kettles rather than the kettle itself that needs redesigning – the solution obtained can change drastically. In reframing the question, the kettle becomes part of the outer environment and the stove becomes the inner environment.

2.1   Scientific Design in Practice

The concept of decomposability that emerges from this parable has generated a lot of attention from scholars who study modularity [12]. Specifically, complex systems can be decomposed in a way similar to what Simon suggested – in other words, into "modules" [13]. Each module only interacts with another through standardized interfaces. As a result, each module becomes a "black-box" [14], possessing the detail required for its functioning, but hiding such detail from other interdependent modules [15].

An implicit assumption in this literature is that the overall system architecture needs to be completely specified *a priori* [12]. In terms of Simon's parable, this means that both Tempus and Hora work with designs that have clearly defined boundaries and pre-set user preferences (accurate time keeping, for example). In such cases, the key design decisions revolve around issues such as detailing the elements that comprise the architecture, establishing stable interface specifications to ensure smooth functioning between modules and "black-boxing" the modules to mask their complexity.

These decisions are integral to a design approach that values completeness. The form and function of a system must be clearly specified. Only with such a complete representation is it possible to identify clear and stable boundaries between self-contained components that mask much of the complexity. These facets are evident in much of the engineering literature with its focus on detailed definition of requirements [16].

In organizational studies, such a scientific approach to design is most evident in the work of Fredrick Taylor and his colleagues [3]. Once the purpose of a corporation was fixed – for example, to produce widgets with certain predetermined features in as efficient a manner as possible – this approach could be used to identify the "one best way" to organize. The design process was driven by the need to completely understand and optimize all the cause and effect relationships that could influence the outcome of an organization's activities.[5] Theorizing was done by specific individuals such as industrial engineers but not by those engaged in ongoing operations whose job it was to "do and not to think". Decomposability was manifest in division of labor and the presence of a hierarchy, as well as the use of interchangeable parts that were tied together through stable interface specifications within an overall organizational architecture. Costs associated with incompleteness that arose from interruptions were to be minimized at all costs by keeping the assembly line running even if there were defects in the system [3]. The result was the design of the quintessential lean and efficient manufacturing process that could mass-produce goods for pre-set user preferences.

2.2   Pragmatic Approach to Design

A scientific approach to design – one that requires complete representation of the problem and identifies the optimal solution – is based on the assumption that the environment is stable. For decades, this assumption held. Contexts within which such designs were deployed changed infrequently, if at all. Consequently, boundaries and

---

[5] See Boland & Collopy [2] for the use of linear programming methods in inventory control, which is based on the maximization of a clearly specified objective function given a set of pre-specified constraints.

preferences could be specified and stabilized, and a design with enduring qualities established, to be tweaked as changes in the environment took place.

However, such an approach is likely to run into problems in environments characterized by continual change [6]. In such contexts, system boundaries are often unclear and user preferences are both heterogeneous and evolving. As a result, the goals and purpose of the design are likely to remain a continually moving target [17].

The advent of new information technologies has made such fluidity possible. Different material artifacts and social groups can now be easily associated in a dynamic network. Rendering the functionality of any product or service through software enables real time changes to a design. Indeed, these technologies make it possible for customers to explore their preferences in use and for different social groups to engage with each other in an emergent fashion.

Simon appreciated the significance of such dynamic situations. For instance, in a section titled "Designing without final goals", Simon explored a paradoxical but pragmatic view of design – to motivate activity that in turn generates new goals. Offering the example of the rebuilding of Pittsburgh, where new goals emerged after initial goals had been accomplished, Simon [5: 163] concluded:

> "Making complex designs that are implemented over a long period of time and continually modified in the course of implementation has much in common with painting in oil. In oil painting, every new spot of pigment laid on the canvas creates some kind of pattern that provides a continuing source of new ideas to the painter. The painting process is a process of cyclical interaction between the painter and canvas in which current goals lead to the new application of paint, while the gradually changing pattern suggests new goals."

In these observations we see how a design approach need not be a static representation of a problem, but can involve a "theory-design-fly-test and start-all-over-again" methodology [18]. Despite this acknowledgement, there has been relatively little work that explores the nature and implications of such a design approach. Indeed, as Boland [19: 109] states, "Much of Simon's concern centers on the local and immediate experience of an individual who faces an environment that is essentially unchangeable. It is a given to which the managers must mold the organization."

The challenges that arise in applying a scientific approach to design in dynamic environments become all the more apparent when we consider the nature of change that is upon us. If we accept Woodward's [20] powerful insight that technologies of work shape the way in which we work with technologies, new information technologies not only link islands of unconnected activities into an action net [21], but, in enabling such connections, change the very meaning of the term "design" to connote continual evolution via interaction. Jelinek [22] fully understood the implications of this change when she stated:

> "A genuine revolution of possibility flowed from the move between prior flat file systems into relational databases and, in computer systems, the hyperlinked nodes of the internet. Similarly, the virtual organization – often pro tem, frequently voluntary, and broadly distributed – is the iconic organization of our times. But how does one design it? Or should one perhaps instead invite it to emerge? Is the issues

> deliberate design, or is it design of a process of interactions that allows a collaborative process to emerge? … How should we theorize about such organizations? … Should we embrace the ephemeral organization as a new norm?"

An image of an organization that is not a series of nested black boxes [23] operating in an immutable environment but, rather, a hyper text organization that continues to emerge is a radical shift indeed. The design problem, then, is not one of developing a static interface (an edge in network terms) that connects the inside and the outside; rather, it involves the creation of multiple edges between many nodes in a dynamic network. In such an action net, each node can potentially act as a boundary object, "remaining between different realms, belonging to all of them simultaneously, and seen from different points of view" [21: 104]. Given this, "When such an organization does emerge, it may be both transient and protean" [22: 115].

These observations further highlight the difficulties associated with designing for completeness in a world that is continually changing. But, what does it mean to design for incompleteness? Is this an oxymoron? We think not. There are now a number of new organizational forms that suggest that incompleteness, rather than pose a threat, can instead be a virtue. We examine two such cases in this paper – the Linux operating system and the Wikipedia online encyclopedia. These cases provide us with an appreciation of the generative nature of incompleteness. They amplify what Weick [24: 43] astutely pointed out, "life persists when designs are underspecified, left incomplete, and retain tension."

3   Research Design

Our objective is to offer a set of observations that form the basis for an ongoing conversation among those interested in understanding the nature of design in continually changing environments. By no means do we claim to offer a full fledged theory – to do so would defeat the very premise of our argument. Here, we subscribe to the notions proposed by Boland and Collopy [2], Romme [1] and others who suggest that the value of theorizing lies in the options that are generated rather than the uncertainties that are resolved. Along these lines, our intent is to sensitize readers to a pragmatic approach to design that harnesses the generative forces of incompleteness.

The research approach that we adopt in this paper involves a detailed exploration of two exemplary cases. Research on analogical thinking [25] suggests that individuals who are presented with multiple cases exhibit greater ease in their abilities to identify underlying patterns. Based on this finding, we decided to provide not one but two in-depth cases. We deliberately chose to examine the Linux operating system and the Wikipedia online encyclopedia as each represents a design that is continually evolving. We tracked available information from a wide variety of online sources as a means of collecting raw data for our case narratives. The multiple data sources helped us "triangulate" [26] in that there were very few disagreements among the data sources on the factual details involved. Two of the authors separately developed the case studies. The individual inferences drawn from each of the cases were discussed and verified with the other authors. Our aim is not to reach a state of theoretical saturation [27; 28]. Rather, much like the phenomena that we are studying, our aim is

to offer a set of observations that will hopefully generate "theoretical tension" and form the basis for ongoing debate and understanding of this phenomenon.

4   Linux: Perpetually in the Making

Linux originated as a hobby project of Linus Torvalds, a computer science student at the University of Helsinki. Initially, he wrote programs to understand how the Minix operating system could use features of the 386 processor. Soon, he had written task-switching programs, a disk driver and a small file system. Torvalds realized that he was actually working on a new operating system, and, as a result Linux 0.01 was born. At this stage, Linux lacked many of the functionalities that users expected from an operating system. Even the numbering of the version – 0.01 – signified its unfinished status [29].

Torvalds initial development efforts can be described as 'discovering design goals in action'. Upon releasing Linux version 0.02, he described the initiative as follows:

> "This is a program for hackers by a hacker. I've enjoyed doing it, and somebody might enjoy looking at it and even modifying it for their own needs. It is still small enough to understand, use and modify, and I'm looking forward to any comments you might have…If your efforts are freely distributable (under copyright or even public domain), I'd like to hear from you, so I can add them to the system….Drop me a line if you are willing to let me use your code." [33]

His announcement to the newsgroup suggests that he was unsure about what others might want in the emergent system. Although such lack of closure could be a problem from a traditional design perspective, it is interesting to observe how Torvalds turned this into a virtue. His decision to allow others to adapt the emergent system to their needs enabled him to harness the energies of the larger programming community. Within two months of his announcement, about thirty people had contributed close to 200 reports of errors and problems using Linux. In addition, these individuals developed new features for the nascent operating system that, in turn, became the basis for future directions along which the system evolved [30].

Traditionally, there has been a clear distinction between the designer and the user. However, Torvalds' actions blurred the boundaries between these two actors – the user could also be the designer (cf. [31]). By catalyzing such a co-creation process, Torvalds ensured that the purpose and functionalities of Linux would now emerge from the efforts of multiple contributors. Providing users with an opportunity to inscribe their local contexts into the design enabled the development and linkage of components in diverse and sometimes non-obvious ways. As these inputs were incorporated, the very purpose and functionality of the platform itself changed.

Technology is society made durable, suggested Latour [32], and in the case of Linux we can recount many instances of how technology made engagement among contributors possible. Clearly, tools such as mailing lists and news groups made it possible for actors to engage with the emergent platform and with one another. Similarly, "installers" that facilitated reuse and distributed modifications contributed to the growing functionality of the system. For example, the Linux community experienced

a large growth in 1992 when the Yggdrasil distribution made it possible for users to install Linux from a CD-Rom [29]. Finally, Torvalds redesigned the Linux kernel to have one common code base that could simultaneously support a separate tree for different hardware architectures, thereby greatly improving its portability [33]. Taken together, these tools made it possible for contributors to continuously extend Linux in a decentralized manner.

Moreover, social rules built into the technology [34] further fostered generative engagement by actors. A key development was Torvalds' decision to release the kernel under the General Public License (GPL), which mandated that any user modifying or adding to the source code would have to make their own contributions available to everyone else [35]. This decision signaled to the community that ownership and control of Linux was not Torvalds alone. Rather, it facilitated the establishment of a stable "generalized exchange system" [36] in which people both contributed to and benefited from the assistance of others. Establishing mechanisms that recognized the contributions of individuals reinforced the feeling of common ownership. A "credits" file made available with every released version of Linux listed various contributors and their roles in developing and maintaining the kernel.

The availability of the source code played a critical part in the actor's generative engagement with the platform. Through documentation in repositories such as the CVS (Concurrent Version System), the source code served as a design trace that provided a memory of how the platform had evolved until a specific point in time. While creating options for the future, contributors could go back to the past to find out how solutions to related problems had been developed. Moreover, the design trace provided the interpretive flexibility [37] required to recontextualize the existing platform and make it work in new application areas without losing coherence with the original platform. Finally, enabling developers to leave their footprint in the source code ensured that their identities became intertwined with the platform and served as a strong attractor for them to contribute.

Overall, these social and technical mechanisms made it easier for contributors to tinker with the system and incorporate their own notions of how it should be further developed. In doing so, they extended Linux in ways that collectively covered a far greater domain than any single individual could have imagined. This manifested itself in the development of a variety of utilities and device drivers that supported specific hardware and peripherals [38]. In an interview, Torvalds said about the community's engagement with the platform:

> "After I had published Linux on the web, other users requested features I had never thought of and more and more new ideas emerged. Instead of a Unix for my own desktop, Linux should now become the best operating system ever. The requests by other people and their patches and help later on made the whole thing more interesting." [39]

Our observations from Linux, then, suggest that incompleteness is generative in two different ways. At one level, incompleteness serves as a trigger for the creation of many diverse ideas on how a design can be extended and further developed. At another level, engagement with such a system both transforms the design as well as creates new avenues for ongoing engagement which, in turn, attracts a new set of contributors who bring into the fold their own contextualized needs, purposes and goals.

The contribution of a graphical user interface, for example, spurred a whole new application area as "Linux on the desktop", a central aspect of current Linux versions [39]. This observation highlights a key benefit of designing for incompleteness – generative engagement with a platform is self-reinforcing in nature, with the boundaries of both its technical and social architecture co-evolving with one another [see also 40]. While generative engagement enables developers to cope with incompleteness in the present, it also is the source of incompleteness in the future.

Such generative engagement by a multitude of users, however, needs to be channeled to maintain coherence. For instance, it is possible for individuals to introduce security holes or malicious code intended to cause damage [41]. But even contributors with the best of intentions can sometimes inadvertently bring about damage to the system. More fundamentally, the lack of control can lead to fragmentation of the platform as factions pursue different avenues of development. As an illustration, the many different distributions of Linux (e.g. RedHat Linux, SUSE, or Debian) all come with proprietary patches when system updates are released. In some cases, there have been so many changes made to the original package that the distributions have become incompatible [42]. Given these challenges, what governance mechanisms can be established to enable incompleteness to be harnessed beneficially?

Linux's governance approach is best described as "centrally facilitated yet organizationally distributed" [43]. A combination of technical and social rules keeps the platform from falling apart. Ongoing documentation of the development process, inscribed into the source code and CVS, serve as a form of cohesive tissue. Chat rooms and bulletin boards act as threads that order interactions among contributors across space and time [44]. These mechanisms are buttressed by an overarching meritocracy within the community in which developers focus on providing new code and modules that add features and functionality to the platform while others, based on their reputation and prior contributions, assume maintenance tasks such as the evaluation of code submitted by developers for inclusion into new releases. Sitting atop this meritocracy is Torvalds who centrally facilitates decisions on strategic issues related to Linux. Decisions to deliberately change the platform are entrusted to a much smaller group of individuals when compared to the total number of contributors. Finally, the provisions of the GPL provide the legal and cultural basis for interaction among community members.

These governance mechanisms facilitate access to knowledge, encourage constant tinkering and curbs private appropriation and free riding. Together, these mechanisms operate with an almost invisible touch vis-à-vis ongoing developmental activity on the platform. This enables extensive experimentation to take place on the system even as it maintains a coherent core. By contrast, a tightly controlled design would bear the risk of falling apart given the difficulties of accommodating the contradictory requirements of a heterogeneous community.

A number of Linux's governance mechanisms have themselves emerged over time in a relatively unstructured fashion. One such convention pertains to the successive release of Linux versions even while distributed development progresses. An even numbered release (for instance, version 2.4) denotes that the release is stable and ready for use, whereas an odd numbered release (for instance, version 2.5) signifies that the release is still being built and under evaluation. The announcement of an odd numbered release serves as the call for new code and modules that drives the current

stable release forward. After a period of development, testing, discussion and reviews, no new additions are allowed so that the entire operating system can be tested for performance and reliability. Once testing has ended and modules of code that compromise reliable performance are removed, the operating system is ready to be released as a stable even-numbered version for widespread use. Such a convention has enabled distributed development.

Overall, our description of Linux highlights what it means to be incomplete by design. To the extent that the system has the capacity to incorporate functionality relatively easily, it is possible to harness its generative properties. Blurring the distinction between users and producers, assuming that preferences are heterogeneous and evolving, and maintaining abstractness in goal definition often facilitates this generative process. These facets lie in contrast to scientific approaches to design that favor clear boundary definitions, fixed user preferences, design closure and platform stability. While systems that are designed for incompleteness may be ugly and messy by conventional design evaluation metrics, they can often outperform traditional designs by being extremely adaptable to continually changing contexts. Adopting such a design approach, then, involves appreciating design as the interplay between intermediary outcomes and processes, with one influencing the other on an ongoing basis.

5   Wikipedia: A Lumpy Work in Progress

Initiated in 2001 by Jimmy Wales and Larry Sanger, Wikipedia's ambitious mission has been to "create and distribute a free encyclopedia of the highest possible quality to every single person on the planet in their own language" [45]. As of 2006, it had become one of the most visited websites in the world, with 5 billion page views monthly. Currently, it is growing at around 6,000 edits a day. What is truly remarkable is that Wikipedia has run as a non-profit organization since 2003 and has five employees in addition to Wales. It meets most of its budget through donations, the bulk of these being contributions of $20 or less [46].

The typical encyclopedia, as represented by the Britannica, is the epitome of a product that is designed to be complete. Its production process involves assembling a large group of experts, working under the direction of a manager, each performing a task on a detailed work chart to produce a work of enormous breadth. This product is then packaged and bound in a set of volumes as an authoritative and accurate source for knowledge. The encyclopedia, then, subscribes to a scientific approach to design, with a clearly defined purpose, a production process that is neatly modularized, clearly delineated boundaries between producers and users, and an emphasis on stability and reliability.

Contrast this with the inner workings of Wikipedia. While the overarching goals of this initiative are similar to that of the Britannica, participation in its development is deliberately inclusive, blurring the boundaries between user and producer. Any registered user can write an article that others subsequently modify and refine. This conceptual shift – along with mechanisms that enable generative engagement – has contributed to a system that has now begun to fundamentally question the ontological basis of a traditional encyclopedia.

Technically, entries are made using a wiki, a tool that allows multiple users to create, edit and hyperlink pages. The word "wiki" comes from the Hawaiian word for "quick", but also stands for "what I know is...", and these definitions, taken together, provide an essence of the design approach underlying Wikipedia. Anyone can initiate a new article by making a basic entry – a "stub" in Wikipedia parlance. If a topic that users are searching for is not covered on the site, then they are encouraged to write it themselves. Knowledge, from this perspective, is assumed to reside in a distributed fashion within the entire online human network.

Why might someone contribute to Wikipedia? For many, its goals – a free encyclopedia that is a fun experience to create, and the open source principles on which it is based -- are an important draw. This translates into a strong level commitment for a sub-set of individuals who end up spending considerable number of hours tending to the project. As a Wikipedian described his attachment to the website "You can create life in there. If you don't know about something, you can start an article, and other people can come and feed it, nurture it." [47]

Given the minimal restrictions on membership, the project has turned fundamental assumptions about the design of knowledge systems on their head. The system is truly democratic in that it does not favor experts over the well-read amateur. As Wales puts it, "To me, the key is getting it right. I don't care if they're a high-school kid or a Harvard professor" [46]. Over 100,000 individuals have made contributions to the website since its inception, with a 24-year-old University of Toronto graduate being the site's premier contributor, having edited more than 72,000 articles since 2001.

How can a system that is based on the participation of literally any individual become a useful knowledge source? On Wikipedia, every article has an open invitation for edits, and the expectation is that most edits will be improvements. Wikipedia operates from the presumption that any individual's knowledge is by definition incomplete, and that ongoing revisions enabled by mass collaboration tools and involving a large group of "eyeballs" will produce a reliable yet continually evolving knowledge repository. More fundamentally, it reflects an appreciation of the inherently accretive nature of knowledge, one in which the content of any article provides the generative basis for the next set of changes. As Earl [48] commented, "The philosophy of Wikipedia is that an article gains validity and maintains currency by continuous widespread updating. Thus, hitherto it has not declared any item finished".

The wiki keeps track of all changes made by users and allows them to compare multiple versions of an article[6]. This information provides a design trace which allows actors to understand how and why an article has emerged over time. Possessing such an overarching temporal perspective is critical to repairing damaged elements in an entry as well as reverting to an older version if the current one is inaccurate. Moreover, this trace often forms the basis for new directions in which the article is developed. The design trace, then, both chronicles and initiates generative engagement with an article. In doing so, it serves as a locus of coordination as well as a point of departure, allowing an article to remain in a state of perpetual change.

In addition, social rules embedded in Wikipedia's website enable further generative engagement. For example, tags such as "The neutrality of this article is disputed"

---

[6] We invite the reader to experiment with the design trace for the Wikipedia entry on "Design" http://en.wikipedia.org/w/index.php?title=Design&action=history.

represent calls for action to improve an article. Disambiguation notices placed at the bottom of certain articles help readers accurately locate information that they are looking for. Finally, the creation of an ontological scheme via categories aids contributors in organizing information available on the site. The categorization of entries itself facilitates a process of adaptive structuration [49] in which the contribution of articles with different content change the meaning of a category that subsequently gets relabeled. These dynamics highlights how the community structures Wikipedia in use.

An interesting by-product of such an evolving knowledge project is its ability to instantaneously respond to current events. When the Indian Ocean tsunami erupted in late 2004, contributors produced several entries on the topic within hours. By contrast, the World Book, who's CD-ROM allows owners to download regular updates, had not updated its tsunami or Indian Ocean entries a full month after the devastation occurred [47]. This ability to provide instant information is a metric on which a knowledge repository organized to change perpetually outperforms a traditional encyclopedia.

The flip side of this design approach, however, is that it produces entries that are often amateurish. While the facts may be sturdy, clarity and concision is sometimes lacking. The initial contributor to an article can set its tone and is not necessarily highly knowledgeable in the area [50]. Disagreements on an article can lead to repeated back-and-forth editing, with the user who spends the most time on the site prevailing in the end. Given the obsession of many users to rack up edits, simple fixes often take priority over more complex edits. Moreover, the open access to the system has produced incidences of vandalism on the site that involve individuals inserting obscenities within entries. Given these shortcomings, what governance mechanisms ensure that articles do not lose their integrity?

Here again, a mix of technical and social elements work together to maintain the integrity of the website. Five webbots continually troll the site, searching for obscenities and evidence of mass deletions and reverting text as they go. Any editing on an article is automatically logged on a "Recent Changes" page that participants can monitor. Anyone who sees something false, biased, or otherwise defective can quickly and easily change the text. Moreover, every article also carries with it a separate discussion section on which debates about what to include on the page are encouraged. Besides this, users employ Internet Relay Chat to discuss ongoing issues, from article details to general policy.

An underlying community of volunteers holds these distributed contributions together. This includes anonymous contributors, people who make a few edits. Next, there are registered users who make edits using their byline. Administrators are individuals who can delete articles, protect pages, and block IP addresses. Finally, there are the super elites who make direct changes to the Wikipedia software and database. There is also a mediation committee and an arbitration committee that rule on disputes. On this front, the inner circle of Wikipedians know each other and value their reputations, which are themselves built bottom-up through their past activity on the site. The consequence of such reputation building has been the creation of a meritocracy with individuals occupying more central positions because of their ongoing involvement when compared to others.

In terms of policies, the founders of Wikipedia have instituted a NPOV (Neutral point of view) rule, which urges contributors to present conventionally acknowledged "facts" in an unbiased way as well as accord space to both sides when arguments

occur [51]. Over time, Wikipedia has instituted additional rules, reflecting the complexities involved in managing a growing decentralized community. For example, a policy preventing certain contentious subject matters from being openly edited by users was ratified recently [52].

This has prompted concern that Wikipedia is becoming a regulatory thicket complete with an elaborate hierarchy of users and policies. Wales, while ambivalent about the growing number of rules and procedures, recognizes them as necessary. According to him, "Things work well when a group of people know each other, and things break down when it's a bunch of random people interacting" [46]. And Earl [48] captured Wikipedia's dilemma as follows:

> "This is a classic knowledge-creation conundrum. On the one hand there is real advantage in assembling collective wisdom; on the other hand there are concerns about validity and about incentives or, more particularly, disincentives for content contributors."

However, taking these steps raises an even more fundamental issue: should Wikipedia abandon its current design approach in order to emulate a traditional encyclopedia? For many, the incomplete nature of an article is valuable in its own right. First, such an article is better than nothing at all. Second, the articles might actually trigger more experienced participants to make contributions that they would otherwise not have made. Instituting governance mechanisms that make it appear more like an encyclopedia could potentially rob Wikipedia of its unique identity and impede generative engagement by contributors. As Carr [53] elaborated:

> "Wikipedia is not an authoritative encyclopedia, and it should stop trying to be one. It's a free-for-all, a rumble-tumble forum where interested people can get together in never-ending, circular conversations and debates about what things mean. Maybe those discussions will resolve themselves into something like the truth. Maybe they won't. Who cares? As soon as you strip away the need to be like an encyclopedia and to be judged like an encyclopedia - as soon as you stop posing as an encyclopedia - you get your freedom back."

This observation underscores the value of incompleteness. Incompleteness allows participants the freedom to engage with the design in search in a way that is meaningful to them. As the quote suggests, to the extent that the goals and metrics of traditional approaches that emphasize completeness are adopted in evaluating designs that are inherently incomplete, there exists a danger of losing the unique value that such designs can provide.

6 Discussion

The Linux and Wikipedia cases affirm what Boisvert (who built upon Deweys' [7] pragmatic approach) pointed out: "Affairs are never frozen, finished, or complete. They form a world characterized by genuine contingency and continual process. A world of affairs is a world of actualities open to a variety of possibilities" [54: 24]. Indeed, these cases provide a deeper appreciation of the title of the paper – incomplete

by design and designing for incompleteness. In an environment that is continually changing, designs that have been completed at a point in time are likely to become incomplete over time. On the other hand, designs that anticipate their incompleteness are likely to remain more up-to-date over time.

6.1   Design Participation within the Pragmatic Approach

The traditional scientific approach employed principles from the natural world to design an artifact with enduring qualities that fulfilled a specific purpose in an unchanging world [5]. From this perspective, design was fixed in time and space; it was opened and modified only to accommodate exogenous environmental changes. Moreover, the locus of design – i.e., the demarcation between designer and user – was clear and unambiguous.

In contemporary environments, however, the distinction between designers and users has blurred, resulting in the formation of a community of co-designers who inscribe their own contexts into the emergent design, thereby extending it on an ongoing basis in diverse and non-obvious ways. Such generative engagement by multiple co-designers is facilitated by numerous socio-technical mechanisms. Tools such as the wiki, licenses such as the GPL, forums such as bulletin boards and the infrastructure provided by the Internet, work with one another to facilitate participation and enable distributed development. This dynamic action net [21], then, contributes to the design remaining in a fluid state.

We can further contrast design participation within the scientific and pragmatic approaches by returning to the parable of the watchmakers offered by Simon. In its original version, user engagement with the design was considered an interruption resulting in watches that remained incomplete and therefore of little value. By contrast, the Linux and Wikipedia cases demonstrate that incompleteness acts as a trigger for generative engagement by co-designers. They are the ones who complete what they perceive is incomplete. They discover the purpose of a design in use. They create avenues for future development that, in turn, attract new groups of co-designers. From a pragmatic design approach, what was considered to be an interruption then now becomes the basis for ongoing change.

6.2   Design Task within the Pragmatic Approach

For co-designers, the design task is very different from the one faced by designers adopting a scientific approach. For the latter, optimization of an objective function given constraints (as in linear programming), represented the dominant approach to designing. By contrast, contemporary designs such as Linux and Wikipedia can be conceptualized as an interlinked set of subjective functions, where one person's subjective function serves as another person's constraints. Complicating matters, the set of interlinked subjective functions is itself underspecified as it emerges in use over time. Under these conditions, the design remains incomplete as the solution to the optimization problem corresponds to more than one point in an n-dimensional space of design parameters. It is for this reason that co-designers must learn to theorize on the fly – i.e., they become reflective practitioners [55] who generate provisional workable solutions to their immediate problems, knowing fully well that the platform that they draw upon and the context to which they apply their solutions will inevitably change.

What is the role of modularity in such an emergent system? Modular elements allow for platform extension as is apparent in both the cases we studied. Under these conditions, extension of the platform takes place through an accretive process where modules are tacked on as "plug-ins" [56]. But, the two cases also demonstrate a less understood, generative facet of incompleteness that is fostered by "partial partitioning". On this front, we suggest that interdependence between partitioned tasks results in situations where changes in one task have a cascading effect on other tasks. Consequently, the system never comes to a rest.

Modularity scholars have suggested that partial partitioning can be a problem [57]. The Linux and Wikipedia cases, however, indicate that there may be benefits to be harnessed from partial partitioning. The "shared division of labor" [58] that emerges allows for redundancy of functions, i.e., certain functions can be fulfilled by more than one component. This enables the system to work even if some components are damaged or left incomplete. More significantly, such redundancy of functions can be generative in that components can be deployed for a different purpose thereby facilitating reconfiguration of the design in response to changes in the environment [59]. It also allows the system to more easily assimilate emergent contributions from co-designers into a continually evolving platform.

6.3 Design Governance within the Pragmatic Approach

Given the distributed, emergent and protean nature of designs, the challenge now becomes one of establishing rules that provide some stability. While an absence of rules is likely to lead to design fragmentation, too many rules can potentially stifle the design. It is this paradox that needs to be managed to preserve the value proposition that such designs offer.

An appropriate form of governance is required to coordinate real time distributed contributions in a way that preserves the design's dynamic qualities – i.e., one which allows elements of a system to inform but not determine one another [6: 267]. Governance mechanisms need to be underspecified [60] or semi-structured [61]; that is, they possess minimum critical specifications [62] to keep the design in a state that is neither too fluid nor too crystallized (cf. [63] for this distinction).

The design trace is a key element that enables such governance. By providing widespread access to knowledge on who contributed what, when and why, the trace makes it easier for actors to understand how a design has emerged over time. Possessing such an overarching temporal perspective is often critical in mitigating design fragmentation. For instance, if a new contribution turns out to be damaged or incompatible, then, the trace makes it possible to simply use an older version. Equally important, the trace can be viewed as a boundary architecture [64] that co-designers draw on to develop extensions to the design. In sum, the ongoing design trace serves as a locus of coordination as well as a point of departure for such designs.

To further elaborate, consider two different logics of engagement described by Bruner [65]. A causal logic, following the role model of the natural sciences, operates when the set of variables and relationships that define the functioning of a design are fully specified (see [1] for a more detailed explication). This logic, associated with a scientific approach to design, leads to the articulation of design rules [1; 12]. A narrative logic, on the other hand, operates on the basis of a narrative's internal coherence

and its external coherence with the listener's existing knowledge. In providing designers with the interpretive flexibility required to generate contextualized solutions and to imagine what might be, the narrative can help coordinate distributed activities across time and space. Such an epistemology connects designs with the emergent purposes of the social groups involved.[7]

The design trace allows for these two different logics to operate simultaneously. The trace possesses a scientific logic that offers co-designers with the raison d'être of the design and the necessary details for it to be functional in real time. From this perspective, each node of a trace is but a module with tags that can be opened up and reused. At the same time, the trace also allows for a narrative logic to operate. Co-designers are motivated to participate with a design because of the flexibility that it offers, and the design in use that emerges is convincing to these participants as the narratives recorded have verisimilitude [65]. A design trace, then, possesses the equivalents of both an ostensive and a performative dimension [66]. By providing connections to assets across time and space, a design trace makes it possible for co-designers to engage in the present by building upon the past in anticipation of a new future [67].

In enabling the two logics to operate simultaneously, the trace is able to alleviate some of the problems associated with human memory and bounded rationality that Simon considered in his parable. Tempus and Hora were boundedly rational individuals, relying on their own short and long term memories. A mechanism such as the design trace could potentially have extended their memories by serving as a collective mind [68]. As a narrative, the trace makes it easier for individuals to recontextualize the design to their own situations. This makes it possible for interruptions to not only be welcomed, but to also serve as the basis for design extension.

In offering these observations we see how future options on a design are generated even as current contributions are incorporated. Moreover, recording changes in the design trace signals to co-designers that their contributions will be honored in the future. If a trace was not to exist, part of the motivation to participate might be lost – why contribute to a commons when the contributions would not be used in the future? At the same time, if the co-designers were over-dependent on the extant trace, then the future design would largely be based on past experiences and become path dependent. For a design (and its trace) to promote diachronic processes, then, anticipations of the future and memories of the past must jointly inform contributions in the present.

7   Conclusion

We have explored the Linux and Wikipedia cases to sketch out the elements of a pragmatic approach to design. In continually changing environments, adopting a design approach that attempts to fix boundaries, goals and purposes is potentially counterproductive. Whereas, such an approach may produce a system that is optimal at a point in time, given continual change, the system is likely to rapidly become obsolete over time. Under these conditions, a pragmatic approach – one that views design as

---

[7] We thank Georges Romme for this insight.

continually evolving and essentially incomplete -- may be more appropriate. Within such an approach, boundaries between designers and users become blurred, heterogeneous user preferences emerge in use, tasks remain partially partitioned and the goals of the design emerge through interaction. Indeed, such an approach harnesses the benefits of incompleteness in comparison to the scientific approach that views incompleteness as a threat.

Eventually, a pragmatic approach involves the fusing together of two meanings of design – that is, as both process and as outcome. Any outcome is but an intermediate step in an ongoing journey, representing both the completion of a process as well as its beginning. Whereas the scientific approach emphasizes the need to crystallize designs, the pragmatic approach highlights the value of retaining fluidity. The essence of this approach is well captured by Hedberg, et al. [69: 43] who noted, "Designs can themselves be conceived as processes – as generators of dynamic sequences of solutions, in which attempted solutions induce new solutions and attempted designs trigger new designs."

References

1. Romme, A.G.L.: Making a Difference: Organization as Design. Organization Science 14, 558–573 (2003)
2. Boland, R.J., Collopy, F. (eds.): Managing as Designing. Stanford University Press, Stanford (2004)
3. Kanigel, R.: The One Best Way: Frederick Winslow Taylor and the Enigma of Efficiency. Viking, New York (1997)
4. Ford, H., Crowther, S.: My Life and Work. Doubleday, Page & Company, Garden City (1922)
5. Simon, H.A.: The Sciences of the Artificial, 3rd edn. MIT Press, Cambridge (1996)
6. Barry, D., Rerup, C.: Going Mobile: Aesthetic Design Considerations from Calder and the Constructivists. Organization Science 17, 262–276 (2006)
7. Dewey, J.: Art as Experience. Minton Balch & Company, New York (1934)
8. Giddens, A.: Central Problems in Social Theory: Action, Structure, and Contradiction in Social Analysis. University of California Press, Berkeley (1979)
9. Alexander, C.: Notes on the Synthesis of Form. Harvard University Press, Cambridge (1964)
10. Simon, H.A.: The Architecture of Complexity. Proceedings of the American Philosophical Society 106, 467–482 (1962)
11. Langlois, R.N.: Modularity in Technology and Organization. Journal of Economic Behavior & Organization 49, 19–37 (2002)

12. Baldwin, C.Y., Clark, K.B.: Design Rules - The Power of Modularity. MIT Press, Cambridge (2000)
13. Langlois, R.N., Robertson, P.L.: Networks and Innovation in a Modular System: Lessons from the Microcomputer and Stereo Component Industries. Research Policy 21, 297–313 (1992)
14. Rosenberg, N.: Inside the Black Box: Technology and Economics. Cambridge University Press, Cambridge (1982)
15. Parnas, D.: On the Criteria to be Used in Decomposing Systems into Modules. Communications of the ACM 15, 1053–1058 (1972)
16. Petroski, H.: Invention by Design: How Engineers Get from Thought to Thing. Harvard University Press, Cambridge (1996)
17. Rindova, V.P., Kotha, S.: Continuous Morphing: Competing through Dynamic Capabilities, Form, and Function. Academy of Management Journal 44, 1263–1280 (2001)
18. Boland, R.J., Jelinek, M., Romme, A.G.L.: Call for Papers: Organization Studies as a Science of Design. Organization Studies (2006)
19. Boland, R.J.: Design in the Punctuation of Management Action. In: Boland, R.J., Collopy, F. (eds.) Managing as Designing, pp. 106–120. Stanford University Press, Stanford (2004)
20. Woodward, J.: Industrial Organization: Theory and Practice. Oxford University Press, New York (1965)
21. Czarniawska, B.: Management as Designing for an Action Net. In: Boland, R.J., Collopy, F. (eds.) Managing as Designing, pp. 102–105. Stanford University Press, Stanford (2004)
22. Jelinek, M.: Managing Design, Designing Management. In: Boland, R.J., Collopy, F. (eds.) Managing as Designing, pp. 113–120. Stanford University Press, Stanford (2004)
23. March, J.G., Simon, H.A.: Organizations. Wiley, New York (1958)
24. Weick, K.E.: Rethinking Organizational Design. In: Boland, R.J., Collopy, F. (eds.) Managing as Designing, pp. 36–53. Stanford University Press, Stanford (2004)
25. Loewenstein, J., Thompson, L., Gentner, D.: Analogical Encoding Facilitates Knowledge Transfer in Negotiation. Psychonomic Bulletin and Review 6, 586–597 (1999)
26. Jick, T.D.: Mixing Qualitative and Quantitative Methods: Triangulation in Action. Administrative Science Quarterly 24, 602–611 (1979)
27. Eisenhardt, K.M.: Building Theories from Case Study Research. Academy of Management Review 14, 532–550 (1989)
28. Glaser, B.G., Strauss, A.L.: The Discovery of Grounded Theory: The Strategies for Qualitative Research. Aldine de Gruyter, Hawthorne (1967)
29. Diedrich, O.: Happy Birthday, Tux! c't, p. 162 (2001)
30. Moon, J.Y., Sproull, L.: Essence of Distributed Work: The Case of the Linux Kernel. First Monday 5 (2000)
31. von Hippel, E., von Krogh, G.: Open Source Software and the Private-Collective Innovation Model: Issues for Organization Science. Organization Science 14, 209–223 (2003)
32. Latour, B.: Technology is Society Made Durable. In: Law, J. (ed.) A Sociology of Monsters: Essays on Power, Technology and Domination, pp. 103–131. Routledge, London (1991)
33. Torvalds, L.: The Linux Edge. In: DiBona, C., Ockman, S., Stone, M. (eds.) Open Sources: Voices from the Open Source Revolution, pp. 101–111. O'Reilly & Associates, Sebastopol (1999)
34. Lessig, L.: Code and Other Laws of Cyberspace. Basic Books, New York (1999)
35. Stallman, R.: The GNU Operating System and the Free Software Movement. In: DiBona, C., Ockman, S., Stone, M. (eds.) Open Sources: Voices from the Open Source Revolution, pp. 53–70. O'Reilly & Associates, Sebastopol (1999)

36. Kollock, P.: The Economies of Online Cooperation: Gifts and Public Goods in Cyberspace. In: Smith, M.A., Kollock, P. (eds.) Communities in Cyberspace, pp. 220–239. Routledge, London (1999)
37. Pinch, T.J., Bijker, W.E.: The Social Construction of Facts and Artefacts: or How the Sociology of Science and the Sociology of Technology might Benefit from Each Other. Social Studies of Science 14, 399–441 (1984)
38. Thiel, W.: Keyboard.S with German Keyboard Linux-Activists (1991), `ftp://tsx-11.mit.edu/pub/linux`
39. Diedrich, O.: Was Anwender tun, ist niemals falsch. c't, p. 90 (2000)
40. Neff, G., Stark, D.: Permanently Beta: Responsive Organization in the Internet Era. In: Howard, P.E.N., Jones, S. (eds.) The Internet and American Life. Sage, Thousand Oaks (2003)
41. Singer, M.: Linux Creator: A Little Fragmentation is Good Inernetnews.com (2005), `http://www.internetnews.com/dev-news/article.php/3467241`
42. Wolf, C.: The ROCK Linux Philosophy linux devcenter.com (2001)
43. Garud, R., Kumaraswamy, A.: Vicious and Virtuous Circles in the Management of Knowledge: The Case of Infosys Technologies. MIS Quarterly 29, 9–33 (2005)
44. Lanzara, G.F., Morner, M.: Artefacts Rule! How Organizing Happens in Open Source Software Projects. In: Czarniawska, B., Hernes, T. (eds.) Actor-Network-Theory and Organizing, pp. 67–90. Copenhagen Business School Press, Copenhagen (2005)
45. Wales, J.: Wikipedia is an Encyclopedia (2005), `http://mail.wikipedia.org/pipermail/wikipedia-l/2005-March/038102.html`
46. Schiff, S.: Know It All The New Yorker (2006)
47. Pink, D.H.: The Book Stops Here Wired (2005)
48. Earl, M.: Wikipedia's Struggle to Govern a Knowledge Democracy. The Financial Times, London (2005)
49. DeSanctis, G., Poole, M.S.: Capturing the Complexity in Advanced Technology in Use: Adaptive Structuration Theory. Organization Science 5, 121–147 (1994)
50. Wattenberg, M., Viegas, F.B.: The Hive Mind Ain't What It Used to Be Edge (2006), `http://www.edge.org/discourse/digital_maoism.html`
51. Poe, M.: The Hive The Atlantic Monthly, pp. 86–96 (2006)
52. FinancialWire: Wikipedia To Change Revision Policy, Forest Hills, MD (June 19, 2006)
53. Carr, N.: Let Wikipedia be Wikipedia (2005), `http://www.roughtype.com/archives/2005/12/let_wikipedia_b.php`
54. Boisvert, R.D.: John Dewey: Rethinking Our Time. State University of New York Press, Albany (1998)
55. Schon, D.A.: The Reflective Practitioner: How Professionals Think in Action. Basic Books, New York (1983)
56. Garud, R., Jain, S., Phelps, C.: Technological linkages & transience in network fields: New competitive realities. In: Baum, J. (ed.) Advances in Strategic Management, pp. 205–237. JAI Press, Greenwich (1998)
57. Ethiraj, S.K., Levinthal, D.: Modularity and Innovation in Complex Systems. Management Science 50, 159–173 (2004)
58. Nonaka, I., Takeuchi, H.: The Knowledge-Creating Company. Oxford University Press, New York (1995)
59. Garud, R., Kotha, S.: Using the Brain as a Metaphor to Model Flexible Production Systems. Academy of Management Review 19, 671–698 (1994)

60. Weick, K.E., Sutcliffe, K.M., Obstfeld, D.: Organizing for High Reliability: Processes of Collective Mindfulness. In: Sutton, R.I., Staw, B.M. (eds.) Research in Organizational Behavior, pp. 81–123. JAI Press, Stamford (1999)
61. Brown, S.L., Eisenhardt, K.M.: The Art of Continuous Change: Linking Complexity Theory and Time-Paced Evolution in Relentlessly Shifting Organizations. Administrative Science Quarterly 42, 1–34 (1997)
62. Emery, F.: Designing Socio-Technical Systems for 'Greenfield' Sites. Journal of Occupational Behaviour 1, 19–27 (1980)
63. Gehry, F.O.: Reflections on Designing and Architectural Practice. In: Boland, R.J., Collopy, F. (eds.) Managing as Designing, pp. 19–35. Stanford University Press, Stanford (2004)
64. Star, S.L., Griesemer, J.R.: Institutional Ecology, 'Translations' and Boundary Objects: Amateurs and Professionals in Berkeley's Museum of Vertebrate Zoology, 1907-39. Social Studies of Science 19, 387–420 (1989)
65. Bruner, J.S.: Actual Minds, Possible Words. Harvard University Press, Cambridge (1986)
66. Feldman, M.S., Pentland, B.T.: Reconceptualizing Organizational Routines as a Source of Flexibility and Change. Administrative Science Quarterly 48, 94–118 (2003)
67. Ricoeur, P.: Time and Narrative. University of Chicago Press, Chicago (1984)
68. Weick, K.E., Roberts, K.H.: Collective Mind in Organizations: Heedful Interrelating on Flight Decks. Administrative Science Quarterly 38, 357–381 (1993)
69. Hedberg, B.L., Nystrom, P.C., Starbuck, W.H.: Camping on Seesaws: Prescriptions for a Self-designing Organization. Administrative Science Quarterly 21, 41–65 (1976)

Challenges in Requirements Engineering: A Research Agenda for Conceptual Modeling

Salvatore T. March[1] and Gove N. Allen[2]

[1] Owen Graduate School of Management, Vanderbilt University,
401 21st Ave South, Nashville, TN 37203-2422
`sal.march@owen.vanderbilt.edu`
[2] A. B. Freeman School of Business, Tulane University,
New Orleans, LA 70118
`gallen@tulane.edu`

Abstract. Domains for which information systems are developed deal primarily with social constructions—conceptual objects and attributes created by human intentions and for human purposes. Information systems play an active role in these domains. They document the creation of new conceptual objects, record and ascribe values to their attributes, initiate actions within the domain, track activities performed, and infer conclusions based on the application of rules that govern how the domain is affected when socially-defined and identified causal events occur. Emerging applications of information technologies evaluate such business rules, learn from experience, and adapt to changes in the domain. Conceptual modeling grammars aimed at representing their system requirements must include conceptual objects, socially-defined events, and the rules pertaining to them. We identify challenges to conceptual modeling research and pose an ontology of the artificial as a step toward meeting them.

1 Introduction

Conceptual modeling is fundamental to information systems requirements engineering. Systems analysts and designers use the constructs and methods of a conceptual modeling formalism to represent, communicate and validate the contents, capabilities and constraints of an envisioned information system within its organizational context. The value of such a representation is measured by the degree to which it facilitates a shared understanding among all stakeholders of (1) the organizational information requirements and (2) the ability of the envisioned information system to meet them [1].

The philosophical discipline of ontology provides a substantive basis for such a shared understanding [2]. A number of researchers have proposed using ontology as the basis upon which to develop and evaluate conceptual modeling grammars (constructs and rules) and methods [1, 3-5]. In general, ontology seeks a definitive and exhaustive classification of entities in all spheres of being. Applied ontology is the attempt to use the rigorous tools of philosophical ontology in the development of category systems which can be of use in the formalization and systemization of

knowledge in a given domain [6, 7]. In information systems requirements engineering the premise is that an ontology can explicitly define the basic concepts used by all stakeholders in the description of the business domain and the envisioned information system artifact. This shared conceptualization enables stakeholders to effectively communicate about the objectives of the business and the capabilities of the information system to meet them resulting in more effective information system designs and implementations.

We observe that an ontology is itself an artifact, a set of constructs developed by human intelligence for the purpose of representing a specific domain [8]. The domain itself determines the types of constructs needed as well as the premises and implications of the ontology. Within the scope of conceptual modeling in information systems the domain of interest is the world of business organizations engaged in the production and exchange of goods and services in social, political, and legal economies. It is primarily a socially constructed world [9] rather than a material world [10]. It is a designed world rather than a natural world. It requires an ontological underpinning that is based on a design science [11] rather than a natural science [4] view of the domain.

The implications of this observation for conceptual modeling constructs and methods are significant. First, the fundamental conceptualization of the task is significantly different. A natural science (material world) conceptualization views the task as the discovery of material objects, their properties, and the natural, immutable laws that govern their behavior [10]. A design science (socially constructed world) conceptualization views the task as the creation of conceptual objects, the ascription of attributes to concrete and conceptual objects, and the development of policies regulations that describe "agreed-upon" behavior [9] and achieve organizational purposes [11]. While systems analysts must in some sense "discover" legal, social and organizational policies and regulations there is frequently significant latitude in designing organizational policies. In the terminology of management science and operations research legal and social regulations may need to be taken as constraints, but organizational policies should be taken as decision variables [11].

Second, the conceptualization of the role of an information system within the domain is significantly different. A natural science conceptualization views the role of an information system as a passive state-tracking mechanism [4, 12]. A design science conceptualization views the role of an information system as actively participating in the work system of the organization [13].

Third, the conceptualization of causality is significantly different. A natural science conceptualization views causality with respect to rules that govern changes in the state of material objects--the chair moved from location A to location B *because* force C was applied to it. A design science conceptualization views causality with respect to intentions and goals--the chair was moved from location A to location B *because* we sold it to company D at a price that maximized profits. Concepts such as ownership, contracts, agreements and transactions are outside the scope of an ontology of the material world. They are socially constructed (artificial) objects that exist only because we agree that they exist [9, 14].

To address the need for a substantive ontological basis for conceptual modeling we pose an ontology of the artificial. We use the ontology of Mario Bunge [10] as a preliminary background and develop six premises upon which to base the ontology. We

acknowledge that this ontology is preliminary and empirically untested. However, we argue that it contains constructs necessary to adequately represent the requirements for information systems that play an active role in business domains [13].

Such information systems document the creation of new conceptual objects, record and ascribe values to their attributes, make decisions and initiate actions within the domain, track activities performed, and infer conclusions based on the application of rules that govern how the domain is affected when socially-defined events occur. Emerging applications of information technologies are taking an even more active role in business domains. They are expected to evaluate business-dependent rules, learn from experience, and adapt to changes in the environment [15]. Conceptual modeling grammars must be capable of representing the conceptual objects, socially-defined events, and rules that characterize their design requirements. Furthermore, they must be capable of representing the purpose of an information system within its organizational context, including goals, decision alternatives, constraints, and evaluation criteria [11].

2   Background

The ontological work most frequently cited in the conceptual modeling literature is that of Mario Bunge. Bunge [10] proposed a scientific ontology that distinguishes two types of objects: concrete and conceptual. The existence of concrete objects is observer-independent. They exist and possess substantial properties "even if we are ignorant of this fact" (p. 58). Conversely, the existence of conceptual objects depends solely upon human invention. They are "creations of the human mind" (p. 116) that "exist as long as there are rational beings capable of thinking them up." Examples of concrete objects include people, buildings, machines, trees, minerals, animals, and electrons. Examples of conceptual objects include numbers, mathematical theories, intellectual property, legally created corporations, contracts and agreements, and the legal institution of a marriage. Bunge's ontology specifically focuses on concrete objects, yet conceptual objects form the heart of business domains. If conceptual modeling is to have an ontological foundation, it must be an "*ontology of the artificial*." One that is intended to represent the objects and attributes created by human intention, for human purposes, and imbued with meaning that is shared among participants.

Existing work in natural ontology should not be ignored in the development of an ontology of the artificial. Specifically, Bunge [10] provides a sound representation of concrete objects and their properties. However, conceptual objects are "lawless." It may be agreed that they will follow invented rules within a specific context; however those rules can be changed as purposes within the context change [16]. Furthermore, such rules are enforced within different contexts. Some are enforced within a governmental context, others within the organizational context. In either case they are outside of Bunge's ontology but they are fundamental to many business domains.

3   Premises

Business organizations use concrete objects such as people, machines, and buildings to accomplish their goals. However, they are concerned primarily with the meaning

and purpose ascribed to concrete and conceptual objects and with invented rules that *form the basis of social intercourse*. A person is important to a business primarily because of the meaning and purpose ascribed to them (e.g., customer, employee). A chair is important to one organization because it is a product that they sell; it is important to another organization because it is used by an employee in the performance of their work tasks and contributes to the assets of the organization.

Premise 1. Conceptual modeling is primarily concerned with representing social phenomena.

Natural phenomena [10] result from processes that are governed by immutable laws. It is the role of science to discover and understand such laws. Social phenomena [9, 17] result from intentional human actions. When deemed important to do so, it is the role of a society, by collective agreement, to invent and enforce rules that govern social phenomena. Such rules frequently result in the creation of conceptual objects (e.g., social contracts and legal entities) that must be documented and validated by social means—there are no corresponding concrete (physical) objects. A major purpose of an information system is to provide such documentation and validation. A sales order, for example, is a conceptual object created when a buyer and a seller agree upon the terms of the sale. It follows the rules of commerce established by governments, regulatory agencies, and specific business organizations. Its existence is documented in an information system (manual or automated).

Another purpose of an information system is to execute the rules invented to accommodate the occurrences of defined social phenomena. We term such social phenomena *conceptual events*.

Premise 2. Defined social phenomena, termed conceptual events, have rules that specify the actions to be taken when they occur.

Conceptual events are conceptual objects. The rules associated with them are invented and changeable. The specified actions frequently involve the creation of conceptual objects and the ascription of attributes to conceptual and concrete objects. A "sale," for example, is a conceptual event resulting in the creation of a sales order. The recognition of a "sale" event is by a social agreement between the buyer and the seller governed by regulations imposed within a larger social system. Such conceptual events and the rules governing them are an integral part of a conceptual model.

We must differentiate this notion of *conceptual event* from the event construct defined by Bunge [10] and used in prior conceptual modeling research [e.g., 4]. In that work an *event* is defined as a change in the *state* of a concrete object, the change being governed by a law, which describes the natural processes effecting the change. Neither the occurrence that initiated the change (intention) nor the purpose (meaning) of the change is considered. A *conceptual event* includes elements of purpose, intention, and rules governing the commitments and obligations (conceptual objects) created by the participants when it occurs [18]. An important role of information systems is to process conceptual events, documenting the agreed upon commitments and obligations incurred.

Premise 3. An information system is primarily an event-processing mechanism utilizing event rules to make decisions, create conceptual objects, and ascribe attributes to concrete and conceptual objects.

A business may declare, for example, that a "person becomes a customer" when that person initiates a "registers at web site" event. The person becomes a customer when he or she fills out the registration form and presses "submit." The rules may specify that the person who initiated the event is to be ascribed: (1) the attribute "authorized for sales transactions" and (2) the set of attributes common to all customers. This set of attributes may include those associated with the data entered, e.g., name and address as well as attributes whose values are generated according to the event's rules, e.g., customer number, customer status (initialized at "provisionary"), and credit limit (initialized to $1000). Furthermore the "registers at web site" event, a conceptual object, is related to the customer object. The values of its attributes do not change even if the values of the customer's attributes are changed by subsequent events [13].

We term such event-processing information systems "active" because they actively participate in the business domain rather then passively record facts from it. The rules for some events may be unknown or outside the scope of the information system. For such events the modeler may choose not to represent the event in the conceptual model but only to represent attributes of the affected objects. The role of the information system is said to be "passive" with respect to that event, being limited to recording attribute values and histories.

Premise 4. The scope of a conceptual event rule is the set of social institutions with the authority to change and enforce it.

The scope of a conceptual event rule represents the degree of control an organization has over its definition, enforcement, and modification. Some of the rules that must be represented in business domains are constraints within their legal environment. These are enforced by governmental agencies. Their violation can result in legal action, fines, and even termination of the business. Such rules include fair trade practices, Sarbanes Oxley, and minimum wage. Changing them requires the act of a socially authorized governing body (e.g., a legislative or judicial act). Other rules represent policies of the business. They are defined, enforced, and changed by the organization itself. Their violation may have financial consequences but are discretionary within the broader social context in which the business operates. Such rules include policies for product pricing, warrantees and returns, employee compensation and incentives, quality control, vendor selection, and technology use. Organizations frequently define event rules to assure compliance with rules imposed by the environment. Policies defining separation of accounting duties with respect to authorization, recording, and custody over organizational assets, for example, assure compliance with the control requirements of Sarbanes-Oxley.

Premise 5. The range of a conceptual event rule is the set of objects that are affected by the rule.

The range of a conceptual event rule represents the breadth of the rule's influence. The range of a policy implemented by a corporation about hourly compensation is the set of hourly employees of the company; the range of a state law on minimum wage is the state's set of minimum wage workers and the companies for which they work.

Premise 6. Identification of objects (conceptual and concrete) is fundamental to the conceptual modeling of business domains.

Distinguishing objects is fundamental to language [17] and hence to communication and exchange within business domains. Business organizations must often differentiate individual objects such as employees, customers, contracts, sales, sales orders, and shipments as well as types of objects such as raw material and product categories, e.g., at the stock keeping unit (SKU) level rather than the individual (serialized) level. They frequently rely on artificial means such as ascribed identifier attributes to distinguish such objects. The representation of such ascribed attributes is fundamental to the purpose of a conceptual model—conveying a shared understanding of the phenomena within the domain [19].

4   Ontology of the Artificial for Conceptual Modeling

One implication of the above premises is that information system requirements are designed not discovered. While a conceptual modeler must analyze the domain to identify the important objects and rules, these must be understood in the context of design—design of the information system, design of the business, and design of the social context in which they operate. All are artificial systems [11] upon which the organization can exert varying degrees of influence. Based on the above premises we propose the following as a rudimentary ontology of the artificial.

1. There are two types of objects—concrete and conceptual.

   1a. *Concrete objects* exist physically.

   1b. *Conceptual objects* exist by human intention and social agreement.

2. Attributes are ascribed to concrete and conceptual objects for human purposes. Attributes map functionally from objects to values.

3. There are two types of attributes: substantial and invented.

   3a. *Substantial attributes* are ascribed to concrete objects to represent human understanding of natural phenomena (in Bunge's terminology [10] these represent substantial properties of concrete objects).

   3b. *Invented attributes* are ascribed to concrete and conceptual objects to enable social intercourse (in Searle's terminology [14] these represent institutional facts).

   3c. One purpose for the ascription of invented attributes to objects is their *identification* (individuation).

4. Objects may be grouped into *types* (classes, categories) based on the ascription of one or more common attributes. Frequently these attributes represent the purpose or role of the object within an organization (e.g., customer, vendor, partner, employee, owner, product, raw material, purchase order, sales order).

   4a. An object may be grouped into multiple types.

   4b. Types may exist in hierarchical or networked relationships with other types. Subtypes inherit attributes from their supertypes.

5. The *status* of an object is the set of values of its attributes at a point in time. The *history* of an object is the chronology of its status.

6. There are two types of events, concrete and conceptual.

6a. *Concrete events* are changes to substantial properties of concrete objects. They follow immutable, natural, discoverable laws (information systems dealing with industrial processes or natural phenomena such as meteorology or volcanic activity are likely to require the representation of concrete events and the laws that govern them). They have neither identity nor meaning.

6b. *Conceptual events* affect changes to invented attributes of concrete and conceptual objects. They have purpose (intention) and follow rules that are designed and defined by human agreement. These rules are mutable, of varying scopes and ranges, subject to evaluation, and can be intentionally violated (information systems dealing with transactions and human agreements such as enterprise resource systems, customer relationship management systems, and supply chain management systems are likely to require the representation of conceptual events and the rules they are required to follow).

6c. Concrete events exist in a causal sequences (natural processes) and can be initiated by a conceptual event; however, concrete events do not have purpose or meaning. Purpose and meaning are associated only with conceptual events.

6d. Objects that are affected by the same event are said to be in *relationship* with the event and, hence, in relationship to each other through the event.

7. Objects may *compose* and *decompose* to form other objects.

8. Events may *compose* and *decompose* to form other events.

While we recognize the above ontology is cursory we believe it is useful for the purpose of developing a research agenda in conceptual modeling—the construction and evaluation of an ontology suited to the representation of the social phenomena. Such an ontology must provide a shared conceptualization that enables the characterization of business domains and to the development of information systems that actively participate in their operation and management. One implication of the proposed ontology is that relationships are not viewed a "mutual properties" as proposed in prior research [4, 20]. Rather relationships (other than subtypes and compositions) are posed as resulting from events that occur. We conjecture that the explicit representation of causal events and their associated concrete and conceptual objects will result in a more effective representation than will the representation of relationships as mutual properties. This conjecture must be tested empirically.

5   Research Challenges

A number of research challenges remain. These involve the development and evaluation of constructs, methods, and symbols by which the proposed ontology can be effectively applied in conceptual modeling. First, constructs are needed to represent concrete and conceptual objects, events and rules (laws). It is not clear if different symbols should be used for all or some of these constructs. We have proposed the use of a single conceptual modeling construct, entity-type, for the representation of

concrete and conceptual objects as well as concrete and conceptual events and the use of attributes of event entity types for the representation of event rules. Furthermore event rules are likely to have exceptions within a subtype structure. Therefore a mechanism for categorizing events may also be necessary. Second, the ontology must be demonstrated to be effective in enabling the conceptual modeling of business domains and must be demonstrated to lead to effective designs and implementations. The parsimony and understandability of conceptual models built using it must be assessed and methods to guide their construction and evaluation must be developed.

Third, the applicability and value of this ontology in active conceptual modeling must be assessed. Active conceptual modeling [15] focuses on enhancing our understanding of how to model systems that learn from past experiences. It requires conceptualizations that facilitate the analysis and re-analysis of social phenomena to generate alternate or proposed histories and conclusions. The representation of events as conceptual objects and the conceptualization of an information system as an active, event-processing mechanism provide the basis for this understanding. They also provide a framework for the representation of stories and narrative [21] and for the representation of episodic and semantic memory [22], each a significant component of human information processing and sense-making. Future research should investigate how the proposed ontology can be used to develop intelligent learning-based applications in areas such as business intelligence, global situation monitoring, surveillance, reconnaissance, design, decision-making and decision-evaluation.

Finally, as we observed earlier, an ontology is itself an artifact, developed by human intention for specific purposes. Within the scope of conceptual modeling the purpose of ontology is to enable the development of effective information systems. We have proposed the beginnings of an ontology of the artificial to address this problem.

References

1. Wand, Y., Weber, R.: Research Commentary: Information Systems and Conceptual Modeling–A Research Agenda. Information Systems Research 13, 363–376 (2002)
2. Gruninger, M., Lee, J.: Ontology applications and design. Communications of the ACM 45, 39–41 (2002)
3. Burton-Jones, A., Meso, P.: Conceptualizing Systems for Understanding: An Empirical Test of Decomposition Principles in Object-Oriented Analysis. Information Systems Research 17, 38–60 (2006)
4. Wand, Y., Weber, R.: On the Deep Structure of Information Systems. Information Systems Journal 5, 203–223 (1995)
5. Gemino, A., Wand, Y.: Complexity and clarity in conceptual modeling: Comparison of mandatory and optional properties. Data & Knowledge Engineering 55, 301–326 (2005)
6. Gruber, T.: Toward principles for the design of ontologies used for knowledge sharing. International Journal of Human Computer Studies 43, 907–928 (1995)
7. Guarino, N., Welty, C.: Evaluating Ontological Decisions with OntoClean. Communications of the ACM 45, 61–65 (2002)
8. Allen, G.N., March, S.T.: A Critical Assessment of the Bunge-Wand-Weber Ontology for Conceptual Modeling. In: Proceedings of the Sixteenth Annual Workshop on Information Technologies and Systems, Milwaukee, WI, pp. 25–30 (2006)

9. Searle, J.: The Construction of Social Reality. Free Press, New York (1995)
10. Bunge, M.: Ontology I: The Furniture of the World. Reidel Publishing Company, Boston (1977)
11. Simon, H.: The Sciences of the Artificial. MIT Press, Cambridge (1996)
12. Wand, Y., Weber, R.: An ontological model of an information system. IEEE Transactions on Software Engineering 16, 1282–1292 (1990)
13. March, S., Allen, G.: Ontological Foundations for Active Information Systems. International Journal of Intelligent Information Technologies 3, 1–13 (2007)
14. Searle, J.: Social ontology: Some basic principles. Anthropological Theory 6, 12–29 (2006)
15. Chen, P., Wong, L.: A Proposed Preliminary Framework for Conceptual Modeling of Learning from Surprises. In: Proceedings of the International Conference on Artificial Intelligence, Las Vegas, NV, pp. 905–910 (2005)
16. Brooks, F.: No Silver Bullet: Essence and Accidents of Software Engineering. IEEE Computer 20, 10–19 (1987)
17. Davidson, D.: Essays on Action and Events. Oxford University Press, Oxford (1980)
18. Geerts, G., McCarthy, W.: An ontological analysis of the economic primitives of the extended-REA enterprise information architecture. International Journal of Accounting Information Systems 3, 1–16 (2002)
19. Kent, W.: Data and Reality: Basic Assumptions in Data Processing Reconsidered. Elsevier Science Inc., New York (1978)
20. Wand, Y., Storey, V., Weber, R.: An Ontological Analysis of the Relationship Construct in Conceptual Modeling. ACM Transactions on Database Systems 24, 494–528 (1999)
21. Robinson, J., Hawpe, L.: Narrative Thinking as a Heuristic Process. In: Sarbin, T.R. (ed.) Narrative Psychology: The Storied Nature of Human Conduct. Praeger Publishers, New York (1986)
22. Tulving, E.: Elements of Episodic Memory. Oxford University Press, New York (1983)

Section 2: Evolution and the Fluidity of Design

John Mylopoulos

Evolution is a fact of life. Environments and the species that operate within them -- living, artificial, or virtual -- evolve. Evolution has been credited with the most advanced biological species on earth. The ability to evolve has also come to be treated as a prerequisite for the survival of a species. Yet, evolution of the software systems species has been studied only at the level of code and design, but not at the level of requirements. In particular, there has been considerable research on software evolution, focusing on code re-engineering and migration, architectural evolution, software re-factoring, data migration and integration. However, the problem of post-deployment evolution of requirements (as opposed to architecture, design and/or code) has not entered into the research discourse.

There are several important reasons why requirements evolution is likely to become a focal point for research activity in Software Engineering in years to come. The change from local, isolated communities to the global village is not happening only for commerce, news and the environment. It is also happening for software. In the past, operational environments for software systems were stable, changes were local, and evolution was local. Today, the operational environment of a large number of software systems is global, open, partially unknown and unpredictable. In this context, software systems must evolve in order to cope ("survive" is the technical term for other species). To be sure, some of this evolution will happen at the code level, and some at the architectural level. The most important evolution, however, will take place at the level of requirements, as to ensure that a system continues to meet the needs of its stakeholders and conform to its constraints -- economic, legal and otherwise -- placed upon its operational environment.

Despite a relatively sparse landscape for research on requirements evolution, some things have become clear in the last years. System architectures play a pivotal role during system evolution, by offering a foundation of (relative) stability in the midst of wholesale change. Alistair Sutcliffe examines precisely this role of architectures in his chapter, titled "On the Inevitable Intertwining of Requirements and Architecture". His study includes the role of requirements and architectures in the context of Michael Jackson's Problem Frames, as well as his own Domain Theory Generic Object System framework. The two frameworks are compared using a case study involving configurable software that supports medical researchers in e-Science applications.

In their chapter, Neil Ernst et al. first review the relevant literature on requirements evolution and then propose a framework for monitoring and diagnosing software-intensive systems that exploit requirements models to determine what is to be monitored and how are monitoring data to be interpreted in order to detect and identify system failures. More specifically, the framework assumes that along with code, a running software-intensive system has access to a requirements model with traceability links between requirements and the code that implements them. The chapter is titled "Requirements Evolution and What (Research) to Do About It."

Bill Robinson and Steve Fickas are among a handful of researchers who have been conducting research for some time on requirements evolution and monitoring. In their

chapter, titled "Designs Can Talk: A Case of Feedback for Design Evolution in Assistive Technology," they present a case study involving the design of a specialized e-mail system for cognitively impaired patients. The case study details the evolution of requirements for the system as each user improved her e-mail skills and made progress in their personal objectives. Their study highlights the importance of continuous feedback while a software system is used, as a means for guiding its improvement and evolution through requirements detection.

On the Inevitable Intertwining of Requirements and Architecture

Alistair Sutcliffe

Manchester Business School, University of Manchester, Booth Street West,
Manchester, M15 6PB, UK
`ags@manchester.ac.uk`

Abstract. The chapter investigates the relationship between architecture and requirements, arguing that architectural issues need to be addressed early in the RE process. Three trends are driving architectural implications for RE: the growth of intelligent, context-aware and adaptable systems. First the relationship between architecture and requirements is considered from a theoretical viewpoint of problem frames and abstract conceptual models. The relationships between architectural decisions and non-functional requirements is reviewed, and then the impact of architecture on the RE process is assessed using a case study of developing configurable, semi-intelligent software to support medical researchers in e-science domains.

Keywords: Requirements process, architecture, adaptable systems.

1 Introduction

In this chapter I will investigate the association between requirements and system architecture, taking my cue from the seminal paper by Swartout and Balzer [1] who pointed at that specification and design are closely interlinked activities rather than being separated in a sequential "waterfall" process. Furthermore, I will argue that the nature of requirements engineering will fundamentally change in the near future as software applications become more intelligent, ubiquitous and mobile. In this case requirements concern systems more directly than users; in other words, architectural requirements are necessary to deliver user requirements. This is a consequence of the evolving relationship between the real world and the designed system: as the software moves in the world and knows more about the world, so it can adapt and react to it. This challenges conventional concepts of requirements as known entities which could be specified and implemented in advance of use. Architecture requirements are necessary to specify how the machine monitors, interprets and adapts to the world, even though these do not directly concern the user who simply wants an adapted, customised or location-aware service.

The changing nature of requirements has also been acknowledged in the context of design exploration, where users only "know what they want when they get it" [2, 3]. The co-evolution of software in response to shifts in the environment motivated Lehman's models of evolving software [4]; while the need to adapt changing requirements was proposed by Fickas and Feather [5] for evolving systems to fit with

changing worlds, which implied requirements for monitoring and interpreting functions. Fischer's distinction between adaptable and adaptive systems [6] also implies requirements for configuration facilities and intelligent adaptive processes. In adaptable systems the boundary can be set by the designer to provide a range of configurable solutions. In adaptive systems the machine can change its own behaviour and appearance as a consequence of monitoring the world, e.g. intelligent training systems adapt to the learner by providing different pedagogical strategies, sets of knowledge, interactive tools, etc.

To address the problems of requirements for ubiquitous applications [7] I proposed a method for personal and contextual requirements engineering (PCRE) which captures requirements for processes that enable adaptation to individual users, or location-aware functionality for mobile and context-aware applications. However, the PCRE method only addressed the process for requirements analysis and did not deal with intelligent applications or admit actual implications. To address requirements engineering for adaptable, collaborative and intelligent systems, I argue that a new conceptual framework will be necessary to focus on *meta-requirements* which will be necessary for designing machines to deal with change and awareness of the world in which they operate.

In the following sections of this chapter, I review the relationship between requirements and system architecture, in particular from the perspective of non-functional requirements. The next section introduces a case study of architecture requirements, or meta-requirements, for delivering adaptive functions. The final section addresses the process implications for intertwining requirements and architectural considerations for RE, concluding with a short discussion of future developments.
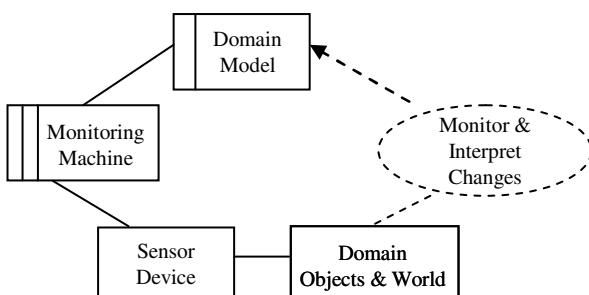
2   Requirements and Architecture

Before proceeding further, it is necessary to pose the question, "Where is the boundary between requirements and architecture?" I believe there is no sharp boundary between the two since requirements analysis progresses gradually towards reification in a design architecture; in spite of exhortations to separate logical (requirements) specification from physical design, transformations between usage, system and development worlds [8], and distinctions between design engineering and requirements engineering [9]. Functional requirements can take on a physical form, and hence have architectural connotations very early in the RE process, in the form of storyboards and user-interface mock-ups. Even though these artefacts are intended to facilitate requirements elicitation, they nevertheless, implicitly, make architectural statements in terms of the UI layout and interactive paradigms (viz. GUI, Web Browser or form-filling interfaces). In collaborative applications, architectural decisions about the distribution of functionality have to be made early in the requirements process. For example, in distributed healthcare applications the allocation of functions to different user-interface roles, such as patient monitoring, nurse supervisor, and expert physician, have to be made early, to effectively focus on appropriate functions for different stakeholders and to enable early prototypes or mock-ups to be constructed. Are such decisions architectural in terms of requirements or system architecture in terms of client-server functionality? Most developers, I suspect, will consider both concurrently.

One means of investigating the dividing line is to use conceptual modelling. In analysing a health care management system, I compared Jackson's Problem Frames [10] and Domain Theory Generic Object System models [11], to investigate how these modelling frameworks packaged requirements into sub-systems. Both approaches partitioned the problem space to specify a distributed computing system architecture with a variety of monitors, transformation and data processing modules, and data display/user-interface components [12]. Although both Problem Frames and Object System models are generic abstractions for requirements engineering, their application produced architectural considerations.
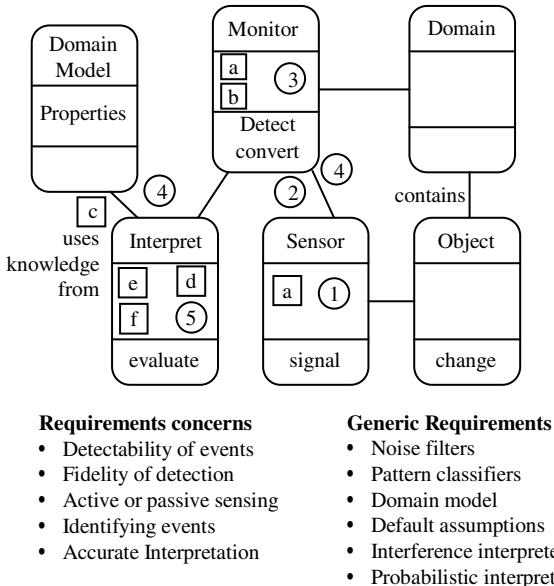
If further argument for the inevitable intertwining of requirements and architecture is needed, then consider the relationship between domain and the designed machine. Jackson's conceptual framework for RE considers requirements in terms of dependencies between the real world and the specified machine inbound, and how the machine acts on the real world outbound. While these dependencies are simple, requirements are simply requirements; but when ambition grows and the machine has intelligence, it needs to maintain a model of the world for interpreting input and planning outbound actions. This in turn implies the need to acquire, be given, and update the world model. This starts to sound like architectural decisions.

These issues can be examined from the perspective of abstract modelling, using Jackson's problem frames [10]. These describe the dependency relationships between requirements, the real world domain and the designed machine, and have had a considerable influence on RE research. I argue that, while this vision is undoubtedly valuable, it can not account for the diversity of requirements found in adaptable, collaborative and intelligent applications. Software machines are embedded in the domain which they serve. The machine has to detect events in the world and respond to them, leading to either taking action in the world or providing information for people to take action. The dependencies between software machines and the environment can be described in Required Behaviour and Commanded Behaviour problem frames, although no single problem frame maps neatly to the monitoring problem; see figure 1. Monitoring becomes a general problem for any application which is mobile, context-aware or adaptive. It therefore merits a generic model that encapsulates the implied requirements inherent in adaptive and context-aware systems.

Fig. 1. The monitoring problem expressed as a problem frame context diagram with dependencies between the components

Requirements for context-aware applications [7] can be described in a generic problem model for monitoring as a set of collaborating objects [11] which presents a requirements view that can be mapped to object oriented design in the Observer pattern in the GOF patterns collection [13]. Figure 2 illustrates the object sensing problem model in UML format.

Requirements concerns
- Detectability of events
- Fidelity of detection
- Active or passive sensing
- Identifying events
- Accurate Interpretation

Generic Requirements
- Noise filters
- Pattern classifiers
- Domain model
- Default assumptions
- Interference interpreters
- Probabilistic interpreters

Fig. 2. Object sensing model as a set of collaborating objects in UML, with requirements concerns and generic requirements as partial solutions

The essence of the problem is a sensing device which captures changes in the world, linked to software which converts signals into a digital format for processing and interprets changes into meaningful events. To do so it needs a model of the world, or at least of the expected events. The problem model shown in figure 2 is composed of two classes: one detects signals and maps then to a processable form, and the second interprets the signals as meaningful events. The third component is a model of the world used for interpretation. Interpreter classes can become increasingly sophisticated as the domain model becomes more complex, with complex methods that enable the machine to adapt more intelligently to its environment. However, successful adaptation depends on solving three problems: acquiring and maintaining accurate domain models, correct inference about how to adapt given a particular context, and the modus operandi of adaptation. So, software machines require models of the world, both for interpreting it and for taking action on the world. Models of the world impose a new class of requirements, which I call *meta-requirements*, since they are not directly part of the functional requirements requested by the users; instead they are requirements which will enable us to build a better and more responsive solution. Meta-requirements can be specialised according to the type of context-awareness

problem, e.g. spatial content, change over time, awareness of the user, groups of users or societies. These meta-requirements need to be elicited and validated early in the process by storyboards and scenarios illustrating a range of adaptive behaviours and contexts. Acquiring and maintaining accurate domain models leads to further meta-requirements which are necessary to specify monitors of domain models, and processes for updating them. This can become a recursive trap of endless monitoring of monitors, although in practice resources enforce limits on the sophistication of awareness a machine (or human) can have about their environment.

Meta-requirements and more mundane monitoring requirements problems are annotated on the sensing model, to draw attention to concerns such as detectability of changes, active or passive sensing, and data fusion from multiple sources. Problems are mapped to generic requirements which point towards solutions via links to design patterns, algorithm libraries, etc.; this generic problem model not only helps analysis of an application by giving a ready-made abstract template describing the problem, but also points the requirements engineer towards solution knowledge. For example, interpreting changes into meaningful events, solutions depend on the modality of data capture, e.g. in numeric data, data mining algorithms (clustering, association, pattern recognisers, classifiers) are appropriate [14]; for language-based media, text-mining algorithms (domain-specific templates, or parsing-semantic analysis) are available [15]; while for image media, image recognition algorithms and model-based scene interpretation can provide the solution [16].

On the output boundary, the machine also needs a model of the world that it seeks to interact with. This model may be sophisticated and include detailed topography of a domain and possible behaviours of the effector device. For example, in robotic applications a model is necessary to plan actions and how to move, grip and interact with the physical world. Less directly coupled are control systems applications which interact via physical devices, e.g. motors, switches, and other actuators. Required Behaviour and Commanded Behaviour problem frames describe these dependencies well but they do not address the problem of domain model acquisition and maintenance, except by recourse to workpiece frames as model editors. In mobile and context-aware systems, the model is essential for planning the response as well as for determining action. Models of the user are necessary to customise the machine's output to an individual's needs; furthermore, information itself may need to be output as an interactive model in decision support and simulation tools.

Finally, configuration of systems, either simple or complex, requires editors (Workpieces problem frames) connected to user interfaces (Model view controller composite frame), with search processes to find appropriate components for the users' goals. This functionality does not map readily to problem frames, which do not contain task/activity-based abstracts; however, it can be modelled in the Domain Theory by the Matching generalised task and Object Allocation object system model. Intelligent, adaptable and configurable systems therefore change the relationship between requirements specification and the real world in a radical manner. Architectural decisions about how the machine understands the world and operates upon it have to be explored early in the requirements process. Configuration makes the relationship between the required system and the world even more complex since there is a range of

possible worlds which machines might be produced to satisfy. The problem frame becomes the architecture for managing and delivering the configured machine.

It is not just an academic debate about abstractions. Early in the requirements process the analyst has to take decisions about the sophistication of functional requirements. For example, take a user goal in a health informatics application, to automatically analyse health data using the language of researchers (epidemiologists); for instance, "Compare the development of obesity in school children in North West health authority regions, according to socio-economic background and age". Such a high-level goal (analysing data using the user's language) motivates many questions for refining the requirements; however, my concern is where the architectural decisions fit it. I argue that architectural calls need to be taken early to manage user expectations. For example, the above requirements could be elaborated along four very different paths, depending on architectural assumptions, e.g.

(i)   Full natural language processing so the user can use speech to control the system. This implies considerable development cost, customising automatic speech recognisers, parsers and semantic analysers for speech understanding with dialogue management for turn taking, error repair, etc. Such systems are currently only research prototypes.

(ii) Development of a restricted natural language interface with a domain-specific lexicon, and guess-ahead sub-language grammar that allows semi-automatic completion of a limited set of analysis requests. Such a solution is still expensive but within the capabilities of current technology [17].

(iii) Development of a standard menu and form-filling user interface where analysis functions are placed in menu lists and parameters are entered in forms. This would probably be the most conventional, low-cost approach.

(iv) Specification of a graphical drag and drop user interface where data sets and analysis functions are represented as icons, and the system is operated by dragging data sets on to the appropriate analysis icon. Some limited form filling may still be necessary, but this interface could have the advantage of providing a map to show where the data sets come from (e.g. areas in the North West).

The above alternatives could all be explored via storyboards and Wizard of Oz techniques to give the user choice about the technology and implementation options. The architectural decisions are being exposed early on; moreover, the user is being involved in assessing architecture costs, and goal trade-offs. Such decision need not involve intelligent systems. In the SCRAM method [18] we explored architectural implications of different design solutions in a shipboard emergency management system. In this case the decisions concerned the type of monitors for detecting fire and chemical hazards, their physical locations on a ship, and the means of communication between the captain and crew (wireless, video link to fire crew, RFID tag location monitoring, etc).

In conclusion, architecture and requirements are interleaved early in the RE process, especially in socio-technical systems where the locations and role of people and hardware devices have to be considered immediately, and in many systems where intelligent processing of input or planning output actions is an option. In the next section I address the architectural implications hidden in non-functional requirements.

2.1   NFRs and Architecture

Non-functional requirements, or soft goals for the followers of i*, have several affinities with architecture. Non-functional requirements have had a history of controversy, whether espoused directly as qualities, performance issues or soft goals [19]. Much of the debate has been sparked by the tension between NFRs as statements of high-level quality criteria which need to be satisfied by a design, viz. security, usability, privacy or maintainability; statements of performance, such as high throughput: transactions per unit time, etc., which have to be satisfied by providing adequate architectural resources to deliver the desired performance; or as direct desiderata of architecture, for example, portability, maintainability, reliability. The latter NFRs have immediate implications for architecture in choice of platforms (Open Source, Microsoft, Apple) for development; modular design and configuration for maintainability, with dependability giving implications for safety kernels and fault tolerant processes for reliability. As the requirements reification process proceeds, NFRs become specifications of functional requirements which deliver the desired qualities. For example, security as a soft goal is elaborated as functional requirements (or hard goals in i*) for checking authorisation of users using a variety of techniques, logs and audit trails to capture undesired access, and encryption to prevent unauthorised access. Rapanotti et al. [20] have explored these implications by positing "architecture frames" based on Jackson's problem frames, whereby the system functions address different aspects of the security problem space, with assumptions annotated on to the specification. In this case, the system architecture is described in terms of processes for checking identity to prevent unauthorised access, audit trails to enable diagnosis of attacks, etc., which are specified to satisfy the safety NFR within the bounds of assumptions made about the behaviour of operators and processes.

Several families of NFRs force consideration of architecture early in design. Usability focuses attention on detailed design of the user interface for ease of learning and operation, even though usability also includes concerns about functional requirements and system effectiveness [21]. Aesthetic design is related to usability to satisfy NFRs for attractive/pleasurable products and user interfaces [22, 23]. Aesthetic design involves not only decisions about the perceptual look-and-feel of a user interface but also how the interaction is structured to achieve the appropriate level of user arousal or flow [24]. Requirements for games imply architectural decisions about the virtual manifestation of the physical game world, as well as monitors and feedback mechanisms to deliver the optimal flow experience [25].

Safety, privacy and reliability turn the designer's attention towards issues of encrypting data, barriers to prevent unauthorised access, identity authorisation procedures to prevent undesired access, and layers of defences to ensure unauthorised attacks do not succeed.
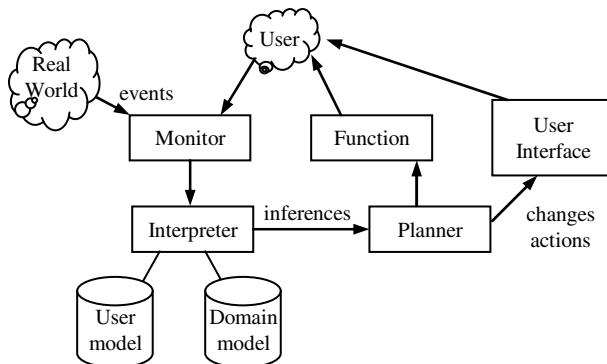
To summarise, most NFRs, even those which are performance related, such as accuracy, response time or throughput, have considerable architectural implications. Performance-related NFRs imply consideration of architecture in terms of resources. For instance, rapid response time needs to be delivered by testing the bandwidth in communication, processor resources and machine loading, data storage access times and contentions, etc. Surveying the breadth of NFR architectural implications is beyond the scope of this chapter; instead, the issue will be investigated further by focusing on a specific area of personalisation.

2.2 Personalisation and Contextualisation

Several related non-functional requirements share the common architectural consideration of designing a system to adapt to the user or its environment, or to provide facilities so the machine can be adapted by the user, following the distinction of adaptive or adaptable systems [26]. Although terminology is inconsistent in the literature, requirements can be specified for personalisation to a specific individual, customisation for a group of people or a culture, or localisation to a country or culture. Other expressions may be more motivated by design considerations of the architecture, such as configurability which tends to assume component-based design, or at least parameterisation of a design.

Two main architectural themes emerge depending on the designer's choice between adaptive and adaptable systems.

Adaptive Systems. These systems automatically adapt to either the user's or the systems' context or both. This implies the need for system components to monitor the user/environment, interpret the events/objects which have been monitored, and then change the system's appearance, behaviour or mix of services. A generic architecture for adaptive systems is illustrated in figure 3.

Fig. 3. Adaptive system architecture, showing both personalisation and contextualisation

An example of adaptive systems for individual users (personalisation) are Recommenders [27], a genre of application which monitors users' behaviour, typically purchasing in e-commerce, and then makes suggestions about other products the user may wish to buy based on their past choices and taxonomies of related products. Examples of adaptive systems for contextualisation are location-aware tourist guides which track the position of a mobile application on PDA, and then provide appropriate information according to the user's location [28, 29].

Adaptive systems pose considerable design challenges in making the recommendations or planned changes as accurately as possible, given limited information about the user and the world. Even with dynamic updating of user and domain models, the system's ability to infer its context accurately is limited. If the change or recommendation is wrong, user dissatisfaction will be the result. Adaptive systems also pose

problems in making change unobtrusive, since unexpected or obtrusive change makes the system inconsistent, which annoys users [30]. Take the problem of adapting to the change in expertise that all users experience as they become familiar with an application. At first they need supportive interfaces with plenty of cues, simple commands and explanatory help; however, as their expertise grows, long cues, easy to use commands and explanations all get in the way of efficient use. Thus ideally the system should radically change its design by providing short cuts, power commands, and ability to program macros for advanced users. Unfortunately, these changes upset the consistency of the interface which adversely impacts on its usability. This is one of several examples where NFRs clash and adaptability can hinder usability.

In reality, most adaptive systems also employ some adaptable components and give the user some control over the adaptive process [31].
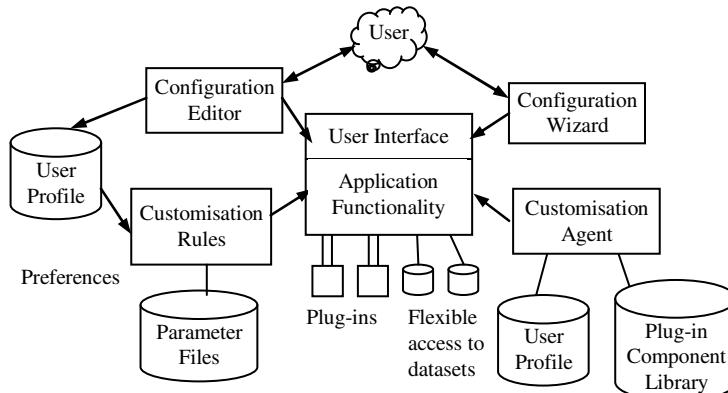
Adaptable Systems. Adaptable systems place the onus for deciding how to change the system on the user. The architectural connotations are summarised in table1 in order of increasing complexity.

Table 1. Architectural implications of adaptability

Level of adaptability		Architecture Implications
1.	Simple parameterisation	Palettes, form filling, limited changes to functions and UI
2.	Initialisation	Set-up Wizards, select functions and UI styles
3.	Feature configuration	Plugs-ins, limited choice of components for variation points
4.	Component engineering	Selection of components, end-user development facilities

In the first simple parameterisation level, customisation or configuration is limited to changing the behaviour of the system by a range of pre-set choices in palettes and form-filling dialogues. Familiar examples in office applications are changing the toolbar in MS Word; or altering security and privacy constraints in a Browser. At this level, additional requirements emerge for the user interface so the user can enter configuration choices. Configuration rule sets are also needed to call functions to make the necessary changes.

The second level provides more user choice in selection of functionality, and this is frequently implemented by configuration wizards which guide the user through a series of set-up choices. Examples are applications which allow a choice of services, such as graphics packages. The additional requirements are for components which can be tailored or added according to a range of users' goals and for the set-up wizard's initialisation sub-system. The architectural components relating to the first- and second-level configurability are summarised in figure 4.

Fig. 4. Architecture for simple configurable applications

Functional requirements will still dominate applications, but with the increase in range of users that any one application intends to serve, the requirements for configuration facilities also increase.
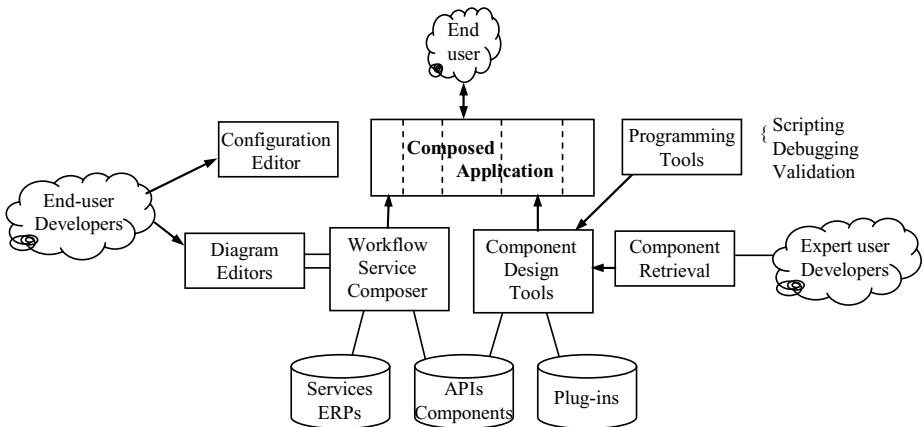
Browsers and application frameworks are examples of the third level where the user has some choice about the functionality that can be selected from a library of components, as well as configuration facilities "inherited" from lower layers, such as User Interface look-and-feel. Single-theme applications may also provide functional choices; for example, social networking sites such as Facebook allow users a choice of a variety of features which they can add to their own personal profile, such as games, sending presents to friends, virtual pets, etc. Requirements become more complex since the application is no longer a simple system but approaches a product line in complexity with a variety of components that can be added to a basic application template.

At the fourth level are component-based engineering systems with end-user (or expert) development facilities, which require functions for searching component libraries, matching component interfaces or APIs, development languages for tailoring components or writing glue code, with programming functions such as syntax editors, debuggers, trace facilities, etc. Enterprise Resource Plans and Product Lines [32] are typical architectures at this level.

Typical configuration components at levels 3 and 4 are illustrated in figure 5. At these levels configuration architecture is dominating the system; moreover, functional requirements are no longer tied to a single stakeholder, instead they have become subject to domain analysis to produce a product line or reuse library for component-based software engineering.

Although many applications have configuration at several levels, the levels do provide a means of classifying the increasing complexity of requirements as configurability becomes more complex, and progresses from end-user customisation to end-user development, and finally to expert component-based software engineering.

The architectural implications for configuration at the second and third levels are illustrated by an example from the ADVISES project in the next section.

Fig. 5. Generic architecture in complex configurable systems

3   Case Study: Configuring e-Science Applications

The ADVISES project is developing software tools to help medical health researchers in epidemiology. The project has two research aims: first to design high-level interfaces for data-driven hypothesis discovery in large-scale epidemiology datasets, essentially providing facilities for researchers to pose research questions in their natural language which the system translates into appropriate analysis routines; and secondly to design usable and effective visualisations for bio-health informatics. As well as producing usable and useful applications for our users, we intend to produce generalisable methods for requirements analysis and usability engineering for e-Science; both acknowledged needs [33].

The architecture to deliver the baseline requirements is illustrated in figure 6.

Essentially the system inputs epidemiology datasets (e.g. complex records of childhood obesity, with variables on age, sex, weight, height, address, socio-economic background, medical history, etc.). These datasets come from local health authorities (Primary Care Trusts) but the data frequently have inaccuracies, so functions for data cleaning and validation are required. The addresses of individuals have to be mapped to postcodes to create area aggregate data. Obesity variables are calculated (Body Mass Index), and a variety of statistical routines are run on the data (correlations, analysis of variance and regressions), before finally outputting the results using maps and graphs. The user interface allows the user to select research questions which then call the appropriate statistical routines, and graph/mapping functions. Examples of questions are:

- Is low income an obesity risk for women living in rural as well as urban areas?
  (select adult females in dataset; call 2-way analysis of variance urban/rural, high/low income women)
- Is obesity varying throughout childhood less than it did 10 years ago?
  (select child datasets most recent, 10 years ago, for each year, call analysis of variance recent/10 years ago, regression on age cohorts).
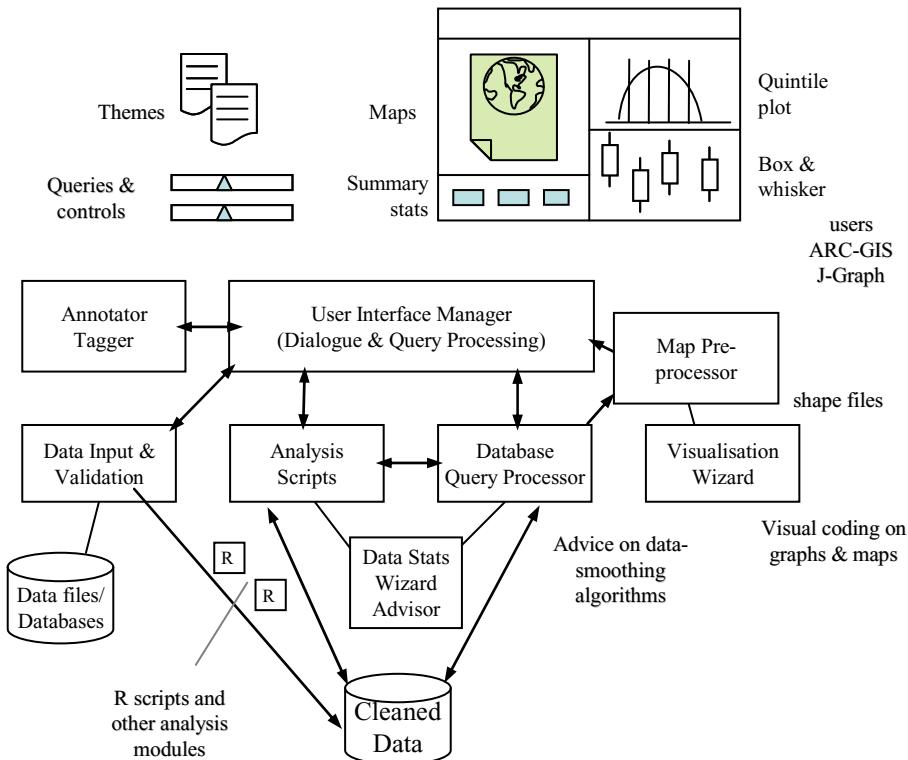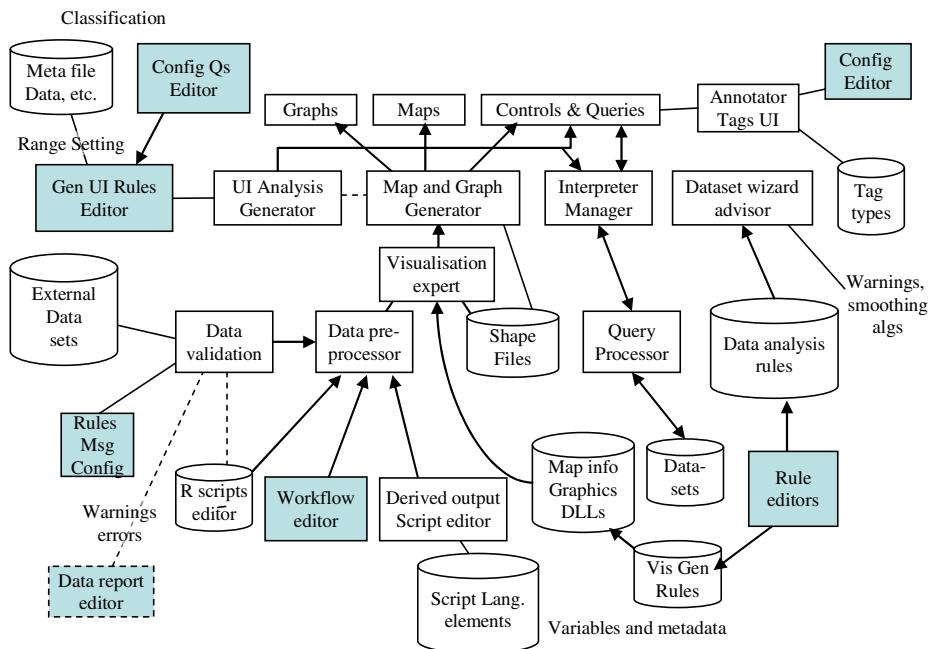
Fig. 6. ADVISES basic functional architecture

The requirements in brackets give a high-level summary of the necessary interpretation rules for the research questions. Some simplification was negotiated with users so questions are formulated from menus of keyword lists to avoid complex natural language processing; nevertheless, an intelligent rule-based interpreter is still required. Other intelligent modules in the system include a visualisation expert which decides how to represent different variables, e.g. averages, data, continuous/discrete data, on maps by colour coding, texture shading, size, etc. There is also a statistical advisor for novice users which warns when the statistical tests implied in the users' questions may be invalid with respect to the data (e.g. use of parametric tests on non-normal distributions). Even this basic architecture is complex, although it is only serving the needs of the immediate users in a limited range of epidemiological investigations.

However, the ADVISES tools should be generic so they can serve a wider range of users in the e-science medical informatics community. Two strategies could be followed. First, make the tools customisable and rely on the programming expertise of the end users. This is possible to some extent since some medical researchers are expert in statistical programming languages such as R, but this approach would still limit configurability to the statistical analysis routines. The second approach is to add

configuration editors, so medical researchers who are not programming experts can configure the tools for their own datasets and analysis goals. This approach has been followed in several e-science applications by providing configurable workflow editors [34] which then call web services to compose user-selected analysis processes.

The additions to the ADVISES architecture for this configurability are shown in figure 7. Several editors have been added to allow users to change validation checks and error messages for input data files, edit workflows to configure statistical analyses, and change annotations on results. Configuration therefore adds considerable functionality to the basic requirements, in the form of data editors which change messages and tags, diagram editors to compose statistical analysis sequences, and rule editors to change functionality of the rule-based experts for analysis advice and visualisation.

Fig. 7. Extended ADVISES architecture for configurable applications. Configuration delivery components are shaded.

Even this level of complexity hides other requirements which are necessary to enable interpretation of the user changes; for instance, workflow modules have an embedded diagram interpreter with rules to link diagram nodes to appropriate statistical analysis functions, and validation checks to ensure that users compose legal analysis sequences. Further configuration can be added to make the workflow module fully generic, so new analysis functions can be added; this requires form-filling editors to link new analysis function APIs to diagram nodes, and rule generation in the diagram interpreter.

Configuration which allows the user to add new functions approaches the complexity of a visual programming language in level 4 of the configuration framework. The final point to note is the extent to which configuration facilities may actually be used. Given that something in the order of 20-30% of the requirements in the extended system are fulfilling configuration needs alone, this represents a considerable investment in reaching more users. Layers of configuration are planned to reach, first, other epidemiologists, who will require their own research questions, datasets and analysis routines. Later layers could extend the system for other research scientists if a wider-ranging ontology of research questions could be developed. The critical question is, "Will the users accept the effort of configuration, or just demand that a default version be prepared for them by software engineers?" Hence there is a dilemma about who the target users of configuration facilities may be. Most research scientists are very busy and dislike spending time not directly linked to their own research work. We therefore targeted the configuration editors towards expert end-users of e-science support staff who can configure applications for end users.

As illustrated in the preceding sections of this chapter, systems that can be adapted by the user and automatically adapt for the user hide a large number of functional requirements which do not directly map to the users' goals. These requirements may be prompted by non-functional requirements for configuration and usability, or to enhance the quality of service for a user's goal. For example, "display results on maps" is a user goal in the ADVISES toolset which requires a visualisation expert to enhance the quality of visualisation by automatically selecting the appropriate visual codings for different data types. These requirements emerge during the dialogue between users and designers, as the requirements engineer explores how the design might be augmented to satisfy the users' goals in a more sophisticated manner. Hence they are in a sense "meta-requirements": they exist to deliver the users' goals (conventional functional requirements). Meta-requirements also extend the requirements specification from a single static version of the system to one which can be changed into versions at design time or change its behaviour at run time. Architecture intrudes into the requirements process as the possibilities for intelligent processing are explored, and configuration enables several groups of stakeholders to be served by one application, but at a cost of involving the users in discussions about how configuration may be implemented.

4   Conclusions

The implications of meta-requirements are that we need to focus less on conventional functional requirements but more on questions about what the machine should know about its environment and how it can use the knowledge it possesses to adapt to the world. To an extent this concern is not new. Fickas and Feather [5] drew attention to requirements monitoring in which an application checked conformance of delivered service against a requirements target by monitoring, and Robinson has developed several adaptive requirements monitoring systems for e-commerce and other applications [35, 36].

Other questions concern the range of functional requirements which could be part of the application configuration process. Requirements bundling for product releases has been explored in processes for configuration and version control and by applying evolutionary computing to optimise cost-benefit trade-offs for service bundling [37]. While configuration has been addressed in requirements methods for product lines and ERP configuration methods, requirements analysis for adaptive systems based on domain and user models has received little or no attention. Ideally the more knowledge possessed by the machine, the more accurate and sophisticated its inferences could be. However, acquisition of domain knowledge incurs a cost and is a well known bottleneck in RE and the development of intelligent systems. One approach to finessing the problem is to employ machine learning, and this provides another twist to the meta-requirements concept: we need to ask whether it is necessary for the machine to learn (volatility and knowingness of the world); or how the machine should learn with respect to the application domain (e.g. classifiers, BBNs, neural nets, explanation-based learning, etc.).

Many research issues remain to be solved at the architecture-requirements interface. This chapter has focused on architecture in the physical implementation sense; however, architecture is also a conceptual organisation in business or enterprise systems architecture [38]. Some convergence has been made in modelling enterprise architecture in the e3value method [39] which describes components for valued added services, although the connection of e3value analysis in enterprise architecture to requirements is so far a matter of heuristic methods for discovering requirements implications of enterprise designs.

Better languages are needed to map the convergence of architecture, requirements and design, particularly in the field of services engineering, where distributed web applications pose questions of distribution in business analysis as much as in requirements specification. Requirements languages, notably i*, have been used to describe service architectures [40], but these descriptions do not address the deeper issues of service composition and mapping requirements to components at optimal levels of granularity to enable flexible configuration and reuse. Product-line architectures and domain analysis methods [32] do provide notations for feature analysis and variation points, but also fail to address the complex intersection of requirements and design.

In conclusion, this chapter has proposed a definition of meta-requirements, as requirements for maintaining the machine's ability to interpret and act in a domain, in contrast to functional requirements which are responses to the user's goals or logical consequences of interacting with a domain. As software becomes more intelligent, context-aware and adaptable, the type of requirements and the way we capture them will also need to adapt to future worlds which will become harder to anticipate. We will need to educate users and requirements engineers about the architectural implications of design decisions from the enterprise to the software level, while also solving the more difficult problem of component abstraction and granularity to enable flexible and efficient composition for requirements in the future with service-led component engineering.

References

1. Swartout, W., Balzer, R.: On the inevitable intertwining of specification and implementation. Communications of the ACM 25(7), 435–30 (1982)
2. Sommerville, I., Sawyer, P.: Requirements engineering: A good practice guide. Wiley, Chichester (1997)
3. Robertson, J., Robertson, S.: Mastering the requirements process. Addison Wesley, Harlow (1999)
4. Lehman, M.M., Ramil, J.F.: Software evolution in the age of component based systems engineering. IEE Proceedings: Software 147(6), 249–255 (2000)
5. Fickas, S., Feather, M.S.: Requirements monitoring in dynamic environments. In: Harrison, M.D., Zave, P. (eds.) Proceedings: 1995 IEEE International Symposium on Requirements Engineering (RE 1995), March 27-29, 1995, pp. 140–147. IEEE Computer Society Press, Los Alamitos (1995)
6. Fischer, G.: Shared knowledge in cooperative problem-solving systems: Integrating adaptive and adaptable components. In: Schneider-Hufschmidt, M., Kuehme, T., Malinowski, U. (eds.) Adaptive user interfaces: Principles and practice, pp. 49–68. Elsevier Science Publishers, Amsterdam (1993)
7. Sutcliffe, A.G., Fickas, S., Sohlberg, M.: Personal and contextual requirements engineering. In: Proceedings: 13th IEEE International Conference on Requirements Engineering, Paris, August 29- September 2, 2005, pp. 19–28. IEEE Computer Society Press, Los Alamitos CA (2005)
8. Jarke, M., Pohl, K., et al.: Requirements engineering: An integrated view of representation. In: Sommerville, I., Paul, M. (eds.) ESEC 1993. LNCS, vol. 717, pp. 100–114. Springer, Heidelberg (1993)
9. Rolland, C., Achour, C.B., et al.: A proposal for a scenario classification framework. Requirements Engineering 3(1), 23–47 (1998)
10. Jackson, M.: Problem frames: Analysing and structuring software development problems. Pearson Education, Harlow (2001)
11. Sutcliffe, A.G.: The Domain Theory: Patterns for knowledge and software reuse. Lawrence Erlbaum Associates, Mahwah (2002)
12. Sutcliffe, A.G., Papamargaritis, G., Zhao, L.: Comparing requirements analysis methods for developing reusable component libraries. Journal of Systems and Software 79(2), 273–289 (2006)
13. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design patterns: Elements of reusable object-oriented software. Addison Wesley, Reading (1995)
14. Maimon, O., Rokach, L.: Data mining and knowledge discovery handbook. Springer, Heidelberg (2005)
15. Ananiadou, S., McNaught, J. (eds.): Text mining for biology and biomedicine. Artech House, Norwood (2006)
16. Kuncheva, L.I.: Combining pattern classifiers: Methods and algorithms. Wiley, New York (2005)
17. Hallett, C., Scott, D., Power, R.: Intuitive querying of e-health data repositories. In: Proceedings: UK e-Science All-Hands Meeting, Nottingham (2005)
18. Sutcliffe, A.G., Ryan, M.: Experience with SCRAM: A Scenario Requirements Analysis Method. In: Proceedings: IEEE International Symposium on Requirements Engineering: RE 1998, Colorado Springs CO, April 6-10, 1998, pp. 164–171. IEEE Computer Society Press, Los Alamitos (2006)

19. Glinz, M.: On non-functional requirements. In: Sutcliffe, A.G., Jalote, P. (eds.) Proceedings: 15th IEEE International Requirements Engineering Conference RE 2007, New Delhi, India, October 15-19, 2007, pp. 21–28. IEEE Computer Society Press, Los Alamitos (2007)

20. Rapanotti, L., Hall, J., Jackson, M., Nuseibeh, B.: Architecture Driven Problem Decomposition. In: Proceedings of 12th IEEE International Requirements Engineering Conference (RE 2004), Kyoto, Japan, September 6-10 (2004)

21. ISO 9241: Ergonomic requirements for office systems with visual display terminals (VDTs). International Standards Organisation (1997)

22. Norman, D.A.: Emotional design: Why we love (or hate) everyday things. Basic Books, New York (2004)

23. Green, W.S., Jordan, P.W.: Pleasure with products: Beyond usability. Taylor & Francis, London (2001)

24. Csikszentmihalyi, M.: Flow: The classic work on how to achieve happiness (Revised ed.). Rider, London (2002)

25. Callele, D., Neufeld, E., Schneider, K.: Emotional requirements in video games. In: Proceedings: 14th IEEE International Requirements Engineering Conference RE 2006, pp. 299–302. IEEE Computer Society Press, Los Alamitos CA (2006)

26. Fischer, G., Scharff, E.: Meta-design: Design for designers. In: Boyarski, D., Kellogg, W.A. (eds.) Conference Proceedings: DIS 2000 Designing Interactive Systems: Processes, Practices Methods and Techniques, August 17-19, 2000, pp. 396–405. ACM Press, New York (2000)

27. Herlocker, J., Konstan, J., Treveen, L., Riedl, J.: Evaluating collaborative filtering recommender systems. ACM Transactions on Information Systems 22(1), 5–53 (2004)

28. Keller, G., Teufel, T.: SAP/R3 process oriented implementation. Addison Wesley-Longman, Reading (1998)

29. Susani, M.: Mobile interaction design in the age of experience ecosystems. In: Sears, A., Jacko, J.A. (eds.) The human-computer interaction handbook: Fundamentals, evolving technologies and emerging applications, 2nd edn., pp. 459–468. Lawrence Erlbaum Associates/Taylor Francis, New York (2008)

30. Shneiderman, B., Plaisant, C.: Designing the user interface: Strategies for effective interaction, 4th edn. Addison-Wesley, Reading (2004)

31. Lieberman, H. (ed.): Your wish is my command: Programming by example. Morgan Kaufmann, San Francisco (2001)

32. Weiss, D.M., Lai, C.T.R.: Software product-line engineering: A family-based software development process. Addison-Wesley, Reading (1999)

33. Beckles, B.: User requirements for UK e-Science grid environments. In: Proceedings: UK e-Science All-Hands Meeting, Nottingham (2005)

34. Oinn, T.M., Greenwood, R.M., Addis, M., et al.: Taverna: Lessons in creating a workflow environment for the life sciences. Concurrency and Computation: Practice and Experience 18(10), 1067–1100 (2006)

35. Robinson, W.N.: Requirements monitoring for enterprise systems. Requirements Engineering 11(1), 17–41 (2006)

36. Robinson, W.N., Pawlowski, S.: Managing requirements inconsistency with development goal monitors. IEEE Transactions on Software Engineering (November/December 1999)

37. Finkelstein, A., Harman, M., Afshin Mansouri, A., Ren, J., Zhang, Y.: Fairness analysis in requirements engineering. In: Proceedings: 16th IEEE International Requirements Engineering Conference RE 2008, Barcelona. IEEE Computer Society Press, Los Alamitos (2008)

38. Hansen, S., Berente, N., Lyytinen, K.: Requirements in the 21st century: Current practice and emerging trends. In: Proceedings: Design Requirements Workshop, June 3-6, 2007. Case Western University, Cleveland (2007)
39. Gordjin, J., Yu, E., VanDerRaadt, B.: E-Service design using i* and e3value modelling. IEEE Software 23, 26–33 (2006)
40. Kartseva, V., Gordijn, J., Tan, Y.H.: Value-based design of networked enterprises using control patterns. In: Sutcliffe, A.G., Jalote, P. (eds.) Proceedings: 15th IEEE International Requirements Engineering Conference RE 2007, New Delhi, India, October 15-19, 2007, pp. 91–100. IEEE Computer Society Press, Los Alamitos (2007)

Requirements Evolution and What (Research) to Do about It

Neil A. Ernst[1], John Mylopoulos[1,2], and Yiqiao Wang[1]

[1] University of Toronto, Dept. of Computer Science, Toronto, Canada
`{nernst,jm,yw}@cs.toronto.edu`
[2] University of Trento, Dept. of Information Engineering and Computer Science,
Trento, Italy
`jm@disi.unitn.it`

Abstract. Requirements evolution is a research problem that has received little attention hitherto, but deserves much more. For systems to survive in a volatile world, where business needs, government regulations and computing platforms keep changing, software systems must evolve too in order to survive. We discuss the state-of-the-art for research on the topic, and predict some of the research problems that will need to be addressed in the next decade. We conclude with a concrete proposal for a run-time monitoring framework based on (requirements) goal models.

Keywords: Requirements, evolution, monitoring, satisfiability.

1   Introduction

It has been known for decades that changing requirements constitute one of the greatest risks for large software development projects [1]. That risk manifests itself routinely in statistics on failure and under-performance for such projects. "Changing requirements" usually refers to the phenomenon where stakeholders keep changing their minds on what they want out of a project, and where their priorities lie. Little attention has been paid to post-deployment requirements changes[1], occurring after a system is in operation, as a result of changing technologies, operational environments, and/or business needs. In this chapter we focus on this class of requirements changes and we refer to them as requirements evolution.

Evolution is a fact of life. Environments and the species that operate within them – living, artificial, or virtual – evolve. Evolution has been credited with the most advanced biological species that has lived on earth. The ability to evolve has also come to be treated as a prerequisite for the survival of a species. And, yet, evolution of the software systems species has only been studied at the level of code and design, but not at the level of requirements. In particular, there has been considerable research on software evolution, focusing on code reengineering and migration, architectural evolution, software refactoring, data migration and integration.

---

[1] . . . with the notable exception of research on traceability mechanisms. Of course, traceability is useful for evolving requirements, but doesn't actually solve the problem.

However, the problem of post-deployment evolution of *requirements* (as opposed to architecture, design and/or code) hasn't made it yet into research agendas (see, for example, the topics that define the scope of a recently held workshop on "Dynamic Software Evolution", http://se.inf.ethz.ch/ moriol/DSE/About.html).

There are important reasons why requirements evolution is about to become a focal point for research activity in Software Engineering. The change from local, isolated communities to the global village isn't happening only for commerce, news and the environment. It is also happening for software systems. In the past, operational environments for software systems were stable, changes were local, and evolution had only local impact. Today, the operational environment of a growing number of software systems is global, open, partly unknown and always unpredictable. In this context, software systems have to evolve in order to cope ("survive" is the technical term for other species). Some of this evolution will be at the code level, and some at the architectural level. The most important evolution, however, will have to take place at the requirements level, to ensure that a system continues to meet the needs of its stakeholders and the constraints – economic, legal and otherwise – of its operational environment.

An obvious implication of the rise to prominence of requirements evolution is that the research to be conducted will have to be inter-disciplinary. Researchers from Management, Organizational Theory, Sociology and Law will have to be part of the community that studies root causes for change and how to derive from them new requirements. Evolution mechanisms and theories that account for them have been developed in Biology, Engineering, Organizational Theory and Artificial Intelligence. Some of these may serve as fruitful starting points for the research to be done.

A precondition for any comprehensive solution to the problem of evolving requirements is that design-time requirements are properly captured and maintained during a system's lifecycle, much like code. Accordingly, we (optimistically) predict that the days of lip service to requirements are coming to an end, as Software Engineering Research and Practice opt for lasting technical solutions in a volatile world. Growing interest in topics such as autonomic software, semantic web services, multi-agent and/or adaptive software, peer-to-peer computing (...and more!) give some evidence that this optimism is not totally unwarranted.

The main objective of this chapter is to review the past (section 2) and suggest a research agenda on requirements evolution for the future (section 3). After a general discussion of topics and issues, we focus on one item of this agenda – monitoring requirements – to make the discussion more concrete. The remainder of the paper presents some of our on-going work on the problem of monitoring requirements and generating diagnoses. Technical details of this work have been presented in [2].

2   The Past

In the area of software evolution, the work of M. Lehman [3] stands out, with a proposal backed by empirical data for laws of program evolution. These laws offer a coarse grain characterization of types of software and the nature of its

evolution over its lifetime. Lehman's work on program evolution actually started with a study of the development of OS/360, IBM's flagship operating system in the late 60s. The study found that the amount of debugging decreased over time and concluded that the system would have a troubled lifetime, which it did. A few years later, Fred Brooks (academic, but also former OS/360 project manager) excoriated the IBM approach to software management in his book "The Mythical Man Month" [4]. Using Lehman's observations as a foundation he formulated his own "Brooks' Law": adding manpower to a late software project makes it later; all software programs are ultimately doomed to succumb to their own internal inertia. Fernandez-Ramil et al. [5] offers a comprehensive collection of recent research on the topic of software evolution.

As noted in the introduction, the focus of much of the research on software evolution has been on the code. Few software systems come with explicit links to requirements models. Pragmatically, it is simpler to understand system evolution by examining code artifacts – files, classes, and possibly UML diagrams. For example, Gîrba and Ducasse [6] present a metamodel for understanding software evolution by analysing artifact history. Their discussion pays little attention to the problem domain, likely because there is no clear way of reconstructing it. Similarly, Xing and Stroulia [7] use class properties to recapitulate a series of UML class diagrams to detect class co-evolution. Again, this study pays no attention to the causes of these changes, some of which relate to requirements.

We begin this section with a discussion of work that first identified the issue of requirements evolution, summarizing various attempts to characterize the problem using frameworks and taxonomies. We conclude with a look at current approaches to managing evolving requirements, including module selection, management, and traceability.

2.1   Early Work

When discussing the drivers behind an evolving model, and ways of managing that evolution, there are many definitions and terminologies in use. Various researchers have attempted to categorize the phenomena of evolving systems, the majority of whom come from the software maintenance community. The importance of requirements models throughout the software lifecycle has long been recognized. Basili and Weiss [8] reported that the majority of changes to a system requirements document were trivial, requiring less than three hours to implement. However, a few errors required days or weeks to resolve. Similarly, Basili and Perricone [9] report that of errors detected in a system during implementation, 12% were due to poor requirements (and 36% due to poor specifications). Rather than present an overarching temporal list, we categorize pertinent research into categories and draw distinctions between them. The majority of these papers present viable approaches to understanding the concepts involved. Where there are differences, they are typically the result of different perspectives.

Harker et al. [10] classifies requirements into:

1. enduring – core to the business;
2. mutable – a product of external pressures;

3. emergent – surfaced during thorough elicitation;

4. consequential – identified after product implementation;

5. adaptive – requirements that support system agility; and finally,

6. migration requirements – those which help during the changeover.

Where Rajlich and Bennett [11] and Lientz and Swanson [12] (see below) are discussing the process (actions) of managing these changing requirements, Harker et al. are focusing on the structure of those requirements. There are many terms one might apply to change and evolution in software. Rowe et al. [13] define evolvability as "a system's ability to accept change", with the addition of the constraints that it be a least-cost change, as well as one preserving the integrity of the architecture. It isn't made clear why the preservation of architectural form is important – perhaps for backwards compatibility.

They mention four properties of evolvability: generality, adaptability, scalability, and extensibility. There is a two-way relationship among these. From generality to extensibility there is an increasing amount of change required for a given requirement; from extensibility to generality there is a increasing amount of up-front cost. In other words, to build an extensible system is initially cheap, but costly when the change needs to be made, since radical extensions to the architecture are required. This dimension characterizes a given system in terms of an architectural state space – similar to the 'space of action possibilities' described in Vicente [14, p. 123].

Another state space model is covered in Favre [15], which presents a '3D' model of evolution. The three dimensions are *model abstraction* (model, meta-model, etc.), *engineering/implementation*, and *representation*. Each dimension has an associated series of stages, and Favre uses the intersection of these dimensions to map a particular product in a software space. For example, engineering stages consist of requirements, architecture, design and implementation – the traditional phases of software development. If we talk about a system at the meta-level of requirements, with an implicit representation, an example might be a conceptual metamodel such as the UML metamodel.

Favre suggests the importance of combining these orthogonal dimensions is for understanding how the various dimensions co-evolve. For example, it is important to consider whether the specification is co-evolving with the implementation, whether the modeling language is keeping pace with the technology, etc. As Favre concludes, it is important to remember that 'languages, tools, and programs evolve in parallel'.

To understand the motivations behind making changes to a system, a seminal work in the field of software maintenance is Swanson [16] (see also Lientz and Swanson [12]). They categorize software evolution into adaptive (environmental changes), corrective and perfective (new internal requirements) maintenance. Later work has added the notion of preventive maintenance. The context in which this work was done differs greatly from today; however, this division can be a useful way of understanding the nature of the changes in the environment which provoke reaction in the system.

Rajlich and Bennett [11] propose a different model, reflecting their belief that the post-delivery lifecycle is more complex than the term 'maintenance' reflects. They divide the post-delivery phase into four stages: evolution (major updates), servicing (corrective maintenance), phaseout, and closedown. Such a model reflects activities undertaken by companies like Microsoft and its Windows family of products. A requirements model is involved at the evolution (and possibly the servicing) stage. This model may be at odds with more agile development techniques, although there is a lack of research into the implications of agile techniques for software maintenance (although see Svensson and Host [17] for a preliminary assessment).

The process of managing change is also the subject of Nelson et al. [18]. They talk about flexibility in the context of business processes. Successful organizations (and their systems) exhibit adaptability, or the willingness to 'engage the unfamiliar'. Flexibility is the ability of such a system to handle change pressures and adapt. This is characterized as either structural or procedural. There are several determinants of each. Structural flexibility relies on modularity, or design separation; change acceptance, the degree to which the technology has built-in abilities to adapt; and consistency, the ability to make changes painlessly. Procedural flexibility is determined by the rate of response, system expertise (up-to-date knowledge), and coordinated action. Together, these characteristics define what it means for a system to adapt to a given change event. High levels of the preceding characteristics imply a high affinity to accommodate change.

Many proposed requirements engineering frameworks ignore change acceptance, relying on users to understand the nature of the change, and manually incorporate it. Buckley et al. [19] offer a taxonomy that describes the HOW, WHEN, WHERE and WHAT questions of software evolution (but not WHY or WHO). They suggest that such a taxonomy will help in understanding the mechanisms of the change, with a view to designing strategies for accommodating these processes. They categorize these questions into four dimensions of software change: change support, temporal change properties, object of change, and system properties. They analyze three tools which have seen evolution along the lines of the taxonomy. Requirements change is not specifically mentioned, but can be thought of as driving temporal change properties – e.g., a change in the environment will drive a change in the software.

In an attempt to bring together various software maintenance taxonomies, Chapin et al. [20] propose a high-level taxonomy for understanding the types of activities that occur in this area. The ontology is based on an impact model, examining evolution in the context of change to business processes and change to software (presumably this can be extended to refer to software-based system). They classify change events into a cascading, 4-part hierarchy of 12 categories, reflecting what they say is the wide diversity of concepts that exist in research and practice. Extending from perfective, adaptive, and corrective, they include four categories: support interface, documentation, software properties, and business rules.

For example, within business rules they define three changes: reductive, corrective, and enhancive. Their business rules category has the closest relationship to the concept of requirements. Changes in this category also have the highest impact on both software and business processes. According to this definition then, requirements changes will have the greatest cost for an organization. This idea certainly fits with the research findings suggesting that fixing requirements problems consistitute by far the largest cost in system maintenance (e.g., see Standish reports, although these are of uncertain research value). However, Chapin et al. do not explicitly discuss requirements. For example, they mention 'change requests', user-driven needs, as drivers, but make no reference to updated requirements. They also distinguish between maintenance – changes in the first 3 categories – and evolution, which (in their definition) primarily affects business rules. This is certainly the sense this chapter refers to.

Many of the prior papers mention requirements only because an implicit change in requirements has driven some corresponding change in the implemented software system. However, our research is concerned with the nature of these requirements changes. This was also the subject of research by Massimo Felici. In [21], he refers to requirements evolving in the early phases of a system, with perfective maintenance occurring toward the end of a system's lifespan. However, this view is at odds with the current view of requirements as something that exists throughout the project lifecycle.

In [22], the analysis begins with the observation that requirements frameworks generally do a poor job handling evolving requirements. The PROTEUS classication of requirements evolution (that of Harker et al.) is presented as a way to understand how requirements evolve. A requirement is either stable or changing. If the latter, it can be one of five subtypes: mutable, due to environmental factors; emergent, due to stakeholder engagement; consequential, resulting from the interaction of system and environment; adaptive, due to task variation; and migration, arising from planned business changes. This taxonomy of causes of requirements evolution is fairly concise yet comprehensive. Felici also discusses the similar causal taxonomy of Sommerville and Sawyer [23], which they term 'volatile requirements'. Sommerville and Sawyer use the categories of mutable, emergent, consequential, and compatibility requirements. Similarly, [24] presents the EVE framework for characterizing change, but without providing specifics on the problem beyond a metamodel.

2.2   Requirements Management

Requirements management studies how best to control the impacts of change on requirements. Properly managing change events — such as new stakeholder requirements — can be essential to reducing the amount of model evolution that occurs. A key research contribution in this area is a better understanding of how exactly these external pressures manifest themselves.

For example, Stark et al. [25] discuss change to requirements during the system release process. A release can be a minor version of an existing product, so this is a legitimate use of the term requirements evolution. They were responsible

for the development of missile warning software. The study produced some invaluable information on how change was occurring in the project: for example, 108 requirements were changed, and of this figure, 59% were additions (scope creep). They attempt to produce a predictive model of changes, but it isn't clear how generalizable such a model would be.

Similar research is reported by Basili and Weiss [8], in the context of another military project, the A-7 control software. They describe the nature of requirements changes on the project. The biggest issue seemed to be that many of the facts used in the requirements document were simply incorrect (51% of errors). They also categorize the errors from trivial to formidable. Although only one of the latter was encountered, it required 4 person-weeks of effort to resolve.

Lormans et al. [26] motivates a more structured approach to requirements management. They used a formal requirements management system, but encountered difficulty in exchanging requirement models with clients. Such 'models' were often in text form, or semi-structured representations. They propose a more elaborate management model that can address some of these challenges.

Wiegers [27] discusses four common tools for requirements management. To some degree each support the notion of managing evolving requirements. There is a question as to how well these tools reflect the reality in the code. Typically the tools store requirements as objects or relations, and then allow various operations, such as mapping to test suites or design documents. The biggest challenge is often maintaining traceability links between requirements and implementation. Roshandel et al. [28] discuss one approach for managing architectural evolution in sync with code. Another approach is to ignore everything but the source code, and reverse engineering requirements from there, as described in Yu et al. [29]. Finally, managing requirements will require configuration management tools similar to CVS, Subversion, and other code repositories. Tools like diff or patch need analogues in the model domain. Work in model merging, e.g., Niu et al. [30] will be important here.

Another emerging issue is the design of dynamic, adaptive software-based system. We discuss one approach to design such a system in section 4. Such systems are composed of multiple components, which may not be under one's direct control. Such systems are often categorized as Software as Service (SaaS) or Service-Oriented Architecture (SOA) domains. For these domains, we view requirements as the business drivers that specify which components, and in what priority, should be composed. A paper by Berry et al. [31] provides a useful 'four-level' characterization of the nature of the compositions and adaptations involved: the levels correspond to who (or what) is doing the requirements analysis: 1) the designer, on the domain; 2) the adaptive system, upon encountering some new condition; 3) the designer of the system, attempting to anticipate the nature of the second adaptation; or 4) a designer of new adaptation mechanisms.

Composing these components (or agents, or services) is an emerging research problem, and one in which requirements evolution will have a major role. Work on software customization Liaskos [32], for example, provides some insight into techniques for managing such composition, although it ignores the problem of

changes in the underlying requirements themselves. Related work in Jureta et al. [33] makes more explicit the idea that requirements cannot be fully specified prior to system implementation. They characterize this approach as one in which there is only one main requirement for the system, namely, that the system be able to handle any stakeholder requirement. Determining which stakeholder requirements are reasonable (i.e., within system scope) will be an important research problem.

Recent work has focused on Commercial Off-The-Shelf (aka COTS) components. A change in one component, driven by an evolution in a particular requirement, might impact other components. Etien and Salinesi [34] term this *co-evolution*. It is a challenge to integrate these COTS-based systems in such an environment:

> [COTS-based systems] are uncontrollably evolving, averaging up to 10 months between new releases, and are generally unsupported by their vendors after three subsequent releases. (Boehm [35, p. 9])

The work that led to that analysis, Yang et al. [36], discusses the issue of COTS-based software and requirements. They claim that defining requirements before evaluating various COTS options prematurely commits the development to a product that may turn out to be unsuitable. They argue for a concurrent development methodology that assesses COTS feasibility at the same time as developing the system itself. In other words, they argue for a spiral model approach (Boehm, 1988) to developing the requirements for such systems (not surprisingly). Nuseibeh [37] makes a similar point with his 'Twin Peaks' model. A requirements management tool that provided support for understanding the features, capabilities, and likelihood of change in various COTS products would be invaluable in such systems. Understanding how the requirements themselves might evolve would be one important aspect.

Traceability is an aspect of requirements management that identifies interdependencies between elements in the environment to elements within a system. Traceability is a necessary, but not a sufficient mechanism for managing evolving requirements. Without a link, the downstream impact of requirements changes will not be clear. Traceability can be divided into two aspects, after Gotel and Finkelstein [38]. One needs a trace from the various phenomena in the environment, to the specification of the requirements for the system. Once specified, a link should also be established between the specification and the implementation. The former case is relatively less studied, and is less amenable to formalization.

Requirements monitoring, first proposed in [39], and extended in [40], is one mechanism for tracing between requirements and code. Monitoring involves inserting code into a system to determine how well requirements are being met. A monitor records the usage patterns of the system, such as numbers of licenses in use. This information can be extracted and used to evolve the system, possibly dynamically. In this sense, monitors are quite similar to control instrumentation in, for example, industrial plants. This approach is promising, but does assume that requirements and environmental conditions can be specified accurately enough that monitoring is possible.

Traceability is more difficult with non-functional requirements, because by definition these requirements do not have quantitative satisfaction criteria. Cleland-Huang et al. [41] discuss a probabilistic information retrieval mechanism for recovering non-functional requirements from class diagrams. Three broad categories of artifacts are defined. A softgoal model is used to assess change impacts on UML artifacts, and an information retrieval approach is used to generate the traceability links between the two models. Ramesh and Jarke [42] give a lengthy overview of empirical studies of requirements traceability.

Monitoring also has a vital role to play in the design of autonomic systems ([43]). These are systems that can can self-repair, self-configure, self-optimize and self-protect. Of course, the ability to self-anything presupposes that such systems monitor the environment and their performance within that environment, diagnose failures or underperformance, and compensate by changing their behaviour.

3   A Research Agenda for 2020

So, assume that we have our operating software system and changes occur that need to be accommodated, somehow. The changes may be in the requirements of the system. For example, new functions need to be supported, or system performance needs to be enhanced. Increasingly, changes to requirements are caused by laws and regulations intended to safeguard the public's interests in areas of safety, security, privacy and governance. Changes may also be dictated by changing domain assumptions, such as increased workload caused by increased business activity. Last, but not least, changes may be dictated by new or evolving technologies that require migration to new platforms. New or evolving technologies can also open new opportunities for fulfilling business objectives, for example by offering new forms of business transactions, as with e-commerce and e-business.

Whatever the cause for a change, there are two basic approaches for dealing with it. The first, more pedestrian, approach to change has software engineers deal with it. This approach has traditionally been called software maintenance and it is generally recognized as the most expensive phase in a software system's lifecycle. A second approach for dealing with a change is to make the system adaptive in the first place, so that it can accommodate changes by using internal mechanisms, without human intervention or at least with intervention from end users only. The obvious advantage of this approach is that it makes change more immediate and less costly. Its main drawback, on the other hand, is that change needs to be thought out at design time, thereby increasing the complexity of the design. The recent focus on autonomic and/or adaptive software in the research community suggests that we are heading for automated approaches to software evolution, much like other engineering disciplines did decades ago.

Next, we list a number of research strands and discuss some of the problems that lie within their scope.

Infrastructure for requirements evolution. Research and practice on code evolution has produced a wealth of research concepts and tools. Version

control and configuration management, reverse engineering and visualization tools, refactoring and migration tools, among many. As indicated earlier, software of the future will consist not only of code and documentation, but also requirements and other types of models representing design, functionality and variability. Moreover, their interdependencies, for example, traceability links, will have to be maintained consistent and up-to-date for these artifacts to remain useful throughout a system's lifetime. Accordingly, the infrastructure for code evolution will have to be extended to accommodate these other kinds of artifacts. This is consistent with Model-Driven Software Engineering, as advocated by the Object Management Group (OMG).

Focusing on requirements, an infrastructure for requirements evolution will have to include tools for version control, configuration management and visualization. These tools will have to accommodate the kinds of models used to represent requirements. These models range from UML use cases that represent functional aspects of the system-to-be, all the way to goal models that capture stakeholder needs and rationalize any proposed functionality for the system-to-be. The problem of evolving traceability links from requirements to code has already been dealt with in the work of Jane Cleland-Huang and her colleagues (e.g., [44, 41, 45]).

Understanding root causes for change. We are interested here in characterizing generic root causes for change that dictate requirements evolution. For example, businesses are moving into network-based business models, such as service value networks and ecosystems. Such trends are bound to generate a host of new requirements on operational systems that will have to be addressed by requirements engineers and software reengineers. As another example, Governments around the world have been introducing legislation to address growing concerns for security, privacy, governance and safety. This makes regulatory compliance another major cause for requirements change. The introduction of a single Act in the US (Sarbanes-Oxley Act) in 2002 resulted in a monumental amount of change for business processes as well as software in business organizations. The costs of this change have been estimated at US$5.8B for one year alone (2005).

We would like to develop tools and techniques for systematically extracting requirements from laws and regulations. In tackling this research task, it is important to note that the concepts of law, such as "right" and "obligation", are not requirements. Consider a law about privacy that makes it an obligation for employers to protect and restrict the use of employee personal information stored in their databases. This obligation may be translated in many different ways into responsibilities of relevant actors so that the obligation is met. Each of these assignments of responsibility corresponds to a different set of requirements – i.e., stakeholder needs – that will have to be addressed by the software systems and the business processes of an organization.

This is a broad, inter-disciplinary and long-term research strand. Some research within its scope has already been done by Annie Anton, Travis Breaux

and colleagues, e.g., [46]. This is also the topic of Alberto Siena's PhD thesis, see [47] for early results.

Evolution mechanisms. Once we have identified what are the changes to requirements, we need to implement them by changing the system-at-hand. This may be done manually, possibly with tool support, by developing novel reengineering techniques. More interestingly, evolution may be done automatically by using mechanisms, inspired by different disciplines (Biology, Control Theory, Economics, Machine Learning, . . .). Doing research along this strand will require much experimentation to evaluate the effectiveness of different evolution techniques.

A number of research projects are working on design principles for autonomic and adaptive software systems (see, for example, on-going series of ICSE workshops on *Software Engineering for Adaptive and Self-Managing Systems*, http://www.hpi.uni-potsdam.de/giese/events/2008/seams2008/). Many of these projects employ a monitor-diagnose-compensate feedback loop in order to support adaptation of a system in response to undesirable changes of monitored data. The inclusion of such a feedback loop in support of adaptivity introduces the problem of designing monitoring, diagnosis and compensation mechanisms in the architecture of software systems. Control Theory offers a rich set of concepts of research results on how to design such loops in the realm of real-time continuous processes. Unfortunately, the development of such a theory for discrete systems is still in its early stages (though work has been done, see for example [48]).

Design for evolution. Some designs are better suited for evolution than others. For example, a design that can deliver a given functionality in many different ways is better than one that delivers it in a single way. Such designs are said to have *high variability*.

Variability is an important topic in many scientific disciplines that study variations among the members of a species, or a class of phenomena. In fact, the theory of evolution as presented by Darwin [49] holds that variability exists in the inheritable traits possessed by individual organisms of a species. This variability may result in differences in the ability of each organism to reproduce and survive within its environment. And this is the basis for the evolution of species. Note that a species in Biology corresponds to a high variability software system in Software Engineering, while an individual organism corresponds to a particular configuration of a high variability software system.

Variability has been studied in the context of product families [50], where variation points define choices that exist within the family for a particular feature of the family. The space of alternative members of a family can be characterized by a feature model [51]. Feature models capture variability in the *design space* of a product family, or a software system for that matter. They tell us what configurations of features are consistent and can co-exist within one configuration. For example, variation points may arise from the operating platform on which a family member will run (Windows, Linux, MacOS), or the weight of the

functionality offered (personal, business, pro). *Problem variability*, on the other hand, focuses on variability in the problem to be solved. For instance, scheduling a meeting may be accomplished by having the initiator contact potential participants to set a time and location. Alternatively, the initiator may submit her request to a meeting scheduler who does everything. The alternatives here characterize the structure of the problem to be solved and have nothing to do with features that the system-to-be will eventually have.

Designing for variability through analysis of both the problem and design space will remain a fruitful area of research with Requirements Engineering. See [32] for a PhD thesis that focuses on problem variability.

Variability of biological species changes over time, as variants are created through mutation or other mechanisms, while others perish. We need comparable mechanisms for software through which the set of possible instances for a software system changes over time. In particular, it is important to study two forms of variability change: means-based variability, and ends-based variability.

Means-based variability change leaves the ends/purpose of a software system unchanged, but changes the means through which the ends can be achieved. For example, consider a meeting scheduling system that offers a range of alternatives for meeting scheduling (e.g., user/system collects timetable constraints from participants, user/system selects meeting timeslot). Means-based variability may expand the ways meetings can be scheduled, for example, by adding a "meeting scheduling by decree" option where the initiator sets the time and expects participants to re-arrange their schedules accordingly.

Ends-based variability change, on the other hand, changes the purpose of the system itself. For instance, the meeting scheduler needs to be turned into a project management software system, or an office management toolbox. In this case, care needs to be exercised in managing scarce resources (e.g., rooms, people's time). Desai et al. [52] offers a promising direction for research on this form of variability change. Along a different path, Rommes and America [53] proposes a scenario-based approach to creating a product line architecture that does take into account possible long-term changes. through the use of strategic scenarios.

Modularity is another fundamental trait of evolvable software systems. Modularity has been researched throughly since the early 70s. A system is highly modular if it consists of components that have high (internal) cohesion and low (external) coupling. A highly modular system can have some of its components change with low impact on other components. Interestingly, Biology has also studied how coupling affects evolution. In particular, organisms in nature continuously co-evolve both with other organisms and with a changing abiotic environment. In this setting, the ability of one species to evolve is bounded by the characteristics of other species that it depends on. Accordingly, Kauffman [54] introduces the NKC model, named after the three main components that determine the behaviors of species' interaction with one another. According to the model, the co-evolution of a system and its environment is the equilibrium of external coupling and internal coupling. [55] presents a very preliminary

attempt to use this model to account for the co-evolution of software systems along with their environment.

Modularity and variability are clearly key principles underlying the ability of a species to evolve. It would be interesting to explore other principles that underlie evolvability.

There are deeper research issues where advances will have a major influence on solutions for the problem-at-hand. We mention three such issues:

Science of design. According to H. Simon's vision [56], a theory of design that encompasses at least three ingredients: (a) the purpose of an artifact, (b) the space of alternative designs, (c) the criteria for evaluating alternatives. Design artifacts that come with these ingredients will obviously be easier to evolve.

Model evolution. Models will be an important (perhaps the) vehicle for dealing with requirements evolution. Unfortunately, the state-of-the-art in modeling is such that models become obsolete very quickly, as their subject matter evolves. In Physics and other sciences, models of physical phenomena do not need to evolve because they capture invariants (immutable laws).

We either need here a different level of abstraction for modeling worlds of interest to design (usually technical, social and intentional), so that they capture invariants of the subject matter. Alternatively, we need techniques and infrastructures for model evolution as their subject matter changes.

Evolutionary design.[2] Extrapolating from Darwin's theory of evolution where design happens with no designer [57], we could think of mechanisms through which software evolves without any master purpose or master designer. An example of non-directed design is the Eclipse platform (eclipse.org). Rather than one centrally directed, purpose-driven technology, Eclipse has evolved into an ecology supporting multiple components, projects and people, leveraging the advantages of open-source licences. These software ecologies act as incubators for new projects with diverse characteristics. It would be fruitful to understand better the evolutionary processes taking place in these ecologies and invent other mechanisms for software evolution that do not involve a single master designer (also known as intelligent design in some places . . .) This is in sharp contrast to Simon's vision. At the same time, this is an equally compelling one.

4   Monitoring Requirements

Requirement monitoring aims to track a system's runtime behavior so as to detect deviations from its requirement specification. Fickas and Feather's work ([39, 40]) presents a run-time technique for monitoring requirements satisfaction. This technique identifies requirements, assumptions and remedies. If an assumption is violated, the associated requirement is denied, and the associated remedies are executed. The approach uses a Formal Language for Expressing Assumptions (FLEA) to monitor and alert the user of any requirement violations.

---

[2] . . . or, "Darwin's dangerous idea" [57].

Along similar lines, Robinson has proposed a requirements-monitoring framework named ReqMon [59]. In this framework, requirements are represented in the goal-oriented requirements modeling language KAOS [60] and through systematic analysis techniques, monitors are extracted that are implemented in commercial business process monitoring software.

We present an alternative approach to requirements monitoring and diagnosis. The main idea of the approach is to use goal models to capture requirements. From these, and on the basis of a number of assumptions, we can automatically derive monitoring specifications and generate diagnoses to recognize system failures. The proposal is based on diagnostic theories developed in AI, notably in Knowledge Representation and AI Planning research [61].

The monitoring component monitors requirements and generates log data at different levels of granularity that can be tuned adaptively depending on diagnostic feedback. The diagnostic component analyzes generated log data and identifies errors corresponding to aberrant system behaviors that lead to the violation of system requirements. When a software system is monitored with low granularity, the satisfaction of high level requirements is monitored. In this case, the generated log data are incomplete and many possible diagnoses can be inferred. The diagnostic component identifies the ones that represent root causes.

Software requirements models may be available from design-time, generated during requirements analysis, or they may be reverse engineered from source code using requirements recovery techniques (for example, Yu et al. [29]). We assume that bi-directional traceability links are provided, linking source code to the requirements they implement.

4.1   Preliminaries

Goal models have been used in Requirement Engineering (RE) to model and analyze stakeholder objectives [60]. Functional requirements are represented as hard goals, while non-functional requirements are represented as soft goals [62]. A goal model is a graph structure, where a goal can be AND- or OR- decomposed into subgoals and/or tasks. Means-ends links further decompose leaf level goals to tasks ("actions") that can be performed to fulfill them. At the source code level, tasks are implemented by simple procedures or composite components that are treated as black boxes for the purposes of monitoring and diagnosis. This allows a software system to be monitored at different levels of abstraction.

Following [63], if goal G is AND/OR decomposed into subgoals G_1, ..., G_n, then all/at-least-one of the subgoals must be satisfied for G to be satisfied. Apart from decomposition links, hard goals and tasks can be related to each other through MAKE(++) and BREAK(--) contribution links. If a MAKE (or a BREAK) link leads from goal G_1 to goal G_2, G_1 and G_2 share the same (or inversed) satisfaction/denial labels.

As an extension, we associate goals and tasks with preconditions and postconditions (hereafter *effects*, to be consistent with AI terminology) and monitoring switches. Preconditions and effects are propositional formulae, in Conjunctive

Normal Form (CNF), whose truth values are monitored and analyzed during diagnostic reasoning. Monitoring switches can be switched on/off to indicate whether satisfaction of the requirements corresponds to the goals/tasks is to be monitored at run time.

The propositional satisfiability (SAT) problem is concerned with determining whether there exists a truth assignment to variables of a propositional formula that makes the formula true. If such a truth assignment exists, the formula is said to be satisfiable. A SAT solver is any procedure that determines whether a propositional formula is satisfiable, and identifies the satisfying assignments of variables if it is.
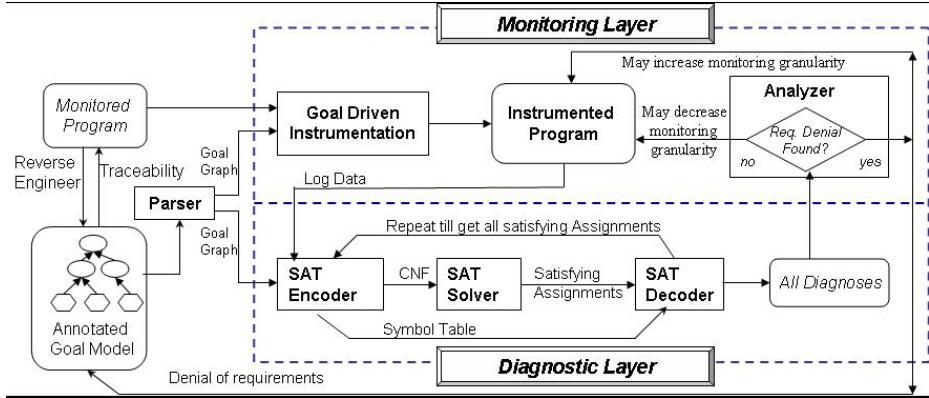
The earliest and most prominent SAT algorithm is DPLL (Davis-Putnam-Logemann-Loveland) [64]. Even though the SAT problem is inherently intractable, there have been many improvements to SAT algorithms in recent years. Chaff ([65]), BerkMin ([66]) and Siege ([67]) are among the fastest SAT solvers available today. Our work uses SAT4J ([68]), an efficient SAT solver that inherits a number of features from Chaff.

4.2   Framework Overview

Satisfaction of a software system's requirements can be monitored at different levels of granularity. Selecting a level involves a tradeoff between monitoring overhead and diagnostic precision. Lower levels of granularity monitor leaf level goals and tasks. As a result, more complete log data are generated, leading to more precise diagnoses. The disadvantage of fine-grained monitoring is high overhead and the possible degradation of system performance. Higher levels of granularity monitor higher level goals. Consequently, less complete log data are generated, leading to less precise diagnoses. The advantage is reduced monitoring overhead and improved system performance.

We provide for adaptive monitoring at different levels of granularity by associating monitoring switches with goals and tasks in a goal model. When these switches are turned on, satisfaction of the corresponding goals/tasks is monitored at run time. The framework adaptively selects a monitoring level by turning these switches on and off, in response to diagnostic feedback. Monitored goals/tasks need to be associated with preconditions and effects whose truth values are monitored and are analyzed during diagnostic reasoning. Preconditions and effects may also be specified for goals/tasks that are not monitored. This allows for more precise diagnoses by constraining the search space.

Figure 1 provides an overview of our monitoring and diagnostic framework. The input to the framework is the monitored program's source code, its corresponding goal model, and traceability links. From the input goal model, the parser component obtains goal/task relationships, goals and tasks to be monitored, and their preconditions and effects. The parser then feeds this data to the instrumentation and SAT encoder components in the monitoring and diagnostic layers respectively.

Fig. 1. Framework Overview

In the monitoring layer, the instrumentation component inserts software probes into the monitored program at the appropriate places. At run time, the instrumented program generates log data that contains program execution traces and values of preconditions and effects for monitored goals and tasks. Offline, in the diagnostic layer, the SAT encoder component transforms the goal model and log data into a propositional formula in CNF which is satisfied if and only if there is a diagnosis. A diagnosis specifies for each goal and task whether or not it is fully denied. A symbol table records the mapping between propositional literals and diagnosis instances. The SAT solver finds one possible satisfying assignment, which the SAT decoder translates into a possible diagnosis. The SAT solver can be repeatedly invoked to find all truth assignments that correspond to all possible diagnoses.

The analyzer analyzes the returned diagnoses, searching for denials of system requirements is found. If denials of system requirements are found, they are traced back to the source code to identify the problematic components. The diagnosis analyzer may then increase monitoring granularity by switching on monitoring switches for subgoals of a denied parent goal. When this is done, subsequent executions of the instrumented program generate more complete log data. More complete log data means fewer and more precise diagnoses, due to a larger SAT search space with added constraints. If no system requirements are denied, monitoring granularity may also be decreased to monitor fewer (thus higher level) goals in order to reduce monitoring overhead. The steps described above constitute one execution session and may be repeated.

4.3   Formal Foundations

This section presents an overview of the theoretical foundations of our framework. The theories underlying our diagnostic component (presented in section 4.2) are adaptations of the theoretical diagnostic frameworks proposed in [69,

70, 61]. Interested readers can refer to [2] for a complete and detailed account of the presented framework.

Log Data. Log data consists of a sequence of log instances, each associated with a specific timestep t. A log instance is either the observed truth value of a domain literal, or an occurrence of a particular task. We introduce predicate $occ_a(a_i, t)$ to specify occurrence of task a_i at timestep t. For example, if literal p is true at timestep 1, task a is executed at timestep 2, and literal q is false at timestep 3, their respective log instances are: $p(1)$, $occ_a(a, 2)$, and $\neg q(3)$.

Successful execution of tasks in an appropriate order leads to satisfaction of the root goal. A goal is satisfied in some execution secession s if and only if all the tasks under its decomposition are successfully executed in s. Goal satisfaction or denial may vary from session to session. The logical timestep t is incremented by 1 each time a new batch of monitored data arrives and is reset to 1 when a new session starts.

We say a goal has occurred in s if and only if all the tasks in its decomposition have occurred in s. Goal occurrences are not directly observable from the log data. Instead, our diagnostic component infers goal occurrence from task occurrences recorded in the log. Two timesteps, t_1 and t_2, are associated with goal occurrences, representing the timesteps of the first and the last executed task in the goal's decomposition in s. We introduce predicate $occ_g(g_i, t_1, t_2)$ to specify occurrences of goals g_i that start and end at timesteps t_1 and t_2 respectively. For example, suppose goal g is decomposed into tasks a_1 and a_2, and we have in the log data $occ_a(a_1, 4)$, $occ_a(a_2, 7)$ indicating that tasks a_1 and a_2 have occurred at timesteps 4 and 7 respectively. Then $occ_g(g, 4, 7)$ is inferred to indicate that g's occurrence started and ended at timesteps 4 and 7.

Theories of Diagnosis. The diagnostic component analyzes generated log data and infers satisfaction/denial labels for all the goals and tasks in a goal model. This diagnostic reasoning process involves two steps: (1), inferring satisfaction/denial labels for goals/tasks that are monitored; and (2), propagating these satisfaction/denial labels to the rest of the goal model. Note that if a goal/task is not monitored, but is associated with a precondition and an effect whose truth values are recorded in the log or can be inferred from it, then its satisfaction/denial is also inferred from step 1.

Intuitively, a goal g can be denied in one of three ways: (1) g itself can be denied, if it is monitored or if the truth values of its precondition and effect are known; or (2) one of g's children or parents is denied and the deniability is propagated to g through AND/OR decomposition links; or (3) one of the goals/tasks that are linked to g through MAKE(++)/BREAK(--) contribution links is denied/satisfied, in which case the denial label is propagated to g. As with goals, tasks get their denial labels if they themselves are denied, or if their parents are denied and denial labels are propagated down to them.

We reduced the problem of searching for a diagnosis to that of the satisfiability of a propositional formula Φ, where Φ is the conjunction of the following axioms:

(1) axioms for reasoning with goal/task denials (step 1); and (2) axioms for propagating inferred goal/task denials to the rest of the goal model (step 2).

Axiomatization of Deniability. The denial of goals and tasks is formulated in terms of the truth values of the predicates representing their occurrences, preconditions and effects. We introduce a distinct predicate FD to express full evidence of goal and task denial at a certain timestep or during a specific session. FD predicates take two parameters: the first parameter is either a goal or a task specified in the goal model, and the second parameter is either a timestep or a session id. For example, predicates $FD(g_1, 5)$ and $FD(a_1, s_1)$ indicate goal g_1 and task a_1 are denied at timestep 5 and session s_1 respectively.

Intuitively, if a task's precondition is true and the task occurred at timestep t, and if its effect holds at the subsequent timestep $t + 1$, then the task is not denied at timestep $t + 1$. Two scenarios describe task denial: (1)[3] if the task's precondition is false at timestep t, but the task still occurred at t; or (2) if the task occurred at timestep t, but its effect is false at the subsequent timestep $t + 1$. Task denial axioms are generated for tasks to capture both of these cases.

We illustrate task denial axioms using the following example. Consider a task a with precondition p and effect q. If the monitoring component generates one of the following two log data for a, task a's denial is inferred:

Log data 1: $\neg p(1); occ_a(a, 1)$

Log data 2: $p(1); occ_a(a, 1); \neg q(2)$

The first log data corresponds to the first task failure scenario: a's precondition p was false at timestep 1, but a still occurred at 1. The second log data corresponds to the second failure scenario: a's precondition was true and a occurred at timestep 1, but its effect q was false at the subsequent timestep 2. The diagnostic component infers $FD(a, 2)$ in both of these cases, indicating that task a has failed at timestep 2.

These failure scenarios also apply to goals. Recall that goal occurrences are indexed with two timesteps t_1 and t_2 that correspond to the occurrence timesteps of the first and last executed tasks under goal's decomposition. A goal g with precondition p and effect q is denied if and only if (1) goal occurrence started at t_1 when p is false; or (2) after goal occurrence finished at $t_2 + 1$, q is false.

For instance, if g is decomposed to tasks a_1 and a_2, the following sample log data correspond to the two failure scenarios for goal g:

Log data 3: $\neg p(1); occ_a(a_1, 1); occ_a(a_2, 2)$

Log data 4: $p(1); occ_a(a_1, 1); occ_a(a_2, 2); \neg q(3)$

From either of the two log data, the diagnostic component infers $occ_g(g, 1, 2)$, indicating that g's occurrence started and ended at timesteps 1 and 2 respectively. Log data 3 and 4 correspond to the first and second goal failure scenarios respectively: p is false when g's occurrence started at timestep 1, and q is false after g's occurrence at timestep 3. In either of these cases, the diagnostic component infers $FD(g, 3)$, indicating that goal g is denied at time step 3.

---

[3] In many axiomatizations it is assumed that $occ_a(a, t) \rightarrow p(t)$.

We say a goal or a task is denied during an execution session s if the goal/task is denied at some timestep t within s. Returning to the above examples, if $FD(a,2)$ and $FD(g,3)$ are inferred, and if timesteps 2 and 3 fall within execution session s_1, the diagnostic component further infers $FD(a,s_1)$ and $FD(g,s_1)$. Inferring goal/task denials for an execution session is useful for efficiently propagating these denial labels to the rest of the goal model.

In the AI literature, propositional literals whose values may vary from timestep to timestep are called *fluents*. A fluent f can take on any arbitrary value at timestep $t+1$ if it is not mentioned in the effect of a task that is executed at timestep t. Axioms are needed to specify that unaffected fluents retain the same the values from timestep to timestep. An axiom is generated to specify that if the value of a fluent f changes at timestep t, then one of the tasks/goals that has f in its effect must have occurred at $t-1$ and not have been denied at t. In other words, the truth value of f reminds constant from one timestep to the next, until one of the actions/goals that have f in its effect is executed successfully. For example, consider a task a with effect q, and assume q is not in any other goal's/task's effect. Suppose the log data include: $\neg q(1)$, $occ_a(a,3)$, and $q(5)$. Then an axiom is generated to infer $\neg q(2)$, $\neg q(3)$, and $q(4)$.

4.4   Axiomatization of a Goal Model

Goal/task denials, once inferred, can be propagated to the rest of the goal graph through AND/OR decomposition links and MAKE/BREAK contribution links. Axioms are generated to describe both label propagation processes.

If a goal g is AND (or OR) decomposed into subgoals $g_1,\ldots, g_n$, and tasks $a_1, \ldots, a_m$, then g is denied in a certain session, s, if and only if at least one (or all) of the subgoals or tasks in its decomposition is (or are) denied in s.

Goals and tasks can be related to each other through various contribution links: ++S, --S, ++D, --D, ++, --. Link ++ and link -- are shorthand for the ++S and ++D, and the --S and --D relationships, respectively, and they represent strong MAKE(++) and BREAK(--) contributions between goals/tasks. Given two goals g_1 and g_2, the link $g_1 \xrightarrow{++S} g_2$ (respectively $g_1 \xrightarrow{--S} g_2$) means that if g_1 is satisfied, then g_2 is satisfied (respectively denied). But if g_1 is denied, we cannot infer denial (or respectively satisfaction) of g_2. The meanings of links ++D and --D are similar to those of ++S and --S. Given two goals g_1 and g_2, the link $g_1 \xrightarrow{++D} g_2$ (respectively $g_1 \xrightarrow{--D} g_2$) means that if g_1 is denied, then g_2 is denied (respectively satisfied). But if g_1 is satisfied, we cannot infer satisfaction (or respectively denial) of g_2.

When contribution links are present, the goal graph may become cyclic and conflicts may arise. We say a conflict holds if we have both $FD(g,s)$ and $\neg FD(g,s)$ in one execution session s. Since it does not make sense, for diagnostic purposes, to have a goal being both denied and satisfied at the same time, conflict tolerance, as in (Sebastiani et al., 2004), is not allowed within our diagnostic framework. In addition, the partial (weaker) contribution links HELP(+) and

HURT($-$) are not included between hard goals/tasks because we do not reason with partial evidence for hard goal/task satisfaction and denial.

Diagnosis Defined. In our framework, a diagnosis specifies for each goal/task in the goal model whether or not it is fully denied. More formally, a diagnosis D is a set of FD and $\neg FD$ predicates over all the goals and tasks in the goal graph, such that D union Φ ($D \cup \Phi$) is satisfiable. Each FD or $\neg FD$ predicate in D is either indexed with respect to a timestep or a session. For example, if goal g and task a are both denied at timestep 1 during execution session s_1, the diagnosis for the system would contain $FD(a,1)$, $FD(a,s_1)$, $FD(g,1)$, and $FD(g,s_1)$.

Our diagnostic approach is sound and complete, meaning that for any D as defined above, D is a diagnosis if and only if $D \cup \Phi$ is satisfiable. A proof of this soundness and completeness property can be found in [2].

Task level denial is the core or root cause of goal level denial. In addition, if a task is denied at any timestep t during an execution session s, it is denied during s. Therefore, it is more useful, for purposes of root cause analysis, that the diagnostic component infer task level denials during specific sessions. We introduce the concept of core diagnosis to specify for each task in the goal graph whether or not it is fully denied in an execution session. More formally, a core diagnosis (CD) is a set of FD and $\neg FD$ predicates over all the tasks in the goal graph, indexed with respect to a session, such that $CD \cup \Phi$ is satisfiable. Consider the same example where goal g and task a are denied at timestep 1 during the execution session s_1. The core diagnosis for the system would only contain $FD(a,s_1)$, indicating that the root cause of requirement denial during s_1 is the failure of task a.

Inferring all core diagnoses for the software system can present a scalability problem. This is because all the possible combinations of task denials for tasks under a denied goal are returned as possible core diagnoses. Therefore, in the worst-case, the number of core diagnoses is exponential to the size of the goal graph. To address the scalability problem, we introduce the concept of *participating diagnostic components*. These correspond to individual task denial predicates that participate in core diagnoses, without their combinations. A participating diagnostic component, PDC, is an FD predicate over some task in the goal model, indexed with respect to a session, such that $PDC \cup \Phi$ is satisfiable.

In many cases, it may be neither practical nor necessary to find all core diagnoses. In these cases, all participating diagnostic components can be returned. However, it is also important to note that, in other cases, one may want to find all core diagnoses instead of all participating diagnostic components. This is because core diagnoses contain more diagnostic information, such as which tasks can and can not fail together.

Our diagnostic approach is sound and complete, meaning that it finds *all* diagnoses, core diagnoses, and participating diagnostic components for the software system. The theory outlined above has been implemented in terms of four main algorithms: two encoding algorithms for encoding an annotated goal model

into a propositional formula Φ, and two diagnostic algorithms for finding all core diagnoses and all participating diagnostic components.
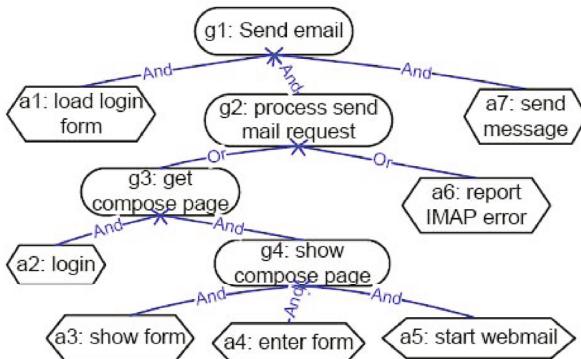
The difference between the two encoding algorithms lies in whether the algorithm preprocesses the log data when encoding the goal model into Φ. The naive algorithm does not preprocess log data and generates a complete set of axioms for all the timesteps during one execution session. The problem with this is the exponential increase in the size of Φ with the size of a goal model. The second and improved algorithm addresses this problem by preprocessing the log data and only generating necessary axioms for the timesteps that are actually recorded in the log data. As demonstrated in [2], this improved algorithm permits the same diagnostic reasoning process while keeping the growth of the size of Φ polynomial with respect to the size of the goal model.

The results of our framework evaluation (subsection 4.6) show that our approach scales to the size of the goal model, provided the encoding is done with log file preprocessing and the diagnostic component returns all participating diagnostic components instead of all core diagnoses. Interested readers can refer to [2] for a detailed account of algorithms and implementation specifics.

4.5   A Working Example

We use the SquirrelMail [71] case study as an example to illustrate how our framework works. SquirrelMail is an open source email application that consists of 69711 LOC written in PHP. Figure 2 presents a simple, high-level goal graph for SquirrelMail with 4 goals and 7 tasks, shown in ovals and hexagons, respectively.

The SquirrelMail goal model captures the system's functional requirements for sending an email (represented by the root goal g_1). The system first needs to retrieve and load user login page (task a_1), then process the sent mail request (goal g_2), and finally send the email (task a_7). If the email IMAP server is found, SquirrelMail loads the compose page (goal g_3), otherwise, it reports IMAP not

Fig. 2. Squirrel Mail Goal Model

Table 1. Squirrel Mail Annotated Goal Model

Goal/Task	Monitor switch	Precondition	Effect
a1	on	correctURL entered	login form loaded
a2	on	¬wrongIMAP∧login form loaded	(user logged in ∧ correct pin) ∨ (¬user logged in ∧ ¬correct pin)
a3	off	user logged in	form shown
a4	off	form shown	form entered
a5	off	form entered	webmail started
a6	on	wrongIMAP	error reported
a7	on	webmail started	email sent
g1	off	correct URL entered	email sent ∨ error reported
g2	off	login form loaded ∨ wrongIMAP	webmail started ∨ error reported
g3	off	login form loaded ∧ ¬wrongIMAP	webmail started
g4	on	user logged in	webmail started
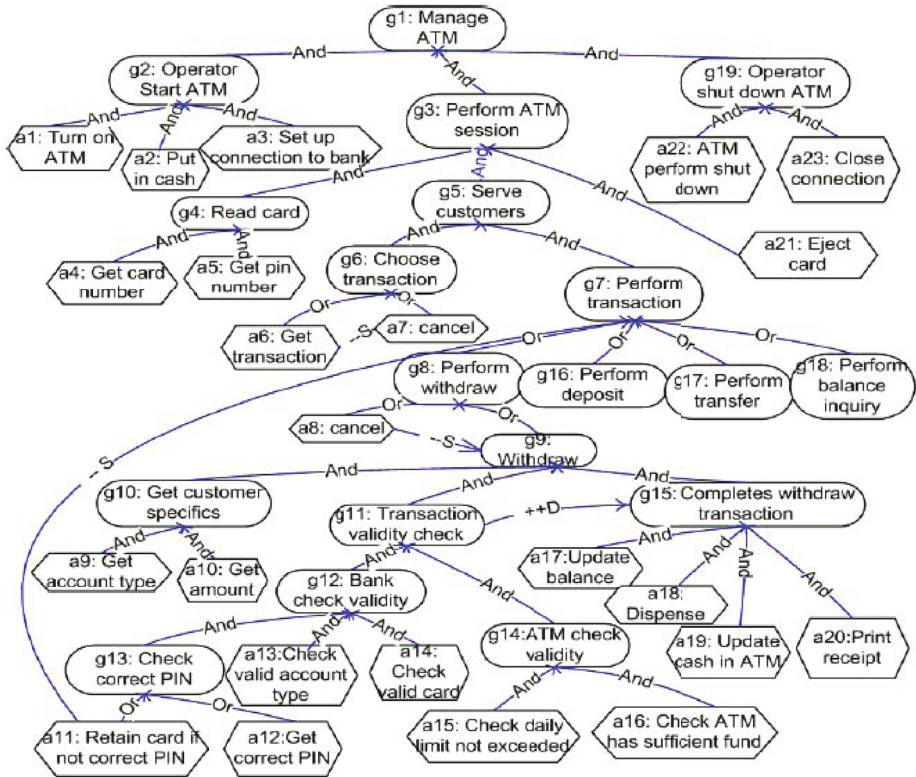
found error (task a_6). Goal g_3 (*get compose page*) can be achieved by executing four tasks: a_2 (*login*), a_3 (*show form*), a_4 (*enter form*), and a_5 (*start webmail*).

Table 1 lists the details of each goal/task in the SquirrelMail goal model with its monitoring switch status (column 2), and associated precondition and effect (columns 3 and 4). In this example, the satisfaction of goal $g4$ and tasks a_1, a_2, a_6, and a_7 are monitored.

SquirrelMail's runtime behavior is traced and recorded as log data. Recall that log data contains truth values of literals specified in monitored goals'/tasks' preconditions and effects, as well as the occurrences of all tasks. Each log instance is associated with a timestep t. The following is an example of log data from the SquirrelMail case study:

$correct\ URL\ entered(1)$, $occ_a(a_1, 2)$, $login\ form\ loaded(3)$, $¬wrongIMAP$ $(4), occ_a(a_2, 5)$, $correct\ pin(6), user\ logged\ in(6)$, $occ_a(a_3, 7)$, $occ_a(a_4, 8)$, $occ_a(a_5, 9), ¬webmail\ started(10)$, $occ_a(a_7, 11)$, $¬email\ sent(12)$.

The log data contains two errors ($¬webmail\ started(10)$, and $occ_a(a_7, 11)$): (1) the effect of g_4 (web mail started) was false, at timestep 10, after all the tasks under g_4's decomposition (a_3, a_4, and a_5) were executed; and (2) task a_7 (*send message*) occurred at timestep 11 when its precondition webmail started was false at timestep 10. The diagnostic component analyzes the log data and infers that goal g_4 and the task a_7 are denied during execution session s. The diagnostic component further infers that if g_4 is denied in s, at least one of g_4's subtasks, a_3, a_4, and a_5, must have been denied in s. The following seven core diagnoses are returned to capture all possible task denials for a_3, a_4, and a_5:

Fig. 3. Partial ATM Goal Model

Core Diagnosis 1: $FD(a_3, s); FD(a_7, s)$
Core Diagnosis 2: $FD(a_4, s); FD(a_7, s)$
Core Diagnosis 3: $FD(a_5, s); FD(a_7, s)$
Core Diagnosis 4: $FD(a_3, s); FD(a_4, s); FD(a_7, s)$
Core Diagnosis 5: $FD(a_3, s); FD(a_5, s); FD(a_7, s)$
Core Diagnosis 6: $FD(a_4, s); FD(a_5, s); FD(a_7, s)$
Core Diagnosis 7: $FD(a_3, s); FD(a_4, s); FD(a_5, s); FD(a_7, s)$

Instead of finding all core diagnoses, we can configure the diagnostic component to find all participating diagnostic components. The following 4 participating diagnostic components are returned to capture individual task denials:

Participating Diagnostic Component 1: $FD(a_3, s)$
Participating Diagnostic Component 2: $FD(a_4, s)$
Participating Diagnostic Component 2: $FD(a_5, s)$
Participating Diagnostic Component 3: $FD(a_7, s)$

4.6   Experimental Evaluation

In this section, we report on the performance and scalability of our framework and discuss its limitations. We applied our framework to a medium-size public domain software system, an ATM (Automated Teller Machine) simulation case study, to evaluate the correctness and performance of our framework. We show that our solution can scale up to the goal model size and can be applied to industrial software applications with medium-sized requirements.

Framework Scalability. The ATM simulation case study is an illustration of OO design used in a software development class at Gordon College [72]. The application simulates an ATM performing customers' *withdraw, deposit, transfer* and *balance inquiry* transactions. The source code contains 36 Java Classes with 5000 LOC, which we reverse engineered to its requirements to obtain a goal model with 37 goals and 51 tasks. We show a partial goal graph with 18 goals and 22 tasks in Figure 3.

We conducted two sets of experiments. The first set contains five experiments with different levels of monitoring granularity, all applied to the goal model shown in Figure 3. This allows us to access the tradeoff between monitoring granularity and diagnostic precision. The second set reports 20 experiments on 20 progressively larger goal models containing 50 to 1000 goals and tasks. We obtain these larger goal models by cloning the ATM goal graph to itself. The second set of experiments shows that our diagnostic framework scales to the size of the relevant goal model, provided the encoding is done with log preprocessing and the diagnostic component returns all participating diagnostic components.

The first set of experiments contains 5 runs. We gradually increased monitoring granularity from monitoring only the root goal to monitoring all leaf level tasks. For each experiment, we recorded: (1) numbers of generated literals and clauses in the SAT propositional formula Φ; (2) the number of participating diagnostic components returned; and (3) the average time taken, in seconds, to find one diagnostic component. When the number of monitored goals/tasks was increased from 1 to 11, the number of returned participating diagnostic components decreased from 19 and 1, and the average time taken to find one diagnostic component increased from 0.053 to 0.390 second.

These experiments showed that diagnostic precision is inversely proportional to monitoring granularity. When monitoring granularity increases, monitoring overhead, SAT search space, and average time needed to find a single participating diagnostic component all increase. The benefit of monitoring at a high level of monitoring granularity is that we are able to infer fewer participating diagnostic components identifying a smaller set of possible faulty components. The reverse is true when monitoring granularity decreases: we have less overhead, but the number of participating diagnostic components increases if the system is behaving abnormally. When the system is running correctly (no requirements are denied, and no faulty component is returned), minimal monitoring is advisable.

The second set of experiments, on 20 progressively larger goal models (containing from 50 to 1000 goals and tasks) allows us to evaluate the scalability of the diagnostic component. We injected one error in one of the tasks. Each

of the experiments was performed with complete (task level) monitoring. Each therefore returned only a single diagnostic component. In addition, all experiments used the encoding algorithm that preprocesses log data. This was done to ensure scalability. For each experiment, we recorded: (1) time taken to encode the goal model into the SAT propositional formula Φ; (2) time taken by the SAT solver to solve Φ plus the time taken to decode the SAT result into a diagnostic component; and (3) the sum of the time periods recorded in (1) and (2), giving the total time taken to find the participating diagnostic component.

Experimental results show that, as the number of goals/tasks increased from 50 to 1000, the number of literals and clauses generated in Φ increased from 81 to 1525 and from 207 to 4083 respectively. As a result, the total time taken to find the participating diagnostic component increased from 0.469 to 3.444 seconds. This second set of experiments shows that the diagnostic component scales to the size of the goal model, provided the encoding is done with log preprocessing and the diagnostic component returns all participating diagnostic components. Our approach can therefore be applied to industrial software applications with medium-sized requirement graphs.

Framework Limitations. Firstly, our approach assumes the correct specification of the goal model, as well as the preconditions and effects for goals and tasks. Errors may be introduced if specified preconditions and effects do not completely or correctly capture the software system's dynamics. Detecting and dealing with discrepancies between a system's implementation and its goal model are beyond the scope of our work. We accordingly, assume that both the goal model and its associated preconditions and effects are correctly implemented by the application source code.

Secondly, the reasoning capability of our diagnostic component is limited by the expressive power of propositional logic and the reasoning power of SAT solvers. Propositional logic and SAT solvers express and reason using variables with discrete values, which typically are Boolean variables that are either true or false. As a result, our diagnostic component cannot easily deal with application domains with continuous values.

Lastly, the reasoning power of our framework is also limited by the expressiveness of our goal modeling language. Goal models cannot express temporal relations. Neither can they explicitly express the orderings of goals/tasks, or the number of times goals/tasks must be executed. Therefore, our framework cannot recognize temporal relations such as event patterns.

5   Conclusions

We have discussed requirements evolution as a research problem that has received little attention until now, but will receive much attention in the future. Our discussion included a review of past research, a speculative glimpse into the future, and a more detailed look at on-going research on monitoring and diagnosing software systems.

References

[1] Lubars, M., Potts, C., Richter, C.: A review of the state-of-practice in requirements modelling. In: Intl. Symp. on Requirements Engineering, San Diego, CA (January 1993)

[2] Wang, Y., McIlraith, S., Yu, Y., Mylopoulos, J.: An automated approach to monitoring and diagnosing requirements. In: International Conference on Automated Software Engineering (ASE 2007), Atlanta, GA (October 2007)

[3] Lehman, M.: On understanding laws, evolution, and conservation in the large-program life cycle. Journal of Systems and Software 1, 213–221 (1980)

[4] Brooks, F.: The mythical man-month. Addison-Wesley, Reading (1975)

[5] Fernandez-Ramil, J., Perry, D., Madhavji, N.H. (eds.): Software Evolution and Feedback: Theory and Practice, 1st edn. Wiley, Chichester (2006)

[6] Gîrba, T., Ducasse, S.: Modeling history to analyze software evolution. Journal of Software Maintenance and Evolution: Research and Practice 18(3), 207–236 (2006)

[7] Xing, Z., Stroulia, E.: UMLDiff: an algorithm for objectoriented design di erencing. In: Intl. Conf. on Automated Software Engineering, Long Beach, CA, USA, pp. 54–65 (2005)

[8] Basili, V.R., Weiss, D.M.: Evaluation of a software requirements document by analysis of change data. In: Intl. Conf. on Software Engineering, San Diego, USA, pp. 314–323 (1981)

[9] Basili, V.R., Perricone, B.T.: Software errors and complexity: An empirical investigation. Commun. ACM 27(1), 42–52 (1984)

[10] Harker, S.D.P., Eason, K.D., Dobson, J.E.: The change and evolution of requirements as a challenge to the practice of software engineering. In: IEEE International Symposium on Requirements Engineering, pp. 266–272 (1993)

[11] Rajlich, V.T., Bennett, K.H.: A staged model for the software life cycle. IEEE Computer 33(7), 66–71 (2000)

[12] Lientz, B.P., Swanson, B.E.: Software Maintenance Management: A Study of the Maintenance of Computer Application Software in 487 Data Processing Organizations. Addison-Wesley, Reading (1980)

[13] Rowe, D., Leaney, J., Lowe, D.: De ning systems evolvability - a taxonomy of change. In: International Conference and Workshop: Engineering of Computer-Based Systems, Maale Hachamisha, Israel, p. 45+ (1998)

[14] Vicente, K.J.: Ecological interface design: Progress and challenges. Human Factors 44, 62–78 (2002)

[15] Favre, J.-M.: Meta-model and model co-evolution within the 3d software space. In: Intl. Workshop on Evolution of Large-scale Industrial Software Applications at ICSM, Amsterdam (September 2003)

[16] Swanson, B.E.: The dimensions of maintenance. In: Intl. Conf. on Software Engineering, San Francisco, California, pp. 492–497 (1976)

[17] Svensson, H., Host, M.: Introducing an agile process in a software maintenance and evolution organization. In: European Conference on Software Maintenance and Reengineering, Manchester, UK, March 2005, pp. 256–264 (2005)

[18] Nelson, K.M., Nelson, H.J., Ghods, M.: Technology exibility: conceptualization, validation, and measurement. In: International Conference on System Sciences, Hawaii, vol. 3, pp. 76–87 (1997)

[19] Buckley, J., Mens, T., Zenger, M., Rashid, A., Kniesel, G.: Towards a taxonomy of software change. Journal of Software Maintenance and Evolution: Research and Practice 17(5), 309–332 (2005)

[20] Chapin, N., Hale, J.E., Fernandez-Ramil, J., Tan, W.-G.: Types of software evolution and software maintenance. Journal of Software Maintenance and Evolution: Research and Practice 13(1), 3–30 (2001)

[21] Felici, M.: Taxonomy of evolution and dependability. In: Proceedings of the Second International Workshop on Unanticipated Software Evolution, USE 2003, Warsaw, Poland, April 2003, pp. 95–104 (2003)

[22] Felici, M.: Observational Models of Requirements Evolution. PhD thesis, University of Edinburgh (2004),
http://homepages.inf.ed.ac.uk/mfelici/doc/IP040037.pdf

[23] Sommerville, I., Sawyer, P.: Requirements Engineering: A Good Practice Guide. John Wiley & Sons, New York (1997)

[24] Lam, W., Loomes, M.: Requirements evolution in the midst of environmental change: A managed approach. In: Euromicro. Conf. on Software Maintainance and Reengineering, Florence, Italy, March 1998, pp. 121–127 (1998)

[25] Stark, G., Skillicorn, A., Ameele, R.: An examination of the e ects of requirements changes on software releases. Crosstalk: Journal of Defence Software Engineering, 11–16 (December 1998)

[26] Lormans, M., van Dijk, H., van Deursen, A., Nocker, E., de Zeeuw, A.: Managing evolving requirements in an outsourcing context: an industrial experience report. In: International Workshop on Principles of Software Evolution, pp. 149–158 (2004)

[27] Wiegers, K.E.: Automating requirements management. Software Development Magazine 7(7) (July 1999)

[28] Roshandel, R., Van Der Hoek, A., Mikic-Rakic, M., Medvidovic, N.: Mae – a system model and environment for managing architectural evolution. ACM Trans. Softw. Eng. Methodol. 13(2), 240–276 (2004)

[29] Yu, Y., Wang, Y., Mylopoulos, J., Liaskos, S., Lapouchnian, A.: Reverse engineering goal models from legacy code. In: International Conference on Requirements Engineering, Paris, pp. 363–372 (September 2005)

[30] Niu, N., Easterbrook, S., Sabetzadeh, M.: A categorytheoretic approach to syntactic software merging. In: 21st IEEE International Conference on Software Maintenance (ICSM 2005), Budapest, Hungary (September 2005)

[31] Berry, D.M., Cheng, B.H.C., Zhang, J.: The four levels of requirements engineering for and in dynamic adaptive systems. In: International Workshop on Requirements Engineering: Foundation for Software Quality, Porto, Portugal (June 2005)

[32] Liaskos, S.: Acquiring and Reasoning about Variability in Goal Models. PhD thesis, University of Toronto (2008)

[33] Jureta, I., Faulkner, S., Thiran, P.: Dynamic requirements specification for adaptable and open service-oriented systems. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 270–282. Springer, Heidelberg (2007)

[34] Etien, A., Salinesi, C.: Managing requirements in a co-evolution context. In: 13th IEEE International Conference on Requirements Engineering, Paris, pp. 125–134 (September 2005)

[35] Boehm, B.: Some future trends and implications for systems and software engineering processes. Systems Engineering 9(1), 1–19 (2006)

[36] Yang, Y., Bhuta, J., Boehm, B., Port, D.N.: Value-based processes for cots-based applications. IEEE Software 22(4), 54–62 (2005)

[37] Nuseibeh, B.: Weaving together requirements and architectures. IEEE Computer 34(3), 115–119 (2001)

[38] Gotel, O.C.Z., Finkelstein, C.W.: An analysis of the requirements traceability problem. In: International Conference on Requirements Engineering, pp. 94–101 (1994)

[39] Fickas, S., Feather, M.S.: Requirements monitoring in dynamic environments. In: RE 1995: Proceedings of the Second IEEE International Symposium on Requirements Engineering, Washington, DC, USA (1995)

[40] Feather, M.S., Fickas, S., van Lamsweerde, A., Ponsard, C.: Reconciling system requirements and runtime behaviour. In: Ninth IEEE International Workshop on Software Specification and Design (IWSSD-9), Isobe, JP, pp. 50–59 (1998)

[41] Cleland-Huang, J., Settimi, R., BenKhadra, O., Berezhanskaya, E., Christina, S.: Goal-centric traceability for managing non-functional requirements. In: Intl. Conf. Software Engineering, St. Louis, MO, USA, pp. 362–371 (2005)

[42] Ramesh, B., Jarke, M.: Toward reference models for requirements traceability. IEEE Transactions on Software Engineering 27(1), 58–93 (2001)

[43] Kephart, J., Chess, D.: The vision of autonomic computing. IEEE Computer 36(1) (January 2003)

[44] Cleland-Huang, J., Chang, C.K., Christensen, M.: Event-based traceability for managing evolutionary change. Transactions on Software Engineering 29(9), 796–810 (2003)

[45] Cleland-Huang, J., Settimi, R., Zou, X., Solc, P.: Automated classi cation of non-functional requirements. Requirements Engineering 12(2), 103–120 (2007)

[46] Breaux, T., Antón, A.: Analyzing goal semantics for rights, permissions, and obligations. In: International Requirements Engineering Conference, Paris, France, pp. 177–186 (August 2005)

[47] Siena, A., Maiden, N., Lockerbie, J., Karlsen, K., Perini, A., Susi, A.: Exploring the effectiveness of normative i* modelling: Results from a case study on food chain traceability. In: Bellahsène, Z., Léonard, M. (eds.) CAiSE 2008. LNCS, vol. 5074, pp. 182–196. Springer, Heidelberg (2008)

[48] Ramadge, P., Wonham, M.: Supervisory control of a class of discreteevent systems. SIAM J. of Control and Optimization 25(1), 206–230 (1987)

[49] Darwin, C.: On the Origin of Species by Means of Natural Selection. Murray, London (1859)

[50] Svahnberg, M., van Gurp, J., Bosch, J.: A taxonomy of variability realization techniques. Softw. Pract. Exper. 35(8), 705–754 (2005)

[51] Kang, K.C., Kim, S., Lee, J., Kim, K.: Form: A feature-oriented reuse method. Annals of Software Engineering 5, 143–168 (1998)

[52] Desai, N., Chopra, A.K., Singh, M.P.: Amoeba: A methodology for requirements modeling and evolution of crossorganizational business processes. Transactions on Software Engineering and Methodology (submitted, 2008)

[53] Rommes, E., America, P.: A scenario-based method for software product line architecting. In: Käkölä, T., Dueñas, J.C. (eds.) Software Product Lines - Research Issues in Engineering and Management, Berlin, pp. 3–52 (2006)

[54] Kau, S.A.: The Origins of Order: Self-Organization and Selection in Evolution. Oxford University Press, Oxford (1993)

[55] Su, N., Mylopoulos, J.: Evolving organizational information systems with tropos. In: Conference on Advanced Information Systems Engineering (2006)

[56] Simon, H.A.: The Sciences of the Artificial, 3rd edn. MIT Press, Cambridge (1996)

[57] Dennett, D.C.: Darwin's Dangerous Idea: Evolution and the Meanings of Life. Simon & Schuster, New York (1995)

[58] Feather, M.S.: Rapid application of lightweight formal methods for consistency analysis. IEEE Trans. Softw. Eng. 24(11), 948–959 (1998)

[59] Robinson, W.N.: A requirements monitoring framework for enterprise systems. Requirements Engineering Journal 11, 17–41 (2006)

[60] Dardenne, A., van Lamsweerde, A., Fickas, S.: Goal-directed requirements acquisition. Science of Computer Programming 20(1-2), 3–50 (1993)

[61] McIlraith, S.: Explanatory diagnosis: Conjecturing actions to explain observations. In: International Conference on Principles of Knowledge Representation and Reasoning (KR 1998), Trento, Italy, June 1998, pp. 167–179 (1998)

[62] Mylopoulos, J., Chung, L., Nixon, B.: Representing and using non-functional requirements: A process-oriented approach. IEEE Transactions on Software Engineering 18(6), 483–497 (1992)

[63] Giorgini, P., Mylopoulos, J., Nicchiarelli, E., Sebastiani, R.: Reasoning with goal models. In: Spaccapietra, S., March, S.T., Kambayashi, Y. (eds.) ER 2002. LNCS, vol. 2503, pp. 167–181. Springer, Heidelberg (2002)

[64] Davis, M., Logemann, G., Loveland, D.: A machine program for theorem proving. Journal of ACM 5, 394–397 (1962)

[65] Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: engineering an efficient sat solver. In: Design Automation Conference, Las Vegas, pp. 530–535 (June 2001)

[66] Goldberg, E., Novikov, Y.: Berkmin: A fast and robust sat-solver. In: Conference on Design, Automation and Test in Europe (DATE), Paris, pp. 142–149 (March 2002)

[67] Ryan, L.: Efficient algorithms for clause-learning SAT solvers. Master's thesis, Simon Fraser University (2004)

[68] Le Berre, D.: A satisfiability library for java (2007), `http://www.sat4j.org/`

[69] Reiter, R.: A theory of diagnosis from first principles. Artificial Intelligence 32(1), 57–95 (1987)

[70] De Kleer, J., Mackworth, A.K., Reiter, R.: Characterizing diagnoses and systems. Artificial Intelligence 56(2-3), 197–222 (1992)

[71] Castello, R.: Squirrel mail (2007), `http://www.squirrelmail.org/`

[72] Bjork, R.: An example of object-oriented design: an atm simulation (2007), http://www.cs.gordon.edu/courses/cs211/ATMExample/index.html/

Designs Can Talk: A Case of Feedback for Design Evolution in Assistive Technology

William N. Robinson[1] and Stephen Fickas[2]

[1] Computer Information Systems, Georgia State University, Atlanta, Georgia, USA
[2] Computer Science, University of Oregon, Eugene, Oregon, USA

Abstract. Requirements engineers gain insights and make improvements on their requirements specifications, as they are applied in natural contexts. Software artifacts are particularly useful requirements instantiations because feedback can be obtained directly from software. Software talks to its designers about its requirements.

We illustrate requirements feedback with a case study in assistive technology (AT). A specialized emailing system was designed for cognitively impaired patients in an effort to decrease their social isolation, which often occurs after a brain injury. The patients continue to expand their email system usage, which is remarkable for AT. We attribute this unusual success to the feedback obtained directly from the software, through monitoring user goal models. Such monitoring has allowed the developers to understand and evolve their software to meet the changing user needs. It illustrates how an operational artifact, like software, can drive design evolution.

Keywords: requirements monitoring, evolution, design science.

1   Introduction

The *Think and Link* (TAL) project developed a specialized email system as part of a clinical treatment package[1-5]. In the clinical setting, individuals are assessed, individual goals are acquired, each individual is given a tailored treatment package, the effectiveness of the deployed package is tracked for each individual, and mid-course corrections can ensue. Currently, the treatment packages delivered in clinical fields may have a software component, but this software is part of the clinical domain, targeted to professionals providing treatment support. In particular, the clinical software used has little or nothing to do with daily-living software *applications* that many of us take for granted; for example, email, web browsers, music-management tools, *etc*. In contrast, the TAL email software is part of the patient's treatment plan.

In TAL, the email software is personalized uniquely to each individual, as required by his or her treatment plan. Personalization is accomplished through evolution; the system continuously monitors user behaviors, and ensures that proper software adaptations are made to accommodate changing user needs. This is working wonderfully. In contrast to most assistive technologies, patients do not abandon the TAL email system. In fact, their emailing skills grow with usage, and in response, their unique email interfaces require continuous updates.

In the commercial version of TAL, much of user monitoring is a manual process. This approach is not feasible for a large patient population. To address the scaling issue, we sought to automate much of monitoring in the research version of TAL. With the objective of monitoring the progress of each patient's treatment progress, we sought an existing software solution. Having found none adequate to the task, we applied a design-science research methodology to design and develop a new monitoring method. This chapter describes that design-science research project.

We have developed a new automated method for user monitoring that reduces the workload of TAL researchers in their efforts to monitor and adapt the TAL email system. The method may be applicable to more general problems of automating continuous monitoring and adaptation of software services.

In this introductory section, we summarize two alternative views of design that require monitoring: the science of design and reflective practice. These two views are related to the fundamental ontology of requirements engineering and their application to software personalization. After introducing these theoretic concerns, we turn to the design-science research project that produced the TAL monitoring system; each of the common design-science activities is described. Finally, we draw conclusions about how requirements monitoring can support the design views espoused by Simon and Schön.

1.1 Science of Design

Simon is attributed with defining fundamentals of the SoD, especially its supporting curriculum. According to Simon, the science of design is about designing artificial things that are synthesized, and characterized in terms of functions, goals, and adaptations[6]—p 5. Artifacts are designed (synthesized) to fit and adapt within a specific environment.

> An artifact can be thought of as a meeting point—an interface in today's terms between an "inner" environment, the substance and organization of the artifact itself, and an "outer" environment, the surroundings in which it operates. If the inner environment is appropriate to the outer environment, or vice versa, the artifact will serve its intended purpose.[6]—p 6.

Designing is not synonymous with wishing. Design desires are tempered by what is feasible in the natural world.

> [A] design has not been achieved until we have discovered at least one realizable inner system obeying the ordinary natural laws. [6]—p. 12.

Although designed artifacts are artificial, they may substitute for natural objects, according to their limitations.

> The artificial object imitates the real by turning the same face to the outer system, by adapting, relative to the same goals, to comparable ranges of external tasks. Imitation is possible because distinct physical systems can be organized to exhibit nearly identical behavior. [6]—p. 13.

Knowledge and skill are required to derive satisfactory artifacts. According to Simon, a design science curriculum should include the following topics[6]—p 134.

1. The evaluation of designs
2. The formal logic of design: imperative and declarative logics
3. The search for alternatives
4. The theory of structure and design organization: hierarchic systems
5. Representation of design problems

As may be evident from the preceding five steps, computers played a central role in Simon's work on design. Simon saw the computer as a means to evaluate elements of his growing theory.

SOAR, a problem-solving software system, was a collaborative design-science effort (with Allen Newell) to simulation human cognition[7]. Through the design of SOAR, Newell and Simon addressed the preceding five SoD issues. Consequently, there has been an intertwining between the science of design and the theory of human cognition. For example, SOAR realizes the *problem space hypothesis*, which asserts that all deliberate cognitive activity occurs as search in combinatoric spaces for goal attainment. This cognitive search theory and Simon's theory of satisficing in the face of bounded rationality are mutually supportive. Through computer simulation of their designed artifact, Newell and Simon realized and elaborated their theory of human cognition.

For Simon, a computer implementation is important for the simulation of artifacts, be they psychological, sociological, or simply computational.

How can a simulation ever tell us anything that we do not already know? …

1. A simulation is no better than the assumptions built into it.
2. A computer can do only what it is programmed to do.
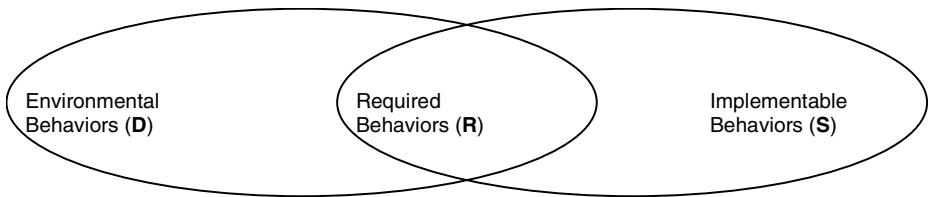
…The obvious point is that, even when we have correct premises, it may be very difficult to discover what they imply… Thus we might expect simulation to be a powerful technique for deriving [knowledge]. [6]—p 14-15

For Simon, the science of design may be entirely theoretical; however, to understand its implications requires simulation. For many interesting designs, this means a computer simulation.

1.2 System Requirements

Requirements engineering is the discipline that studies the relationship between Simon's "inner" and "outer" environments. Fig. 1 illustrates required behaviors as the intersection between environmental behaviors and implementable behaviors[8]. The artifact interacts with a portion of the world, which exhibits the environmental behaviors, as represented in domain properties. Implementable behaviors are executed by the artifact. A specification describes how the artifact produces its behaviors. A requirement refers to properties of both the environment and the artifact. A domain property only refers to properties of the environment. An artifact specification only refers to properties of the artifact.

The requirements problem is finding a specification **S** that for given domain assumptions **D** satisfies the given requirements **R**, which is written as **D**, **S** ♣**R**[9]. This simplified ontology does not address partial requirements satisfaction or requirements preferences. To do so, requirements are divided into functional goals **G**, soft goals **Q**, and attitudes **A**, which specify preferences over **G** and **Q**[10]. Using the extended

Fig. 1. Requirements as the boundary between environment behaviors and implementable behaviors

ontology, the requirements problem is finding a specification **S** that for given domain assumptions **D** satisfies the given functional goals **G**, soft goals **Q**, and attitudes **A**, which is written as **D**, **S** ♣– **G**, **Q**, **A**.

The relation ♣– does not allow new information—once a fact is asserted it always remains true. The bounded rationality present in Simon's problem solving view of design means that the many conclusions are tentative, to be retracted in favor of new evidence. Nonmonotonic reasoning allows us to draw conclusions based on the evidence at hand, and retract those conclusions as new evidence arises. Thus, the defeasible consequence relation ♣~ is use to assert nonmonotonic satisfaction[11].

According to Simon, design is a problem-solving, goal-seeking process with inherent uncertainties driven by unknowns in the changing environment and implementation. Design includes evolutionary adaptations, which increases artifact complexity. *Nearly decomposable* system design addresses complexity by ensuring that interactions among components are weak, although not negligible[6]. Thus, designing a nearly decomposable, evolving artifact in the face of changing and uncertain requirements is a difficult satisficing problem.

Requirements engineering commonly considers three distinct kinds of requirements properties.

1. A functional **goal** $g \in$ **G** is a desired property of the system and its environment; it describes what "ought" to occur. A realizable goal exhibits three characteristics relative to the system described: (i) it is described entirely in terms of values monitored by the system; (ii) it constrains only values that are controlled by the system; and (iii) the controlled values are not defined in terms of future monitored values. Because *the system* defined can be a composite of computer and human capabilities, a goal may describe desired human behaviors. An example user goal is, *eventually a user shall send an email message*.

2. A **softgoal** $q \in$ **Q** describes qualities or constrains quality values, whereby the described qualities have a subjective or ill-defined structure. A soft goal can only be satisfied, not only because of subjectivity, but also because the ideal level of satisfaction is beyond the resources available. For example, *an email message shall exhibit the best qualities of prose*.

3. An **attitude** $a \in$ **A** is a relative evaluation in terms of degree of favor or disfavor, which varies in sign (positive or negative) and in intensity. For example, *the Outlook email client is too complex and cumbersome*.

Requirements specification is the process that seeks to define a specification that maximizes compulsory and optional requirements according to attitudes[10].

One cannot distinguish between a requirement and an implementation specification without knowing what the system is intended to do. Email management is a requirement of an email client. Consequently, prescriptions of email manipulation (e.g., creation, deletion) are requirements. However, email transportation over computer networks are relative implementation details because they do not refer directly to properties of the email management domain. Thus, email client prescriptions are requirements in this context. When requirements are refined into their supporting service requirements, the implementation specification phase begins. By resetting the context, the requirements problem can begin anew with the requirements specification of the computer network system.

The emphasis on requirements specification distinguishes the requirements problem from Simon's general characterization of the design problem. Requirements specification, however, is not separable from implementations[12]. The *real-world* defines the feasible implementations, and guides the satisficing, problem-solving, search process. Requirements engineers, and all designers, listen to the feedback provided by the system's context.

1.3   Designs Can Talk

Schön may have been too hard on Simon. Schön's reflective practice intertwines the neat academic theories with the scruffy practice of real-world problem solving. In the dilemma of rigor or relevance, Schön emphasizes the relevance of problem setting, "the process by which we define the decision to be made, the ends to be achieved, that means that may be chosen. In real-world practice, problems do not present themselves to the practitioner as givens. They must be constructed from the materials of problematic situations that are puzzling, troubling and uncertain."[13] In Schön's view, the problem, practice (or artifact), and practitioner co-evolve with a new knowledge that is generated through the act of practice. In Simon's design context, the goals and uses of the artifact co-evolve with its usage. A good designer will listen to the issues that arise in the artifact's context, and redesign the artifact to address theoretically messy issues, which are often set aside for later consideration. Schön believes that Simon ignores the unanticipated or messy issues that arise in practice by "proposing a science of design that depends on having well-formed instrumental problems to begin with." Simon could have argued that simulation plays the environment's role prior to artifact deployment. Whether it be simulated or real, the artifact's environment includes unknowns because of bounded of rationality, and thus when new information is available the design goals and design evolve. Schön's reflective practices makes explicit the feedback necessary to guide the adaptation described in Simon's science of design.

Simon and Schön are consistent in that they recognize the significant role that the environment has on a systems evolution. In software, Lehman formulated eight "laws of software evolution", from empirical studies[14]. Two laws are particularly relevant to the role of environment-driven evolution. The Declining Quality law asserts that the system quality will diminish unless specific actions are taken to ensure quality evolution. The Feedback System law asserts that evolution includes a feedback

system, which describes the tension between change management and the desire to continually satisfy new stakeholder requirements. These laws are consistent with systems feedback theories from System Theory and Control Theory. It seems that it is widely accepted that monitoring and system adaptation are necessary if a system is to meet the needs of its ever-changing environment.

The theoretical paradigm of listening to the design in context is reflected in today's best-practice software companies, which are customer-focused businesses with agile-engineering practices placing stakeholders within the product evolution process, where continuous feedback guides product improvements. For Google Inc. and others, this entails focus groups, beta software, and other feedback mechanisms. Monitors, within the delivered software, send back customer preferences and behavioral information. Such feedback is used by developers to build user models, which guides software development. Stakeholders co-design and co-evolve products with their providers.

1.4 Software Is Personal

Continually adapted software placed in a unique environment results in personalized software. Consider the case where a software system is deployed to several unique sites. Each site places unique demands on the software. A responsive software system will listen and adapt to its environment. Because of the unique adaptations generated at each site, each software system will diverge from the original system to form a unique and personalized system.

Software personalization through continuous adaptation is possible. For some, software must be personalized. More than one million adults in the U.S. are diagnosed each year with cognitive impairments (CI) due to neurological disease or trauma (e.g., traumatic brain injury, stroke, tumor, epilepsy, infectious disease). Cognitively impaired patients often experience social isolation and its ancillary effects. Naturally, one might expect that some form of computer assistive technology (AT) would aid CI patients—for example, in communicating via email. Sadly, AT has not been shown to assist CI patients. One research project, however, suggests that personalized software may help[3, 5].

The Think and Link (TAL) project provides a specialized emailing system to CI patients[1, 3, 5, 15, 16]. In contrast to other AT studies, TAL patients continue to expand their email system usage. This unusual success is attributed, in part, to the personalized software. By continually monitoring clinical and user goals, TAL can responsively provide personalized software adaptations. With usage, the software used by each TAL CI patient is unique in its interface and functionality.

The approximately 53 million disabled Americans are only one group that may benefit from software personalization. Many of us have used Microsoft Word, Adobe Photoshop, or similarly complex software, only to wish for a reduced, simplified version that did just what we wanted, and no more. Microsoft introduced adaptable menus with this aim in mind. Instead of helping, the constant menu variation became another user burden. Consequently, most users disabled the adaptive menus. This technology is absent in the most recent version of Microsoft Office (2007).

Personalized software is still desired by users and organizations, as evidenced by the web pages, books, and consultants that provide customization guidance. A recent

article questions the relevance of IT for strategic advantage[17], in part because much software is a commodity. It may be that "IT doesn't matter", in part, because much of IT is not unique. Without some unique aspect, IT is not an intellectual property or strategic differentiator, and therefore, it is simply an infrastructure cost, like the manufacturing plant or electricity. In contrast to commodity software, personalized software is unique, and therefore it can be intellectual property and a strategic differentiator.

1.5   Requirements Monitoring for Design Feedback

Evolution through monitoring and reactive adaptations is a common, central theme in design, be it the positivist SoD or constructivist reflective practice[18]. Both monitoring and adaptation are important research topics. Herein, we focus on monitoring. In particular, monitoring the system and its impact on the evaluation of requirements, which in turn will inform the adaptation process.

A *requirements monitor* is a software system that analyzes the runtime behavior of another (target) system and determines the satisfaction of the target system's requirements from observing a stream of inputs (**I**). A requirements monitor can be characterized as a function that processes its input data stream to derive the status of requirements satisfaction.

$reqMon(i) \rightarrow Sat(r)$

The observed input stream in comprised of events from the software's environment, or domain **D**, and the internal events of the software itself, **S**, where $\mathbf{I} = \mathbf{D} \cup \mathbf{S}$.

In the remainder of this paper, we present a design science study of requirements monitoring for assistive technology. The study demonstrates that requirements level feedback on the runtime behavior of a system is possible. Such feedback provides designers information that enables responsive adaptation. In the larger context of design science, it demonstrates how design feedback may be obtained directly from natural, situated contexts. For software intensive systems, this suggests that designers may rely less on the feedback obtain from simulated systems and more on the objective feedback obtained directly from real-world systems.

2   Feedback for Assistive Technology

The *Think and Link* (TAL) project produced prototypes of (1) an assessment process called Comprehensive Overview of Requisite Email Skills (CORE), and (2) the TAL email interface. Recently, TAL (think-and-link.org) has been commercialized by Life Technologies LLC as CogLink (coglink.com). In support of TAL, the group defined and applied their *clinical requirements engineering* (CRE) methodology, a personalized software development process applied in support of clinical treatment that includes a high-level, individual process of assessment, goal-setting, treatment plan and periodic monitoring[16]. As a consequence of the TAL project, researchers posed three propositions of their CRE methodology[1].

1. For some domain, such as clinical rehabilitation, uniquely personal customization may be the only way to deliver usable systems.

    1.1. No single customization will be superior because of individual differences, both in skill sets and living environments. Therefore, uniquely personal customization is necessary.
2.   Modeling, monitoring, and adapting are necessary for satisfying the changing user software needs, preferences, and skills.
3.   Uniquely personal customization requires automation to scale to large populations.
    3.1. The activities of user monitoring and product adaptation are eligible for automation.

TAL researchers found that, although technical adaptations of the email client were relatively easy to accomplish, adaptation *decisions* required careful consideration of collected data: a wrong adaptation choice often led towards system abandonment. Statistical and qualitative data, along with direct input from the CI user, were reviewed at weekly staff meetings to determine whether the user was ready to take on new goals. These manual processes do not scale beyond small studies. Consequently, we began the research on REQuirements MONitoring (REQMON) of CI user models in the latter phase of the TAL project with a focus on automating the monitoring, re-evaluation, and adaptation steps in the overall clinical process.

The following sections describe our application of the Peffers *et al.* design science process steps to the CRE automation proposition, from 'problem identification and motivation' through to 'evaluation'"[19].
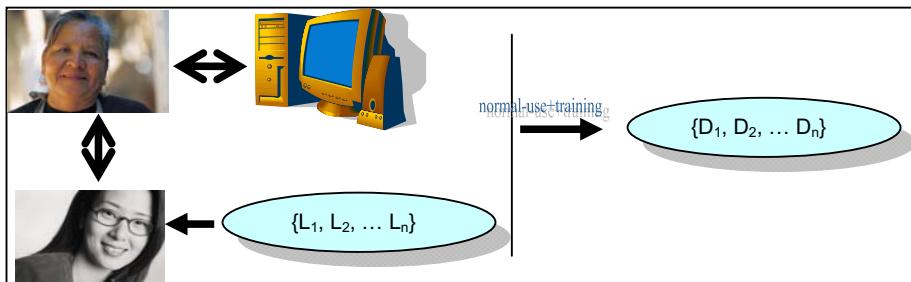
2.1   Problem Identification and Motivation

More than one million adults in the U.S. are diagnosed each year with cognitive impairments (CI) due to neurological disease or trauma (e.g., traumatic brain injury, stroke, tumor, epilepsy, infectious disease). Currently, there are between 13.3 to 16.1 million Americans living with chronic brain disorders and associated CI[20]. In the coming years, incidence rates are expected to rise due to the development of dementias associated with a rapidly aging population and increased survival rates associated with improved medical management of neurological impairments[21].

Assistive Technology (AT) should offer great promise for supporting people with CI in their own homes with their own computers. However, research on effective *technology design* that facilitates long-term adoption by individuals with acquired and developmental CI is sorely lacking. To address this need and support of CI individuals, the TAL project applied CRE theory to the continuous adaptation of an email client for CI individuals.

2.1.1   A Problem Scenario

Assume that Jill is interested in learning to use email. Jill acquired a brain injury in an auto accident, and has impairments in both memory and executive functions rendering it difficult to learn new skills. Jill has no memory of using a computer in the past, although her closet contains several computers, which were given by friends and family. Jill is unable to use her computers. She decides to work with a TAL staff member, Andrew, to explore the use of email. Andrew uses the CORE process to obtain two important items[22]: (1) Jill's personal goals for using email, and (2) Jill's existing
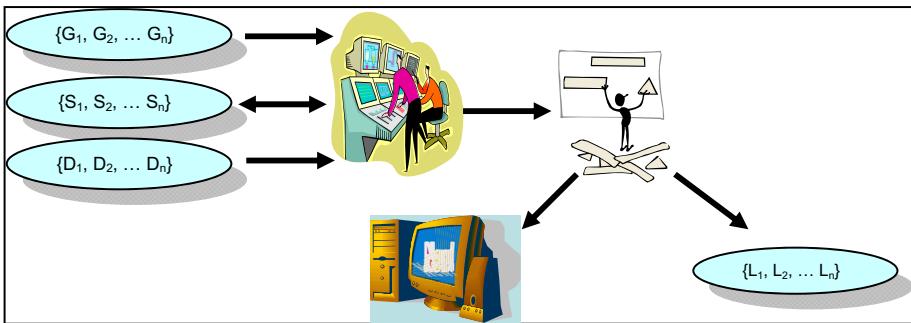
Fig. 2. Normal usage and training generates data

skills for using email independently. (Other information is obtained but omitted here for brevity.) Using this information, Andrew produces a user profile containing (1) a specification of an initial system to deliver to Jill, and (2) a training plan broken into a set of lessons. Notice that Jill's personal goals play an important role here (and will continue to play a role in the future): they filter from all possible skills the subset that is necessary to meet Jill's needs. In essence, this is goal-directed training. In a similar fashion, the system that is delivered is one that fits with both Jill's current skills and her personal goals—some of which are deferred. It's important to note that Jill has high aspirations for her use of email. She would eventually like to contribute articles to the online newsletter, published by a local group that advocates for the disabled. This goal, however, is not realizable given her current skills. Thus, it becomes a deferred goal that will be monitored. More on this shortly.

Next, the email system is delivered to Jill. A family careprovider, Jill's daughter-in-law Ann, assists in the training task. Soon, Jill is busy using the email system to reconnect with family and friends that have dropped out of touch. Both Jill's daily usage, and her training sessions with Ann, produce raw data. This data includes that which is generated from the email system itself, along with Ann's input on training progress. Data is also collected periodically from Jill and her email buddies through online questionnaires. Fig. 2 illustrates the data collection process.

As data is generated, it is used to make decisions about deferred goals. Working backwards, Jill has goals that are not satisfied currently. Each of these goals has preconditions, in terms of skills, that enable them. Each skill, in turn, has measured activities that signal the learning and retention of the skill. These measures can be evaluated from the collected data. Fig. 3 illustrates the process. Raw data is monitored. This data is evaluated for evidence of existing, and eventually, new skills, shown as an arrow back to Jill's skill set. Eventually, a match is made between a goal deferred and the skills necessary to achieve it. At this point, email system adaptation becomes the means to enabling the goal. With Jill's deferred goal of contributing to the online advocacy newsletter, the following skills are among those require. Evidence for these skills can be measured. Suppose that all but the last skill have been demonstrated. At this point, two options are possible: (1) continue to train Jill in spelling and grammar, or (2) provide automated support through her email system. A spelling checker and a grammar checker are two of the functions that can be added to Jill's system (shown as the pile of pieces available to the person at the board in

Fig. 3. Data monitoring supports the adaptation process

Fig. 3). A decision is made to adapt Jill's system to include a spell checker, deferring the decision on the grammar checker until a later point. Two explicit products come from the adaptation: (1) a newly designed system that includes spell checking, and (2) additional training lessons to accommodate the new design. The implicit outcome, not shown in Fig. 3, is forward progress toward the needed skill, and hence, progress toward a deferred goal.

It is important to note here that we take a holistic system view of the problem: adaptations can occur at the software-architecture level, but also at the social-human level. The system "configuration" is much more than simply the software pieces.

In closing, if we were to continue, we would form a cycle: deferred goals are achieved, new goals may arise, and other goals may be dropped. Although our examples are optimistically forward driven, it may be that certain skills are lost over time, making system *retraction* a real possibility, thus adapting the system to a less complex version rather than a more complex version.

2.1.2 Problem Summary

We want to highlight three issues from Jill's story.

1. Monitoring is critical. CI users can and will abandon poorly designed AT systems at any point along a timeline. It behooves us to monitor their progress, noticing obstacles and achievements. Such *micro-usability monitoring* tracks individual user behaviors on the scale of minutes to months. This is particularly challenging, as we must hypothesize monitored properties that serve as proxy for our understanding of Jill's skills.
2. Analysis must consider the *composite* design[23, 24]. The software component of a system plays an important role. But the human context is equally important. Looking at Jill's example, her careprovider and her buddies are components in the design space. These components can be "configured" (e.g., add/remove buddies, add/remove careproviders). We can also attempt to influence their behavior in the system (e.g., through prompting).
3. System adaptation is based on selection from a given set of alternatives. The size of this set is roughly five thousand different alternative designs for the TAL email system. The TAL experience tells us that, for AT delivered to the CI population, there is little need for creative-design to generate unforeseen systems on

the fly. Consequently, the processes of generating and installing *new* design components are not central to TAL. Instead, careful attention must be paid to correctly choosing among a moderate set of pre-determined designs.

These three research issues are central to the TAL project and to CRE projects in general. Of the three, TAL researchers felt that automated monitoring was the linch-pin to their success—they could address analysis and adaptation for their growing CI population, but only if monitoring were automated. Thus, we formulated our design research question:

1. [DSRQ] Can low-level monitoring of software events be used to de-termine user goal successes and failures?

2.2   Solution Objectives

Device abandonment in the AT field is well documented[25-27]. CRE represents a successful strategy to combat abandonment [22]. However, the problematic processes are user monitoring and product adaptation, which are dependent on changing user skills. Moreover, these processes are labor intensive. Unfortunately, current com-puter-aided techniques for extracting usability information are insufficient for the CI models[28]. Automation of user monitoring and product adaptation needed be ad-dressed before TAL could scale to larger populations.

To ground the problem in the TAL context, consider Fig. 4, which is a snapshot of a TAL email interface. This configuration is minimal in terms of functions and prompts, as required by CI users. As can be seen, there are eight buddies. The inter-face must adapt with the user. For example, when the user becomes successful—even bored—with emailing, then a new buddy may be added to the list. To avoid device abandonment, TAL needs an automated method for continuously adapting the TAL email system to individual CI users—each CI user must have their own continuously personalized email interface. To inform the adaptation process, an automated method for monitoring CI user goals is a prerequisite. This leads us to our design research objective:

1. [DSO] Define and demonstrate a computer supported method for monitoring CI user models within the context of the TAL project.

Although CRE requires both monitoring and adaptation, the TAL project required immediate results on DSO 1. The TAL researchers felt that they could manually adapt some systems if automated goal monitoring existed, but they could make no scaling-up progresses without it. Thus, we initiated a design science project to address DSO 1 in the latter phase of the TAL project. Next, we describe the design and development of an automated goal monitoring system.

2.3   Design and Development

We applied a design research method to answer DSO 1. A review of extant monitor-ing systems revealed that none would be sufficient. In particular, the specification languages were too limited for our user goal models. To see why, we describe the TAL user models, and then our goal monitoring system.
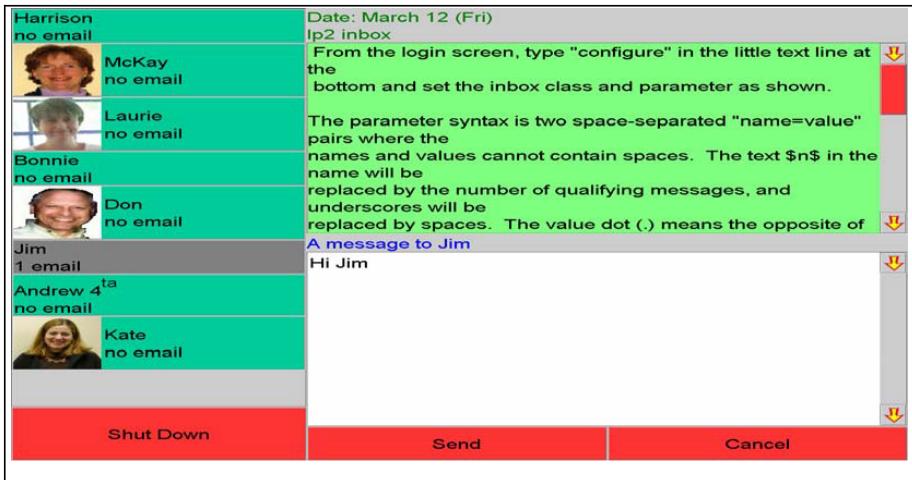
Fig. 4. One configuration of the TAL email interface

2.3.1 User Modeling

User modeling serves two purposes. First, it can describe user requirements (i.e. goals, softgoals, and attitudes) and their associated skills, which are used to define training and system adaptation. Second, it can describe hypothesized behaviors, which are important to adaptation. Both models are represented in a common notation.
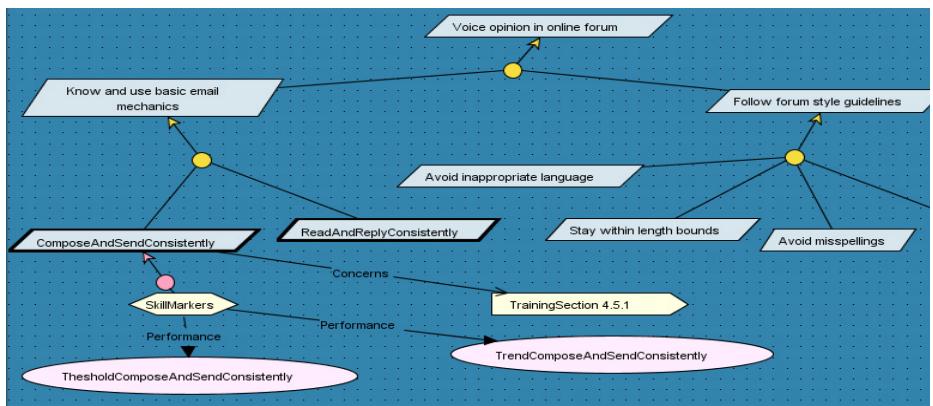
Fig. 5. A portion of Jill's goal graph in the KAOS tool, Objectiver

Fig. 5 illustrates a portion of Jill's KAOS goal graph, i.e., the portion centered on her desire to have her opinions known. The TAL project defined the generalized pattern of wishing to participate in an online forum (e.g., newsletter, discussion group). We use this pattern here to capture Jill's goal. To personalize the pattern, we must fill-in details on the particular newsletter Jill wishes to target. Before doing that, we will

look at the left side of the figure, focusing on the skills associated with basic email use.

The goal of knowing how to use basic email functions is decomposed into lower level skills, which are denoted by a thicker outline in the diagram of Fig. 5. Users such as Jill can fail to acquire the emailing skill; for instance, because they forget to send a composition after becoming distracted with another intermediate activity. But what if Jill is able to send a message at least once. Is that enough to mark off the skill as acquired? Typically not. What we would like to see in skill acquisition are: (1) a good success-to-failure ratio over sessions, and (2) a trend that shows this ratio improving or at least holding constant, i.e., not declining. We have applied the KAOS tools to define measures of this type[29], as the following definition shows:

```
Goal Maximize [ComposeAndSendConsistently]
  UnderResponsibility CI-User (Jill)
  FormalDef
     ThesholdComposeAndSendConsistently ∧
     TrendComposeAndSendConsistently
```

The goal is satisfied when both skill measures are satisfied; the measures and related definitions follow.
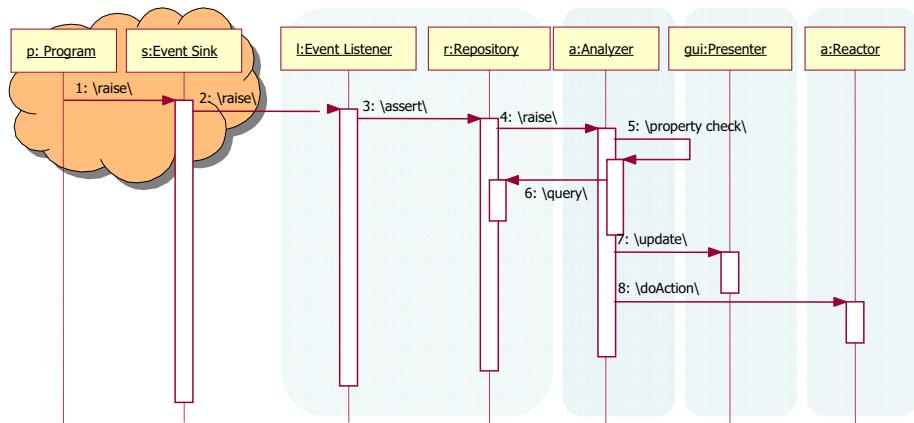
```
Measure[ThesholdComposeAndSendConsistently]
  FormalDef
    Average(PercentSucessful(SendComposedEmail(*),24h),-14d))
    ≥ 75%
Measure[TrendComposeAndSendConsistently]
  FormalDef
    Slope(PercentSucessful(SendComposedEmail(*),24h),-14d)) ≥
    0
Definition [PercentSucessful]
  FormalDef
    Satisfied(g,p) / Failed(g,p)
Goal Ideal [SendComposedEmail]
  UnderResponsibility CI-User (Ideal)
  FormalDef
   ∀ m:EmailMessage
       EmailCompose(m) ⇒ ◊_{<t} EmailSend(m)
```

The two measures track Jill's overall success. Each relies on the perfect condition of sending a composed email (within time *t*) captured in the ideal goal `SendComposedEmail`. Given the ideal, the measures compute the average and trend of success for the last 14 days, where each 24-hour period is a data point. The measures are automatically updated in real-time by our goal monitor system. Thus, as a user works with their AT system, the evaluations of measures, skills, and goals are updated in real-time, providing views of a user's goal satisfaction to caregivers and the adaptation system.

2.3.2 Goal Monitoring

The monitoring system, REQMON.CI was developed in response to DSO 1. It is built upon the REQMON toolkit[1, 30-34], which supports KAOS goal monitoring[29].

Fig 6. illustrates the REQMON component architecture. In the figure, each box is a software component, which may be network distributed; alternatively, the whole system can be deployed as one embedded program. Fig 6 illustrates component interactions that occur when a monitored event is observed. The shaded portions toward the right indicate typical process boundaries; thus, the monitored program and event sink typically comprise one process, the event listener and repository comprise another process, finally the analyzer, presenter and reactor each have their own processes. REQMON defines the components from the event sink through the reactor.

Fig. 6. Illustration of ReqMon component interactions

The roles of the REQMON components are as follows:

- The PROGRAM represents any general-purpose executing program.
- The EVENT SINK represents the observed state of the executing program. Events are generated at monitoring points, which are inserted into the target program, or its context.
- The EVENT LISTENER represents the observer. An EVENT LISTENER can listen to multiple EVENT SINKs. REQMON defines listeners for common instrumentation and computer management frameworks, including log4J/log4net and Common Base Event (CBE), a common open, heterogeneous, enterprise-monitoring framework..
- The REPOSITORY maintains the historical database of events and properties. A rule-based system caches the working database, while the complete records are persisted to a relational database system.
- The ANALYZER executes the monitoring algorithm to evaluate properties.
- A PRESENTER maintains a user display of events and property evaluations.
- The REACTOR is the event handler. It responds to REPOSITORY changes, such as property violations. Actions can include reconfiguration or substitution of the targeted software components.

REQMON is designed with a component architecture, as Fig. 6 illustrates. Each component is defined as a 'plug-in'; developers can extend or override components with their own implementations. This software architecture supports the exploratory nature of design science research. For example, REQMON.CI is a refinement of REQMON that defines a specialized ANALYZER 'plug-in' for evaluating KAOS goals in the fulfillment of DSO 1. SERMON, another extension of REQMON, supports the specification and monitoring of SOA protocol policies[30].

2.4 Demonstration

To monitor the user goals, a TAL software analyst engages in the following activates:

1. Specify the user goals informally.
2. Formalize those goals in KAOS.
3. Translate the KAOS goals into the REQMON property specification language.
4. Add the property specification to REQMON, which begins the monitoring process.

As a natural part of software development, we evaluated the REQMON design through functional (black-box) testing, performance analysis, and scenario test cases. What follows, is a portion of one of the scenario test cases.

2.4.1 Goal Modeling

Consider typical CI-user emailing goals. They include subgoals for the basic steps of emailing. For example, consider the following goals, which are included in our common goal library.

$G_{presence}$: The period between viewings of the email in-box shall be no more than k days.

G_{send}: After composing an email message, the user shall send the email, within k hours.

G_{read}: After noticing a new email, a user shall read the email, within k hours.

G_{reply}: After receiving an email, a user shall read and reply to the sender, within k days.

$G_{read-delete}$: Before deleting an email, a user shall read the email.

Clinicians want to see: (1) a good success-to-failure ratio over sessions, and (2) a constant or improving trend of this ratio. This leads us to refine these goals further. For example, the preceding G_{reply} goal has the following two refined goals:

$G_{reply-ratio}$: The ratio of successes vs. attempts for email replies shall be $\geq 75\%$, with any two-week period.

$G_{reply-ratio-trend}$: The trend of goal $G_{reply-ratio}$ shall be positive (increasing), with any two-week period.

A TAL software analyst formalizes such informal goals in KAOS. As an illustration, the preceding G_{reply} goal is specified in KAOS as follows:

```
Goal Achieve [Reply]
 UnderResponsibility u1
 FormalDef
  ∀ m1,m2:EmailMessage, u1,u2:User
  EmailArrive(m1,u2,u1) ⇒ ◊<t EmailSend(m2,u1,u2)
```

The KAOS goal `Reply` represents our informal G_{reply} goal.

2.4.2 REQMON Property Specification

The KAOS goals represent requirements without regard to monitorable events. The monitoring system, however, receives events from the TAL email system as it executes. The goals and monitorable events have two distinct ontologies. For example, the KAOS predicate `EmailArrive` gives no indication of the software component that generates the email event—that is, there is no design information is the non-operational KAOS goals. To bridge this gap, the software analyst rewrites the KAOS goal in terms of events monitored by REQMON.

Here, a design research decision had to be made regarding the input language of REQMON. An early prototype relied on the language underlying the REQMON implementation[34]. The KAOS language itself was also considered. However, it is mainly used for requirements definition—design decisions are eschewed. Conversely, the unified modeling language (UML) and its object constraint language (OCL) are mainly used for design. Moreover, the UML has a much greater market than does KAOS, and therefore has the potential for new streams of design science research with REQMON. Thus, we chose the UML OCL, extended to include temporal expressions, for a REQMON specification language.

The REQMON property specification for the property, `Reply`, follows.

```
-- Every email read must have a following reply
-- to the email buddy, within timeout.
context Session
inv:
    Events->
      forAll(r,s : EmailEvent | r <> s
                   and r.methoName='read' and s.methoName='send'
           implies (response@6h(r,s and r.buddy = s.buddy)))
```

The property is expressed using a standard OCL invariant, except for the use of the `response` temporal pattern. The pattern `response@t(A,B)` specifies that event `B` responds to event `A` within time `t`[35]. The property specifies that a `Session`'s `Events` must satisfy the specified conditions, namely an arriving email message must have a matching reply. (TAL's email uses a `buddy` field to indicate the non-CI user.)

We have adopted the Dwyer *et al.* temporal patterns, which define five scopes over eight temporal patterns. They were generalized from an empirical study of 555 specifications [35]. Their property patterns include `universal`, `absence`, `existence`, `bounded existence`, `response`, `precedence`, `chained precedence`, and `chained response`. Their scope patterns include `global`, `before R`, `after Q`,

between Q and R, and after Q until R. Distinguishing scoping properties from other properties seems to simplify property specifications through modularization.

The Dwyer temporal patterns defined the test cases for our functional (black-box) testing and performance analysis. Thus, we had 40 functional test cases (5×8). With REQMON passing the functional tests, we were convinced that it worked properly and that it could evaluate most temporal properties that it would be likely to encounter, according to[35].

2.4.3 Monitoring Point Implementation

REQMON relies on instrumentation and computer management frameworks to generate and transport events. For example, important events can be published from the TAL software using instrumentation statements such as the following:

```
Log.Info(GetXml(EmailEvent));
```

The call to GetXml converts an event into a XML representation. The call to the logging program,

log4J's Info method, generates the log stream on which REQMON listens. Thus, events can travel through a distributed logging program to REQMON.

2.4.4 Presenting Property Changes

REQMON can initiate actions in response to changes. In the REQMON toolkit, this amounts to executing rules in response to changing property evaluations. When a property is violated, for example, a rule can notify an external system, which can present a graphic visualization or a textual trace. As an illustration, consider monitoring the preceding EmailReply property. It is satisfied by the event sequence presented in Table 1. The SessionEvent(open) and SessionEvent(close) events *open* and *close* the between scope, respectively. While its scope is open, the monitor checks its property specification against arriving events. Together, the EmailEvent(read) and EmailEvent(reply) event sequence satisfy the EmailReply property.

Table 1. Incremental evaluation of email events by the EmailReply property monitor

Event	*Resulting Animated slide evaluation*[†]
E$_1$: SessionEvent(open)	PropertyEvaluation(pe$_1$,SessionOpen,(E$_1$),satisfied) ScopeEvaluation(se$_1$,InSession,open,pe$_1$)
E$_2$: EmailEvent(read)	PropertyEvaluation(pe$_2$,EmailRead,(E$_2$),partial)
E$_3$: EmailEvent(send)	PropertyEvaluation (pe$_3$,EmailSend,(E$_3$),satisfied) **PropertyEvaluation(EmailReply,se$_1$,(pe$_2$,pe$_3$),satisfied)**
E$_4$: SessionEvent(close)	PropertyEvaluation(pe$_4$,SessionClose,(E$_4$),satisfied) ScopeEvaluation(se$_1$,InSession,open,pe$_1$,close,pe$_4$)

[†]PropertyEvaluation*(<property-evaluation>, <property>, (<events>), <satisfaction>)*
 ScoperEvaluation*(<property-evaluation>, <property>, (<events>), <satisfaction>)*

2.4.5 Performance Analysis

A performance analysis experiment was conducted by comparing the total runtime of no monitoring for the 40 combinations of the Dwyer temporal patterns. Data sets were provided to each of the 40 configurations. Each data set was generated in real-time by iteratively executing the following code until the iteration limit was reached— iterations of 1,000; 2,000; 4,000; 6,000; 8,000; and 10,000 were executed.

```
ProgramEvent pe = new ProgramEvent();
pe.setMethodName("event1");
pe.setClassName("class1");
reqmon.assertObject(pe);
ProgramEvent pe2 = new ProgramEvent();
pe2.setMethodName("event2");
pe2.setClassName("class1");
reqmon.assertObject(pe2);
```

Two events are required to satisfy the sequential properties (precedence and response), whereas other properties only requires one event. Nevertheless, two events were generated in each iteration so that all configurations are comparable.

In the experiments, the event generator and REQMON run in the same multi-threaded process. (REQMON can process multiple, asynchronous event streams.) In the experiments, each test runs as a JUnit test case within Eclipse on a Windows Server 2003 dual core 2.8 GHz with 1G memory. The results suggest that, within the test configuration, sequential properties (of length 2) can process 137 event pairs per second, and other properties can process 417 events per second. This indicates that algorithm is reasonably efficient for many monitoring problems.
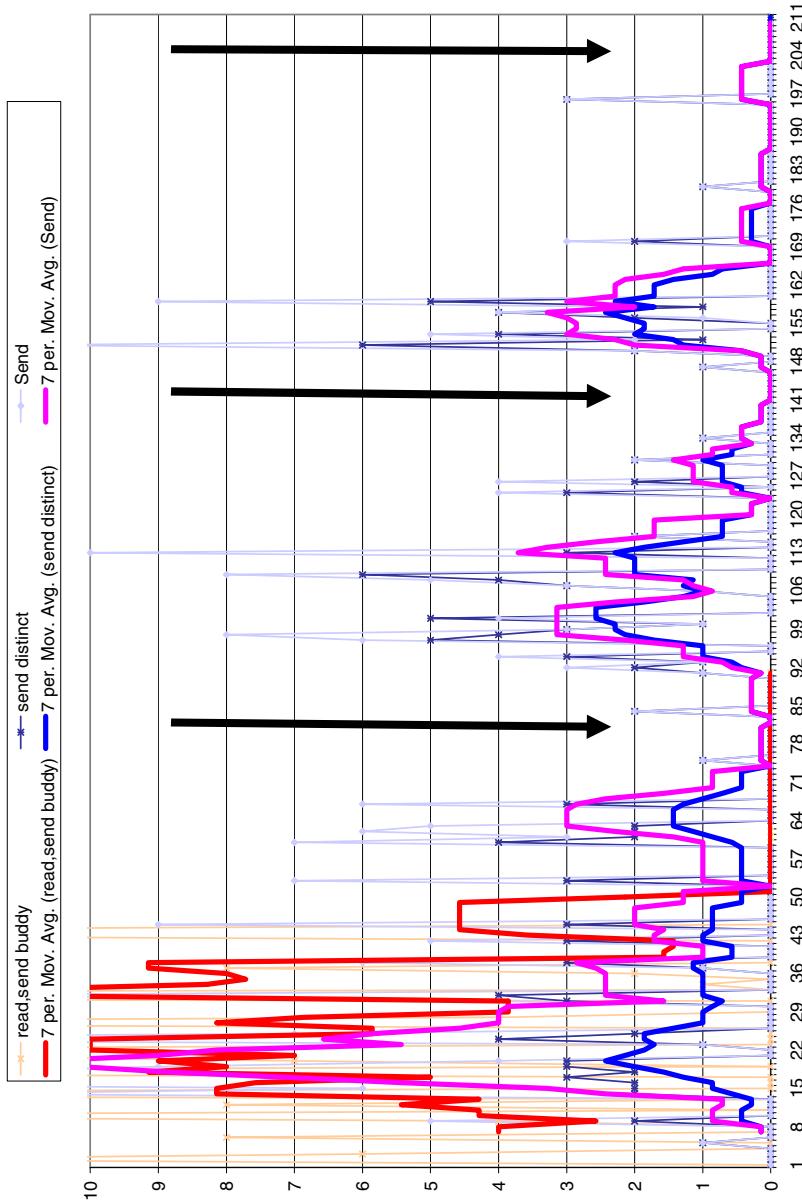
2.5 Evaluation

The TAL project provides an exploratory case study in the use of REQMON.CI for user model monitoring[1, 36, 37]. Before initiating formal case studies, we sought to gain insights from this real-world application concerning DSRQ1.

In collaboration with clinician, developers specified the user models. Then, the CI user computer usage was monitored by REQMON.CI. Analysts presented the results to the clinicians as graphs, generated by Excel from the REQMON.CI data.

Fig. 7 illustrates monitoring higher-level goals, such as replying to email (G_{reply} of §0). Although goal G_{reply} is satisfied during the first 50 days, it drops to zero thereafter. Email sending—including sending to distinct buddies—spikes and lags corresponding to the emailing pattern of the initiating buddies.

The exploratory case study confirms the DSRQ1: low-level monitoring of software [email] events can be used to determine user goal successes and failures. In particular, the low-level events captured by the TAL email software can be recognized, abstracted, and related to user goals, such as `EmailReply`, thereby, the user goals can be monitored. In the study, the eight users generated about 200 events per day. Two years data for eight CI users occupies 80,486 database records in 321.8 MB within SQL Server 2005, which include 138,107 property evaluations produced by REQMON.CI.

Fig. 7. The first 210 days for Don showing 7 day moving averages of replying (mainly at left), sending email, and sending to distinct buddies (below sending). Arrows point to periods of email client adaptation using new "email buddies."

Although the exploratory case study confirms the existential question posed by DSRQ1 within the CI emailing domain, the context of the TAL project led us to refine DSRQ1 into the following sub-questions:

    1.1. Is the user modeling language sufficiently expressive for the application?
    1.2. Can a team of users and developers specify properties properly?
    1.3. Is property analysis sufficiently efficient?

The case study does not provide definitive answers to these questions. However, it does provide the basis for some initial insights, which we summarize as follows:

- *Is the user modeling language sufficiently expressive for the application?* All monitoring requests (for available data) have been accommodated. The logic-based language includes temporal patterns and timeouts, which seemed to simply the property specifications. It was relatively simple to specify, for example, "During a session, when a CI user receives an email, the user should read and reply to that specific email." Both the email session context and the matching of the email sender with the reply recipient are context-sensitive expressions that can be directly expressed in the specification language. Our success, however, does not suggest that all the properties types for this domain were represented—we only represented those that were requested.

- *Can a team of users and developers specify properties properly?* The team of cognitive rehabilitation specialists specified properties, informally. The software developers formalized those properties and derived the monitors. In particular, a REQMON.CI developer (the first author) did most of the formalization and monitor definition.

- *Is property analysis sufficiently efficient?* A small Microsoft Windows server handed the load of eight CI users, easily. The test configuration, can continually evaluate about 36,028,800 events each day for the tested two-event sequence property. This translates to roughly 180,144 CI users per day—again, assuming only the simple two-event sequence property in the preceding Demonstration section. Nevertheless, this suggests that REQMON.CI scales to larger sets of CI users. We tentatively suggest that it may be efficient enough to be deployed on a typical Windows server computer and perform the monitoring of several non-trivial properties over continuous streams of a hundred events per second.

It must be emphasized that this has been an exploratory case study, which included informal, nondirected interviews. Consequently, these finding may not generalize to other monitoring projects. Nevertheless, given the data volume and property complexity, it does suggest that this approach may be utilized in a variety of application domains.

3 Discussion

Despite their varied views, both Simon and Schön would agree that continuous feedback guides design improvements. Modern software companies apply this practice through mainly manual mechanisms. Both Simon and Schön observed that designed artifacts contain implicit value. Through simulation, Simon sought to understand the implications of a specified theory. Through reflecting on practice, Schön sought to understand the messy, poorly understood implications of a theory's application.

Ideally, both views seek to obtain design feedback from natural problem settings. For computing systems, the requirements monitoring paradigm illustrated by REQMON provides a means to obtain such high-level design feedback.

Automated requirements monitoring provides effective technique for guiding design evolution. User-goal satisfaction can be tracked and aggregated over time and users to provide an objective measure of goal attainment. Weakly satisfied user-goals can be traced to poorly performing components, which can be reconfigured or reengineered to meet user expectations.

Assistive technology for the cognitively impaired provides an interesting environment to study design feedback. Like many users, the cognitively impaired evaluate the whole system, not just the software; they will abandon a system if it does not satisfy their goals. Consider the consequences for TAL if goal monitoring were absent. Our illustrative user, Don, would have stopped using the system, which is the norm for such assistive technology. Instead, with monitoring, the developer can react to the identified changes in user satisfaction. Consequently, the system continues as it evolves to meet the changing user needs. Automated requirements monitoring facilitates the unending process of seeking to meets user requirements.

Requirements monitoring is fundamental to design theory. It supports Simon's theory of goal attainment and adaptation.

> The artificial world is centered precisely on this interface between the inner and outer environments; it is concerned with attaining goals by adapting the former to latter. … The proper study of those who are concerned with the artificial is the way in which that adaptation of means to environments is brought about—and central to that is the process of design itself. [6]—chapter 5.

Requirements monitoring can reveal essential design properties that support or obstruct goal attainment.

Automated requirements monitoring may not be appropriate for all designs contexts. As Simon notes, "Everyone designs who devises courses of action aimed at changing existing situations into preferred ones" [6]—chapter 5. Thus, one can envision the design of a non-operational artifact, such as a business strategy. A designed artifact having an informal representation provides little objective feedback; the quality of an informal design is in the eye of the beholder. However, if the strategy (the designed artifact) is applied in a real or simulated business, then objective feedback can be obtained. Thus, design representation (informal, formal, or operation) determines largely the quality of the feedback, and thereby the evaluation of the design. Automated requirements monitoring applies to operational systems, which allows for many interesting designed artifacts. Simon, for example, placed great emphasis on computer models as a means to explore designs, even social and cognitive designs. Automated requirements monitoring is applicable to equally diversion domains.

References

1. Fickas, S., Robinson, W., Sohlberg, M.: The Role of Deferred Requirements: A Case Study. In: International Conference on Requirements Engineering (RE 2005). IEEE, Paris (2005)
2. Sohlberg, M.M., Ehlhardt, L.A., Fickas, S., Sutcliffe, A.: A pilot study exploring electronic mail in users with acquired cognitive-linguistic impairments. Brain Injury 17, 609–629 (2003)

3. Sohlberg, M.M., Fickas, S., Ehlhardt, L., Todis, B.: The longitudinal effects of accessible email for individuals with severe cognitive impairments. Aphasiology 19, 651–681 (2005)
4. Sutcliffe, A., Fickas, S., Sohlberg, M.M., Ehlhardt, L.A.: Investigating the usability of assistive user interfaces. Interacting with Computers 15, 577–602 (2003)
5. Todis, B., Sohlberg, M.M., Hood, D., Fickas, S.: Making electronic mail accessible: Perspectives of people with acquired cognitive impairments, caregivers and professionals. Brain Injury 19, 389–402 (2005)
6. Simon, H.: The Sciences of the Artificial. MIT Press, Cambridge (1996)
7. Newell, A., Simon, H.A.: Human problem solving. Prentice-Hall, Englewood cliffs (1972)
8. Jackson, M.J.: Software requirements & specifications: a lexicon of practice, principles, and prejudices. ACM Press, Addison-Wesley Pub. Co., New York (1995)
9. Zave, P., Jackson, M.: Four Dark Corners of Requirements Engineering. Transactions on Software Engineering and Methodology, ACM 6, 1–30 (1997)
10. Jureta, I.J., Mylopoulos, J., Faulkner, S.: Revisiting the Core Ontology and Problem in Requirements Engineering. In: 16th IEEE International Conference on Requirements Engineering. IEEE, Barcelona (2008)
11. Simari, G.R., Loui, R.P.: A Mathematical Treatment of Defeasible Reasoning and its Implementation. Artificial Intelligence 53, 125–157 (1992)
12. Swartout, W., Balzer, R.: On the inevitable intertwining of specification and implementation. CACM 25, 438–440 (1982)
13. Schon, D.: From Technical Rationality to reflection-in-action. Supporting Lifelong Learning (2002)
14. Lehman, M.M.: Software Evolution. Encyclopedia of Software Engineering 2, 1507–1513 (2002)
15. Sutcliffe, A., Fickas, S., Sohlberg, M.M.: Personal and Contextual Requirements Engineering. In: 13th IEEE International Conference on Requirements Engineering, 2005. Proceedings, pp. 19–30 (2005)
16. Fickas, S.: Clinical requirements engineering. In: Proceedings of the 27th international conference on Software engineering, pp. 28–34 (2005)
17. Nicholas, G.C.: IT doesn't matter. Harvard Business Review 81, 41 (2003)
18. Cross, N.: Designerly Ways of Knowing: Design Discipline Versus Design Science. Design Issues 17, 49–55 (2001)
19. Peffers, K., Tuunanen, T., Gengler, C.E., Rossi, M., Hui, W., Virtanen, V., Bragge, J.: The Design Science Research Process: A Model for Producing and Presenting Information Systems Research. In: First International Design Science Research in Information Systems and Technology, Claremont, CA, pp. 83–106. Claremont Graduate University (2006)
20. Alliance, F.C.: Incidence and Prevalence of the Major Causes of Brain Impairment (2001)
21. Sohlberg, M.M., Mateer, C.A.: Cognitive rehabilitation: An integrated neuropsychological approach. Guilford Publication, New York (2001)
22. Sohlberg, M.M., Ehlhardt, L., Fickas, S., Todis, B.: CORE: Comprehensive Overview of Requisite E-mail Skills. University of Oregon, Department of Computer and Information Science, Eugene, OR (2002)
23. Avison, D.E., Wood-Harper, A.T.: Multiview: An Exploration in Information Systems Development. Blackwell Scientific Publications, Malden (1990)
24. Fickas, S., Helm, R.: Knowledge representation and reasoning in the design of composite systems. IEEE Transactions on Software Engineering 18, 470–482 (1992)
25. LoPresti, E.F., Mihailidis, A., Kirsch, N.: Assistive technology for cognitive rehabilitation: State of the art. Neuropsychological Rehabilitation (in press)

26. Wright, P., Rogers, N., Hall, C., Wilson, B., Evans, J., Emslie, H., Bartram, C.: Comparison of pocket-computer memory aids for people with brain injury. Brain Injury 15, 787–800 (2001)
27. Wilson, B.A., Emslie, H.C., Quirk, K., Evans, J.J.: Reducing everyday memory and planning problems by means of a paging system: a randomised control crossover study. Journal of Neurology, Neurosurgery, and Psychiatry 70, 477–482 (2001)
28. Hilbert, D.M., Redmiles, D.F.: Extracting Usability Information from User Interface Events. ACM Computing Surveys 32, 384–421 (2000)
29. Dardenne, A., van Lamsweerde, A., Fickas, S.: Goal-Directed Requirements Acquisition. Science of Computing Programming 20, 3–50 (1993)
30. Robinson, W.N., Purao, S.: Specifying and Monitoring Interactions and Commitments in Open Business Processes. IEEE Software (2008) (accepted for publication)
31. Robinson, W.N.: Extended OCL for goal monitoring. Electronic Communications of the EASST 9, 1–12 (2008)
32. Robinson, W.N.: A requirements monitoring framework for enterprise systems. Requirements Engineering Journal 11, 17–41 (2006)
33. Robinson, W.N.: Implementing Rule-based Monitors within a Framework for Continuous Requirements Monitoring, best paper nominee. In: Hawaii International Conference On System Sciences (HICSS 2005). IEEE, Big Island (2005)
34. Robinson, W.N.: Monitoring Web Service Requirements. In: 11th IEEE International Conference on Requirements Engineering, pp. 65–74. IEEE Computer Society, Monterey Bay (2003)
35. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: Twenty-First International Conference on Software Engineering, Los Angeles, pp. 411–420 (1999)
36. Kitchenham, B., Pickard, L., Pfleeger, S.L.: Case Studies for Method and Tool Evaluation. IEEE Softw. 12, 52–62 (1995)
37. Yin, R.K.: Case Study Research: Design and Methods. Sage Publications, Thousand Oaks (2002)

Section 3: Quality and Value-Based Requirements

John Mylopoulos

Traditionally, research and practice in software engineering has focused its attention on specific software qualities, such as functionality and performance. According to this perspective, a system is deemed to be of good quality if it delivers all required functionality ("fitness-for-purpose") and its performance is above required thresholds. Increasingly, primarily in research but also in practice, other qualities are attracting attention. To facilitate evolution, maintainability and adaptability are gaining popularity. Usability, universal accessibility, innovativeness, and enjoyability are being studied as novel types of non-functional requirements that we do not know how to define, let alone accommodate, but which we realize are critical under some contingencies. The growing importance of the business context in the design of software-intensive systems has also thrust economic value, legal compliance, and potential social and ethical implications into the forefront of requirements topics. A focus on the broader user environment and experience, as well as the organizational and societal implications of system use, thus has become more central to the requirements discourse. This section includes three contributions to this broad and increasingly important topic.

Traceability mechanisms capture critical information on inter-dependencies that exist between software artefacts. For instance, traceability links may relate a design object to the requirements it depends upon, or to programming artefacts that implement it. Traceability mechanisms have been recognized as an important prerequisite for dealing with critical software qualities such as maintainability and evolvability. In their paper "Value-Based Requirements Traceability: Lessons Learned", Alex Egyed and colleagues focus on the problem of deciding what traceability strategy to adopt for near and long-term utilization of software. In other words, what traceability links should the software engineer maintain to serve short-term maintenance processes for a software system, but also its long-term evolution. The research methodology adopted for this work is empirical: Three case studies are presented and lessons are drawn from the traceability practices adopted in each case.

The second paper of this section is authored by Tetsuo Tamai and Mayumi Tamata and is titled "Impact of Requirements Quality on Project Success or Failure." It has long been conjectured that there is a causal relationship between the quality of requirements and success or failure of a software project. The paper offers concrete evidence concerning this conjecture. Specifically, the paper draws on survey data from 72 industrial projects, including data on the quality of their requirements specifications and performance data indicating whether projects had incurred time/cost overruns and uses statistical techniques to draw conclusions about the strength of the relationship between requirements quality and project success/failure.

The third paper by Vera Kartseva et al. reports of a long-term effort to introduce into the design of networked enterprises – a particular form of software-intensive systems -- concepts that represent economic value. It seeks to make facets the basis for requirements analysis techniques that can complement goal and process-oriented

approaches. The paper is titled "Designing Value-Based Inter-Organizational Controls Using Patterns." It focuses on the specific problem of introducing control mechanisms that can discourage selfish, opportunistic inter-organizational behaviors by members of the networked enterprise. As indicated by the title, the approach is pattern-based and it demonstrates through a case study how inter-organizational controls can be realized in terms of three project-based patterns that increase economic value.

Value-Based Requirements Traceability: Lessons Learned

Alexander Egyed[1], Paul Grünbacher[1], Matthias Heindl[2], and Stefan Biffl[3]

[1] Institute for Systems Engineering and Automation, Johannes Kepler University,
A-4040 Linz, Austria
`{ae,pg}@sea.uni-linz.ac.at`
[2] Siemens IT Solutions and Services, Siemens Austria, Gudrunstraße 11, A-1100
Vienna, Austria
`matthias.a.heindl@siemens.com`
[3] Institute for Software Technology and Interactive Systems, Vienna Univ. of Techn.,
A-1040 Vienna, Austria
`Stefan.Biffl@tuwien.ac.at`

Abstract. Traceability from requirements to code is mandated by numerous software development standards. These standards, however, are not explicit about the appropriate level of quality of trace links. From a technical perspective, trace quality should meet the needs of the intended trace utilizations. Unfortunately, long-term trace utilizations are typically unknown at the time of trace acquisition which represents a dilemma for many companies. This chapter suggests ways to balance the cost and benefits of requirements traceability. We present data from three case studies demonstrating that trace acquisition requires broad coverage but can tolerate imprecision. With this trade-off our lessons learned suggest a traceability strategy that (1) provides trace links more quickly, (2) refines trace links according to user-defined value considerations, and (3) supports the later refinement of trace links in case the initial value consideration has changed over time. The scope of our work considers the entire life cycle of traceability instead of just the creation of trace links.

Keywords: Requirements engineering, software traceability, value-based software engineering.

1 Introduction

Trace links define dependencies among key software artifacts such as requirements, design elements, and source code. They support engineers in understanding complex software systems by identifying where artifacts are implemented [12]. A significant body of work has been published on software traceability and its usefulness, particularly on requirements traceability [12,19]. Traceability has also made its way into a number of software engineering standards and initiatives, such as ISO 15504 and the CMMI that mandate or recommend traceability as 'best practice'.

Over the last couple of years we have been collaborating in the area of requirements traceability with several industry partners in the US and Europe. These included large organizations such as Siemens Austria [13], Boeing Company, and NASA but also very small companies such as geDV [17]. We have seen that companies introducing traceability techniques face significant challenges. Capturing trace links requires a significant effort even for moderately complex systems [19]. While some automation exists for trace capture it still remains a mostly manual process of non-linear complexity (i.e., if m requirements trace to n pieces of source code then there are m times n potential trace links). Even worse, once generated, trace links degrade over time as the software system evolves. While trace links are often utilized immediately, they still have to be maintained continuously thereafter to remain useful over time. Maintaining trace links after changes to the system is a continuous, manual task, also of quadratic complexity.

In practice, engineers rarely capture trace links completely because of the uncertainty of their later usage. Engineers thus attempt to predict what trace links will likely be needed in the future and concentrate on these. If the trace utilization is planned for the near term their predictions tend to be good. There is, however, a significant uncertainty concerning long-term utilization. Engineers thus often have to err on the side of perfection (i.e., producing links that are not needed later) or on the side of incompleteness (i.e., omitting links that are needed later). Both errors can be costly. The practical alternative, i.e., to delay trace acquisition and to create trace links upon request, is typically infeasible because the manual generation and maintenance of trace links hinges on the engineers' ability to recollect necessary facts. Asking for traceability information long after the fact is like asking for documentation as soon as somebody decides to read it.

The dilemma of requirements traceability is thus that engineers must consider both the near-term and long-term utilization needs of trace links.

This chapter presents lessons learned from three case studies. The lessons suggest that a traceability strategy should first identify trace links quickly and completely on a coarser level of granularity and then refine them according to some user-defined value consideration (i.e., based on predicted utilization needs). Engineers may still err during the refinement – if value considerations change or turn out to be incorrect – however, our lessons learned indicate that engineers are able to recover from such errors more quickly if they can rely on a complete set of coarser-grained trace links. Software maintenance costs represent a significant share of the total software cost so our analysis tackles an important industry problem [5]. This chapter builds on an initial workshop paper [4] describing the proposal for this work and a short conference paper discussing some of the basic points [11]. The novel contributions in this chapter are the detailed data of the three case studies together with lessons learned that can be used directly by practitioners in their projects.

The chapter is structured as follows: Section 2 discusses related work. Section 3 explores and illustrates the problem in greater detail. Section 4 presents lessons learned in the three case studies and detailed analyses of data. Section 5 discusses results. We round out the chapter with conclusions and an outlook on further work.

2   Related Work

Numerous commercial and research approaches are available that support requirements traceability. Most approaches focus on the acquisition and management of trace links: Tool support exists in two forms: (1) Tools for capturing and managing trace links provide a structured way of defining trace links and keeping track of missing or potentially outdated links. Many commercial tools primarily focus on these aspects of recording and replaying trace links. The tools have very limited capabilities to automate the definition of trace links, which remains a manual and error-prone activity. (2) Tools for automating the acquisition of trace links are typically based on providing some basic facts initially followed by some deductive reasoning to fill-in missing pieces. Examples of approaches in this category are based on scenarios [9] or information retrieval techniques [14]. Such tools are sometimes based on heuristics and neither complete nor correct results can be guaranteed.

Despite existing tool support traceability remains costly and complicated. There are typically no guarantees of completeness or correctness (errors and omissions). Developers can neither fully trust automatically generated trace links nor manually defined ones.

Traceability approaches typically do not provide explicit support for trace utilizations such as impact or coverage analysis. They rather provide general-purpose features to create reports or query traceability information. Researchers have been proposing techniques to improve support for important tasks such as analyzing change impacts [1] or understanding the conflict and cooperation among requirements [10].

There is very little literature on the quality implications of trace links. This is partly because there are many applications that benefit from trace links with different cost/quality trade-offs. An exception is [2]: Bianchi *et al.* show that the effectiveness of maintenance can be improved by varying the degree of granularity of the traceability model. A reason for the lack of research results lies in the fact that it is unknown in advance which trace links will be used.

Murphy *et al.* have explored the idea of "good enough" techniques in the context of closing the gap between design and implementation. Their reflexion model technique helps engineer evolving a structural mental model of a system. As soon as this model is "good enough" it can be used for reasoning about tasks such as design conformance, change assessment, and experimental reengineering.

Recently, several publications appeared in the area of value-based software engineering [3,6]. This thread of research provides a new perspective on balancing cost and benefits of software engineering techniques. An initial cost-benefit analysis for traceability is reported in [13]. Cleland-Huang *et al.* have also started exploring the economic aspects of traceability [7]. In [15], Lindvall and Sandahl show in a case study that tailoring of traceability models is essential in practice.

The literature review reveals that many traceability approaches emphasize isolated aspects (e.g., trace generation) but do not consider the full life cycle "end-to-end" traceability. Furthermore, the impact of different quality levels of trace links is not yet well understood. There is still no systematic way for

understanding the value contribution of trace links and for dealing with cost-quality trade-offs.

3   Problem Illustration

To illustrate the problems in more detail we take a look at the life cycle of software traceability. This life cycle includes in essence four tasks:

Acquisition. Software engineers create trace links between requirements and other software artifacts such as design elements, or source code either manually or with the help of tools.

Utilization. Software engineers consume trace links in tasks such as change impact analysis, requirements dependency analysis, etc. It is useful to distinguish between short-term utilization (e.g., determining test coverage in later project stages) and long-term utilization (e.g., a particular change request years later).

Maintenance. Software engineers continuously revisit and update trace links as the system and its various artifacts (requirements, design elements, code, etc.) are being changed. Trace maintenance ensures that the quality of trace links does not degrade.

Enhancement. Software engineers improve the quality of trace links (i.e., increasing completeness or correctness) in case their quality is insufficient for the intended utilization.

Better tools, more capable engineers, more calendar time, or better documentation are certainly helpful in improving the quality and reducing the cost of traceability in any or all tasks. But, in essence these measures do not mitigate the following fundamental problems:

Finding the right level of trace quality with finite budgets. Even if developers have some quality threshold in mind, it is not obvious whether the allocated budget is sufficient for the planned traceability task. For example, it is not obvious that improving trace links is cost-efficient as the benefits gained through trace utilization are offset by the added cost of producing better trace links.

Increasing the quality of trace links comes at an increasingly steep price. Trace acquisition suffers from a diseconomy of scale where low-quality trace links can be produced fairly quickly and economically while perfection is expensive and very hard to achieve and determine.

Traceability planning under uncertainty about future utilization needs. We cannot know which trace links at which level of quality will be needed in the future as detailed knowledge about applications utilizing them is not available at the time of their creation.

An engineer performing a traceability task typically faces a situation where the time available to complete the task is much shorter than the time required performing the task in a complete, correct, and consistent manner. Basically, the engineer has two fundamental strategies to deal with the problem:

"Brute force", i.e., trying to generate the trace links for the complete system in the limited time available. Obviously this will have some negative impact on

Table 1. Impact of "Brute force" vs. "elective" strategies on traceability tasks

Task	Brute force strategy	Selective strategy
Acquisition	Full coverage but many incorrect links.	Correct links but small coverage (area and/or depth).
Utilization	Utilization hampered by erroneous feedback caused by incorrect links. Similar performance for short-term and long-term utilization.	Good utilization limited to available links. For other utilizations problematic or infeasible. Likely emphasizing on short term utilization.
Maintenance	Hampered by incorrect links (lack of trust in trace links).	Limited to smaller set of available trace links.
Enhancement	Hampered by incorrect links (lack of trust in trace links).	Very hard due to later unfamiliarity.

the correctness of trace links and their later utilization if the allocated time is insufficient as is often the case.

"Selective", i.e., trying to achieve the most valuable traces until running out of resources driven by an explicit or implicit value prioritization strategy such as easy-things-first, gut feeling, or predicted future utilization. As a result of applying the strategy some parts of the system will have trace links of reasonable quality while other trace links will be missing or incorrect. This can limit or in some cases even preclude future utilization.

Table 1 summarizes the impact of the two strategies on the traceability tasks discussed earlier.

The brute force strategy negatively affects trace utilization because of the higher degree of incorrectness. While there are typically some errors in trace links due to the complexity involved in generating them, the brute force strategy worsens this situation. Trace maintenance also suffers from higher incorrectness because it builds on the presumed correct trace links. Trace enhancement is, however, unnecessary. The selective strategy is better but failure in predicting trace utilization needs will make enhancement difficult. This is less of a problem for near-term utilization because the engineers are still available and knowledgeable to recover missing trace links. However, the selective strategy suffers immensely during long-term utilization because recovering trace links later is very difficult (i.e., if engineers moved on or forgot necessary details). The most serious drawback of these two approaches is the inability to recover from missing or incorrect traces.

4   Lessons Learned in Three Case Studies

We describe lessons learned based on data from three case studies. The purpose of the lessons is to guide practitioners to define a traceability strategy and to understand the expected benefits. Our intention is not to propose a concrete approach or improve upon a specific tool or traceability technique. Rather, we demonstrate that value-based software engineering techniques can have an

Table 2. Case studies and their context

System	Development context
ArgoUML	UML modeling tool; distributed developers; open-source development
Siemens Route Planning	Route-planning application for public transportation; industrial developer team
Video on demand	Movie player; single developer

impact in very practical terms. We will discuss what trace links to generate at which time and at what level of detail. We will not discuss how to generate them. Any existing guidable process or tool should be able to adopt our lessons learned.

We derive the lessons from three case studies: The open-source ArgoUML tool [18], an industrial route-planning application from Siemens Corporation, and an on-demand movie player. ArgoUML is an open-source software design tool supporting the Unified Modeling Language (UML). The Siemens route-planning system supports efficient public transportation in rural areas with modern information technologies. The Video-On-Demand system is a movie player allowing users to search for movies and playing them [8]. We chose these systems because they cover a range of different development contexts (open source vs. industrial), application characteristics (large vs. small), and domains (see Table 2).
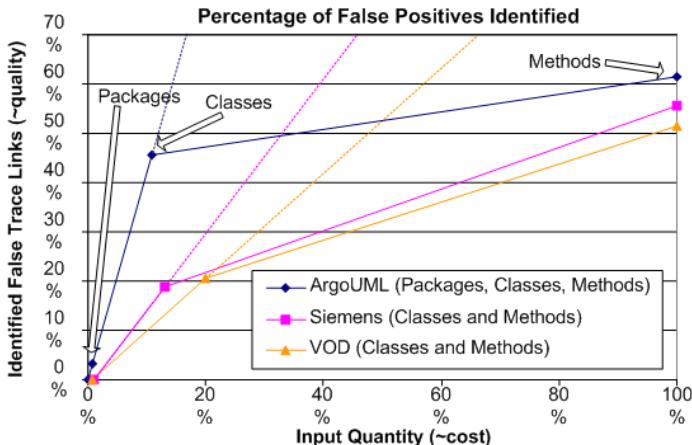
4.1 Reducing Granularity

Lessons learned:

- *Save traceability effort by reducing granularity.*
- *Requirements-to-class-level granularity provides better value for money compared to requirements-to-package-level and requirements-to-method-level granularity.*
- *Focus on completeness and correctness first – these are essential for follow up tasks such as maintenance and enhancement.*
- *Reducing granularity reduces the benefits of trace utilization.*

We learned in the case studies that combining value-based and granularity-based trace acquisition is a good strategy. Granularity is the level of precision of a trace link (e.g., requirements to packages vs. requirements to classes). Adjustment of granularity can provide a cheap and quick way of exploring correct and complete requirements-to-code traces. We will see that granularity-based trace analysis may be imprecise but it can be computed much more efficiently without sacrificing correctness and completeness.

The trace links considered in the three case studies were between requirements and source code. In some development contexts, engineers might define trace links to the granularity of methods (e.g., Java methods in case of the ArgoUML system). Sometimes, engineers might even decide to define trace links to

Fig. 1. Decreasing number of false positives with increasing level of detail

individual lines of code (e.g., when exploring crosscutting concerns in safety critical systems). The needs of the techniques that utilize traces links normally drive this decision. The benefit of adopting coarse-grained trace links is better coverage and higher quality of trace links at lower costs. However, there is a sacrifice: Low granularity trace links are not as precise and useful during trace utilization. We have found, however, a remedy to this issue which will be discussed later.

We analyzed the granularity trade-off for the three case study systems. Cost was measured in terms of the effort required and the input quantity generated. We considered the following three levels of granularity: requirements-to-methods, requirements-to-classes, and requirements-to-packages. Quality was measured in terms of the number and percentage of false positives. In particular, for each case study system we analyzed the impact of trace acquisition on the quality of the generated trace links. As a baseline, we took the level of false positives produced on the most detailed level of granularity (i.e., requirements-to-methods). The analysis compared how a reduction of granularity resulted in a higher number of false positives (note that a reduction in granularity does not cause false negatives – missing trace links).

Figure 1 presents our findings for the three levels of granularity and the three case study systems. For example, the ArgoUML system consisted of 49 packages, 645 classes, and almost 6,000 methods. The quantity of trace links captured at the granularity of Java classes was thus only one-tenth the order of magnitude compared to the quantity at the granularity of methods. This reduction in input quantity obviously also reduced the effort spent: we observed a three-fold reduction in the effort needed to generate the coarser-grained trace links, a very significant saving. However, this saving came at the expense of trace quality. Figure 1 also shows the quality drop relative to the total number of traces. We found that the trace links at the granularity of classes had 16% more false positives compared to the ones at the granularity of methods. This effect was much stronger on the granularity of packages which had over 40% more false positives

Table 3. Granularity-based tracing

Task	Impact of lower granularity
Acquisition	Correct and complete but less precise.
Utilization	Reduced quality but similar performance for short-term and long-term utilization.
Maintenance	Easier because of smaller quantity, completeness, and correctness.
Enhancement	Limited because can ignore code not identified on a coarser-grained trace.

with another ten-fold reduction in input quantity (20/30% more false positives on the level of classes and 55% more false positives on the level of packages – no packages were defined for the movie player).

Our data strongly indicates that there is a decreasing marginal return on investment (ROI) with finer-grained input. Indeed, the data strongly suggest that the granularity of classes provides the best cost/quality trade-off. Adjusting the level of granularity can be used as a cost saving measure and some techniques that utilize trace links would still produce reasonable results. However, it is the ability to provide complete and correct trace links at lower costs that is of particular interest in this chapter. Table 3 summarizes the benefits of granularity-based traceability for the four tasks of the trace life cycle.

Trace Maintenance heavily utilizes trace links. For example, if the change of a requirement requires a code change then the engineer uses trace links to identify the affected pieces of code. As a result of the change the requirement may then map to more or fewer elements of code (classes or methods). Trace maintenance ensures that the trace link is updated to reflect the new, correct, and complete relationship. Trace links on the granularity of Java classes identify all classes that need to be changed (i.e., because they are complete and correct). The lack of precision implies that the engineer must search through the methods of the identified classes but does not have to study the remaining classes not identified by the trace (no false negatives!). Consequently, the maintenance of coarse-grained trace links does not suffer from the problems identified in Section 3 where a significant portion of the source code had to be searched because of incompleteness. The following demonstrates the strong savings this entails.

In the ArgoUML case study we studied 38 requirements in detail. On average, a requirement traced to 247 methods and 46 classes. Clearly, these requirements were not trivial. It required roughly one-third of the effort to produce the coarser-grained requirements-to-classes traces as compared to the requirements-to-method traces. With that effort, the selective approach described in Section 3 only identifies 30% of the requirement-to-methods traces. With that level of effort, the remaining 70% (or 4,200) traces to methods would remain incomplete. While the requirements-to-classes traces identify all classes owned by a given requirement, the requirements-to-method traces would only identify some 30% of methods.

This leaves it up to the developer to guess the missing information. With the requirements-to-class traces, we know that only 46 classes (out of 645) are

affected in average by a requirements change. Since there is a 1:10 ratio of classes to methods in the ArgoUML system, an average of 460 methods have to be explored in more detail. The selective approach, on the other hand, guesses 30% of the methods correctly but misses out on the more than 4000 other methods that were not covered. It is much easier to refine 46 imprecise classes to methods than it is to discover 250 or so methods in a pool of over 4000 methods that were ignored because of incompleteness – this represents a 20-fold improvement in complexity and cost! This example illustrates that it is more beneficial to have completeness than precision for maintenance. The Siemens system and Video-on-demand system behaved similarly well.

Trace Enhancement benefits similarly from coarse-grained trace links. Recall from Section 3 that trace enhancement is necessary when an engineer discovers missing or incorrect trace links. This problem is obviously reduced or perhaps eliminated as the 3-fold reduction in effort increases the likelihood of correct and complete trace links at the granularity of classes. However, trace enhancement is still needed for upgrading the coarse-grained trace link to fine-grained trace links on demand. Thus, the problem has changed somewhat. Similar to trace maintenance, trace enhancement utilizes its own trace links. Only the classes identified by the trace link need to be refined but not all the other classes. Again, trace enhancement does not suffer from the problems of the selective strategy identified in Section 3 where a significant portion of the source code has to be searched because of incompleteness. The empirical data supporting this benefit is omitted because it is largely identical to the discussion under trace maintenance above.
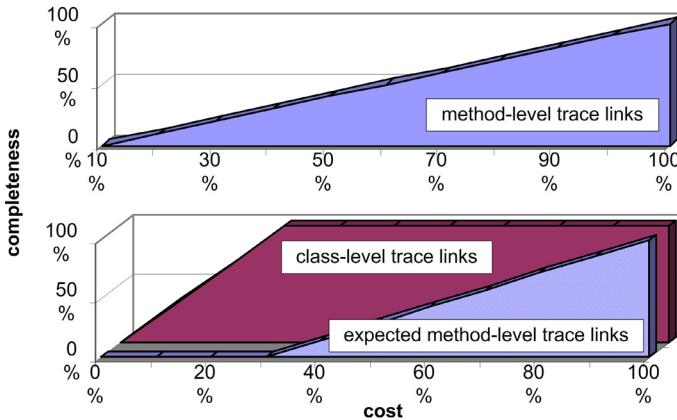
Trace Utilization techniques differ with respect to handling false or missing trace links. This chapter does not provide a comprehensive overview here due to brevity. If the quality of the trace links is inadequate then trace enhancement is necessary and we already demonstrated that coarse-grained trace links outperform selective trace links during trace enhancement 20-fold. However, selective trace links perform better if the engineer is able to guess correctly which links will be needed. In our experience, this is rarely the case for long-term trace utilization. Only if trace links are generated for immediate use without later reuse, the selective approach may outperform our proposed strategy.

4.2   Value-Based Enhancements

Lessons learned:

– *Consider stakeholder value propositions to focus traceability for refinement.*
– *Use savings from lower granularity to focus trace refinement on high-value requirements.*

Thus far, we demonstrated that granularity-based trace links can save substantial cost during trace maintenance and enhancement. However, the resulting across-the-board quality reduction may not be acceptable. While an engineer may be willing to sacrifice some benefits to save cost, we believe that such a

Fig. 2. Trace Acquisition with Selective (top) and Value-based Approach (bottom)

process must be guidable. In the following, we present a value-based extension to granularity-based trace acquisition, maintenance, and enhancement:

Value-based software engineering [3] invests effort in areas that matter most. A value-based approach relies on considering stakeholder value-propositions to right-size the application of a software engineering technique [3,5]. The definition of value depends largely on the domain, business context and company specifics. In essence, engineers can place value directly on trace links (i.e., this trace is important) or indirectly on the artifacts they bridge (i.e., this requirement is important and consequently its trace link to code as well). Our approach does not prescribe a particular value function. Indeed, the selective strategy discussed earlier could be considered a value-based approach, except, that the selective approach accepted incompleteness which we discourage in the value-based approach. The following demonstrates that not the initial acquisition but the enhancement should be value driven.

We discussed trace enhancement as a method for improving the precision of trace links in Section 3. Trace enhancement can be done at a later stage when an engineer discovers missing traces or incorrect traces. Trace enhancement can also be done early on to "upgrade" high-value traces as initially all traces including the high-value ones are only coarse grained.

Figure 2 (top) shows that the cost of trace acquisition for the selective approach maps directly to completeness. A 40% cost investment implies, in average, 40% completeness. Value-based trace acquisition follows a different pattern. We previously discussed that coarse-grained trace acquisition consumes in average 30% of the cost of fine-grained trace acquisition [13]. Figure 2 (bottom) depicts that with 40% of cost we reach 100% completeness on the class-level but have not yet created a single fine-grained trace link. The additional 10% funding can thus go into trace enhancement. Since trace enhancement is part of trace acquisition, we would expect it to be as effective as the selective approach but trace enhancement only has to refine those classes identified by coarser-grained trace links. As a consequence, if 100% of the cost is invested, we would expect the
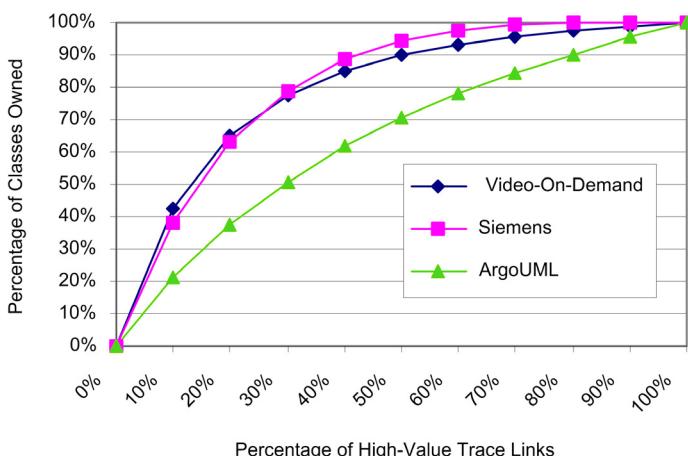
value-based and selective approaches to be roughly equivalent. However, this expectation is wrong in two ways: (1) Few high-value traces result in most source code covered creating a negative effect; (2) Only overlapping trace links need to be enhanced for creating a positive effect. Both issues are discussed next.

Understanding the Diseconomy of Scale. Intuition says that if only half the trace links are important then only half the trace links need to be refined to a finer level of granularity – thus saving 50% of the cost. This intuition is misleading because requirements map to large portions of source code and each piece of code may be related to multiple requirements.

Above we presented details on the 38 requirements-to-code traces for ArgoUML. These 38 requirements covered 4,752 methods (roughly 80% of the ArgoUML source code) with the average trace covering 248 methods.

Trace acquisition is usually not done by taking a requirement and guessing where it might be implemented. Typically, trace acquisition iterates over the source code, one class/method at a time, and reasons to which requirement(s) it belongs. For the ArgoUML system, we found that a class was related to 3.2 requirements in average. If only one of these three requirements was important then the class would need to be refined. The likelihood for this increased non-linearly.

Figure 3 depicts the percentage of classes that were traced to by at least one high-value trace link in relationship to the percentage of high-value trace links. The cost is normalized across all three case studies and it can be seen that the cost varies somewhat although it is similarly shaped. The x-axis depicts the percentage of high-value trace links. A high number of high-value trace links increases the number of classes they own collectively. However, we also observe a diseconomy of scale. For example, if 40% of the ArgoUML trace links are of high value then half of its classes (i.e., more than 40%) are owned by them. Consequently, 50% of the classes need to be refined. The other two case studies

Fig. 3. Diseconomy by Enhancing Classes belonging to a High-Value Trace Link
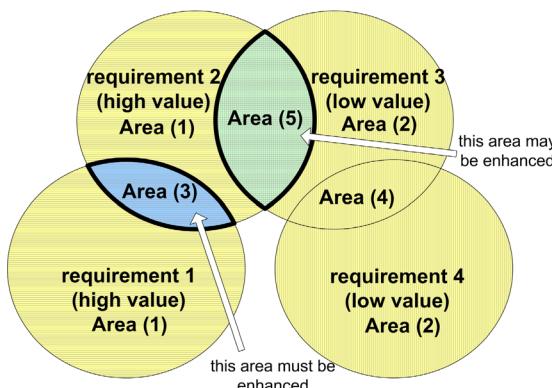
behaved much worse. In both cases, 40% of the high-value trace links owned almost 80% of the classes.

This diseconomy of scale seems to invalidate the benefits of value-based trace acquisition. This diseconomy is certainly another reason why the simple selective strategy discussed earlier was not desirable. In the case of the Siemens and VOD systems, the cost for trace enhancement for 40% high-value requirements is almost as high as doing the enhancement for all requirements.

Enhancing Common Classes. Fortunately, there is also a positive effect that counters the diseconomy of scale discussed above. We made the trivial assumption that every class owned by a high-value trace link must be refined to the granularity of methods. This is in fact not necessary. Figure 4 depicts four trace links: two high-value trace links covering requirements 1 and 2; and two low-value trace links covering requirements 3 and 4. Each circle represents the set of classes traced to by each requirement. These requirements "share" some classes, i.e., their traces overlap in their common use of classes as indicated by the intersecting circles but also own classes they do share with other requirements [9].

The question is which of the classes in the various areas (overlapping or not) in Figure 4 must be refined to a finer level of granularity. We distinguish five areas: (1) classes owned by a single high-value requirement; (2) classes owned by a single low-value requirement; (3) classes shared among high-value requirements; (4) classes shared among low-value requirements; and (5) classes shared among multiple requirements including one high-value requirement (if there are multiple high-value requirements than area 3 applies).
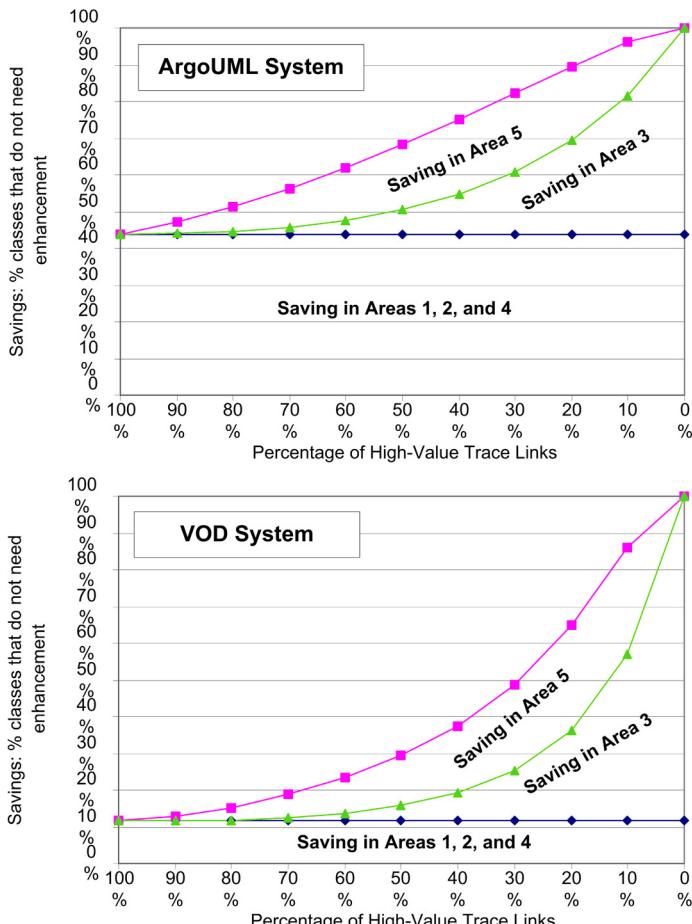
Obviously, classes owned by low-value requirements (area 2) or shared among low value requirements (area 4) should not be enhanced. However, even classes owned by single high-level requirements (area 1) do not need to be enhanced. We discussed previously that coarse granularity is correct and complete. Thus if a class is owned by a single requirement then all its methods must be owned by this artifact (i.e., if the class is not shared then its methods cannot be shared

Fig. 4. Detailed Granularity is Only Necessary for Overlaps Involving Higher-Value Trace Links

either). However, classes owned by multiple high-level requirements (area 3) must be enhanced because we cannot decide what methods are owned by the one requirement versus the other. Only overlaps between a single high-level class and one or more low-level classes (area 5) represent a gray zone. Most techniques do not benefit from the enhancement of area 5. In those cases, defining area 5 for one requirement but no other is a waste also.

These observations lead to substantial savings with no loss in quality. Figure 5 depicts the results for the ArgoUML case study (top) and the VOD case study (bottom); the Siemens results are similar. Over 45% of the 645 classes of the ArgoUML were owned by single requirements (areas 1, 2, and 4). These classes did not need to be refined to a finer level of granularity. This resulted in an instant saving of 12-45% effort depending on the case study. This saving was independent of the percentage of high-value trace links.

Fig. 5. Effort Saved due to Value Considerations in the Source Code (top: ArgoUML, bottom: VOD)

In addition, depending on the percentage of high-value trace links, we saved on the overlapping areas 3 and 5 (itemized separately). For example, if 40% of the trace links were of high value then an additional 39% of input effort was saved (in both case studies) because many of the overlapping areas did not require fine-grained input. We will discuss later that in our experience between 15-50% of trace links are typically of high value. Based on our case studies, this translates to 30-70% savings during enhancement compared to a value-neutral approach. This extra saving comes at no expense in terms of trace link quality.

So not only trace maintenance benefits from a complete, coarse-grained trace acquisition on the granularity of classes. Even the enhancement of trace links during trace acquisition benefits from it because it piggybacks from the results obtained on the coarser granularity to decide where to refine. From the 645 classes of the ArgoUML case study only 134 classes needed to be refined to the granularity of methods. Given that there were in average 9.2 methods per class in the ArgoUML system 1,232 methods needed to be looked at in more detail compared to 6,000 methods in case of a value-neutral approach. We reduced the effort by 80%.

5   Discussion

It is well known that most software development effort goes into maintenance and evolution [5]. Consequently, more effort is spent on maintaining trace links or enhancing them than on acquiring them. This leads to a final lesson: *"Don't focus on saving effort during trace acquisition without considering trace maintenance and enhancement."*

However, while working and interacting with industrial partners (including Siemens Austria, Boeing Company, NASA, and geDV) we did observe legitimate reasons for wanting to limit trace acquisition efforts. Limited time and budget were obviously the more dominant reasons. The lessons we presented suggest investing traceability effort on what matters most and provide a way of limiting trace acquisition without seriously impeding trace maintenance and enhancement.

Granularity Trade-Off. Section 4 demonstrated that reducing the granularity of trace links reduces the complexity of trace acquisition by a factor of ten and cost by a factor of three. Coarse-grained trace acquisition produces correct and complete trace links (though imprecise ones) three times as fast. These links are thus more likely to be available early on. During trace maintenance and enhancement, they are mostly useful because of their ability to correctly identify where requirements do not trace to. We demonstrated that an average requirement change may affect dozens, even hundreds of methods. While this impact seems large, one must consider that this is only a small percentage of the total number of methods – the ArgoUML system defined almost 6,000 methods. While coarse-grained trace links may not identify the individual methods well (it errs by a factor of 2), it nevertheless identifies the much larger set of methods not needed to be looked at because if a requirement does not trace to a class then it also does not trace to any of its methods.

Value-based Trade-Off. If we would not have considered trace utilization, we could have stopped here. However, coarser-grained trace links are not as useful for trace utilization as finer-grained trace links. It is trace utilization that drives the quality needs of trace links. We thus demonstrated how to refine the granularity of trace links on a selective basis. This refinement process is guided by the engineer; however, we believe that it should follow value-based criteria. Interestingly, value-based trace acquisition does not appear scalable on a first glance. The problem is that classes and methods are often shared among multiple requirements. This causes a diseconomy of scale in that few high-value requirements, collectively, may own a larger share of the source code. Indeed, we found that 40% high-value requirements own between 50-80% of the source code. Under these circumstances, little is gained by following a value-based approach to trace acquisition. Fortunately, we also found that not every class traced to by a high-value trace link must be refined. In fact, only classes traced to by at least two high-value trace links needed to be refined. This resulted in a saving of 30-70% compared to a value-neutral approach. Of course, not even a value-based approach can guarantee accurate prediction of trace links that will be needed later. Trace enhancement, which is enabled by our strategy, improves the precision of a trace link that was incorrectly identified as a low-value trace link. It must be noted that our strategy does not identify high value trace link. This is done by the customer and/or engineer.

Tool support. The lessons presented do not prescribe a particular method or tool for doing trace acquisition, maintenance, and enhancement. The three strategies outlined in this chapter, brute-force, selective, and value-based, are applicable to any method or tool that is guidable (and most are guidable). However, trace acquisition is a mostly manual process and the value-based strategy does not require the engineer to change how to perform these manual tasks either. Rather, it guides them what traces should be done when and at what level of granularity but leaves it up to the engineer how this should be done.

Cost and Effort. We advocate that trace acquisition should always be done completely. The minimal investment of trace acquisition is the cost/effort needed to complete trace acquisition on a coarser level of granularity.

Correctness. A value-based strategy can significantly save cost and effort. This saving does not come at the expense of the quality among the higher-value trace links. All higher-value trace links are produced at the highest quality. Only low-value trace links are produced at a lower level of quality; but we have seen that even this quality reduction is moderate (between 15-30% more false positives depending on the case study).

Human Error. It must be noted that we ignored the issue of human error in this chapter. There is always some degree of error associated with trace links. The degree of error depends on a range of factors such as engineer's experiences, engineer's ability to recollect facts about the artifact and/or code to be traced or even tool errors. This chapter ignored these kinds of errors because it is not affected by our value-based strategy. Recall that this work does not prescribe a

Table 4. Requirements value based on typical ranges for business value feasibility

Value/Feasibility	Easy to realize (40% to 60%)	Hard to realize (40% to 60%)
high (30% to 80%)	High value (12% to 48%)	Low value (12% to 48%)
Low (20% to 70%)	Low value (12% to 48%)	Low value (12% to 48%)

different tool, method, or engineer for doing traces. This chapter only suggests what to do when.

Finding your value function. As pointed out above our strategy relies on the ability to identify the important trace links. We previously argued that typically between 15-50% of trace links are important (high value). This data is based on previous work on the importance of requirements during the software lifecycle. It is important to stress that arbitrary, user-definable utility functions can be used to experiment with different scenarios. The possible savings depend on the ability of the function to predict short-term and long-term utilization.

We found that the priorities of requirements represent a good proxy for stakeholder value. For software traceability, value-based software engineering means to produce better quality trace links for higher-value requirements or other valuable artifacts. It is common practice in industry to prioritize requirements or design elements according to their importance and feasibility ratings from success-critical stakeholders. Important requirements that are easy to realize have a higher value than unimportant requirements that are hard to realize. Table 4 indicates this relationship together with typical percentages for business value and ease of realization.

This utility function is just one example. However, we believe this function is reasonable in many contexts. Obviously utility functions can be optimized and calibrated to allow even higher savings. The dilemma is that it is not possible to devise a perfect utility general-purpose function that tells about short term and long term needs. Granularity allows us to save money in the short term and to significantly benefit the maintenance and later enhancement in the long run. A value-based approach to traceability is most likely to strike the right balance between the cost and benefits of traceability.

6   Conclusion

In this chapter we have presented lessons learned from three case studies. The lessons suggest a value-based approach to software traceability: Spending the money where it matters most (value); exploring trace links incrementally based on an initial, complete base of trace links; and considering trace utilization, maintenance, and enhancement. A value-based approach does not suggest ignoring trace links. On the contrary, this work strongly advocates the completeness and correctness of trace links. In fact, it is the goal of this work to accomplish completeness and correctness as quickly as possible, even at the expense of precision, and to then enhance (refine) the trace links as the budget allows.

We believe that neglecting these lessons will lead to higher cost and inappropriate trace links. Ad-hoc trace generation may have some immediate benefits but is bound to result in more disadvantages over the course of the software development life cycle and its maintenance. Our value-based strategy tells when to establish which traces but it does not tell how to do trace acquisition. It thus applies to any existing traceability method or tool that is guidable.

References

1. Abbattista, F., Lanubile, F., Mastelloni, G., Visaggio, G.: An experiment on the effect of design recording on impact analysis. In: Int. Conf. on Software Maintenance, pp. 253–259 (1994)
2. Bianchi, A., Visaggio, G., Fasolino, A.R.: An Exploratory Case Study of the Maintenance Effectiveness of Traceability Models. In: Proc. of the 8th Int. Workshop on Program Comprehension, p. 149 (2000)
3. Biffl, S., Aurum, A., Boehm, B.W., Erdogmus, H., Grünbacher, P.: Value-based Software Engineering. Springer, Heidelberg (2005)
4. Biffl, S., Heindl, M., Egyed, A., Grünbacher, P.: A Value-Based Approach for Understanding Cost-Benefit Trade-Offs During Automated Software Traceability. In: Proc. of the 3rd Int. Workshop on Traceability in Software Engineering, Long Beach, CA (2005)
5. Boehm, B.W., et al.: Software Cost Estimation with COCOMO II. Prentice Hall, Englewood Cliffs (2000)
6. Boehm, B.W., Huang, L.: How Much Software Quality Investment Is Enough: A Value-Based Approach. IEEE Software 23(5), 88–95 (2006)
7. Cleland-Huang, J., Zemont, G., Lukasik, W.: A Heterogeneous Solution for Improving the Return on Investment of Requirements Traceability. In: Proc. of the Int. Conf. on Requirements Engineering (RE), Kyoto, Japan, pp. 230–239 (2004)
8. Dohyung, K.: Java MPEG Player (1999),
   `http://peace.snu.ac.kr/dhkim/java/MPEG`
9. Egyed, A.: A Scenario-Driven Approach to Trace Dependency Analysis. IEEE Transactions on Software Engineering (TSE) 29(2), 116–132 (2003)
10. Egyed, A., Grünbacher, P.: Identifying Requirements Conflicts and Cooperation. IEEE Software 21(6), 50–58 (2004)
11. Egyed, A., Heindl, M., Biffl, S., Grünbacher, P.: Determining the Cost-Quality Trade-off for Automated Software Traceability. In: Proc. 20th IEEE/ACM Int. Conf. on Automated Software Engineering (ASE), Long Beach, CA (2005)
12. Gotel, O.C.Z., Finkelstein, A.C.W.: An Analysis of the Requirements Traceability Problem. In: Proc. of the 1st Int. Conf. on Rqts Eng., pp. 94–101 (1994)
13. Heindl, M., Biffl, S.: A Process for Value-based Requirements Tracing – A Case Study on the Impact on Cost and Benefit. In: Proc. of the European Software Engineering Conf. and Foundations of Software Engineering (ESEC/FSE), Lisboa, Portugal (September 2005)
14. Huffman Hayes, J., Dekhtyar, A., Osborne, J.: Improving requirements tracing via information retrieval. In: Int. Conf. on Requirements Engineering (2003)
15. Lindvall, M., Sandahl, K.: Practical Implications of Traceability. Journal on Software – Practice and Experience (SPE) 26(10), 1161–1180 (1996)
16. Murphy, G.C., Notkin, D., Sullivan, K.J.: Software reflexion models: bridging the gap between design and implementation. IEEE Transactions on Software Engineering 27(4), 364–380 (2001)

17. Neumüller, C., Grünbacher, P.: Automating Software Traceability in Very Small Companies: A Case Study and Lessons Learned. In: 21st Int. Conference on Automated Software Engineering, Tokyo (September 2006)
18. Robins, J., et al.: ArgoUML, `http://argouml.tigris.org/`
19. Ramesh, B., Stubbs, L.C., Edwards, M.: Lessons learned from implementing requirements traceability. Crosstalk – Journal of Defense Software Engineering 8(4), 11–15 (1995)

Impact of Requirements Quality on Project Success or Failure

Tetsuo Tamai[1] and Mayumi Itakura Kamata[2]

[1] The University of Tokyo
tamai@acm.org
[2] Tokyo Research Laboratory, IBM Research
mitakura@jp.ibm.com

Abstract. We are interested in the relationship between the quality of the requirements specifications for software projects and the subsequent outcome of the projects. To examine this relationship, we investigated 32 projects started and completed between 2003 and 2005 by the software development division of a large company in Tokyo. The company has collected reliable data on requirements specification quality, as evaluated by software quality assurance teams, and overall project performance data relating to cost and time overruns. The data for requirements specification quality were first converted into a multiple-dimensional space, with each dimension corresponding to an item of the recommended structure for software requirements specifications (SRS) defined in IEEE Std. 830-1998. We applied various statistical analysis methods to the SRS quality data and project outcomes.

The results showed some interesting relationships between the quality of the requirements and the success or failure of projects; for example, (1) a relatively small set of SRS items had a strong impact on whether a project succeeded or failed; (2) descriptions of SRS in normal projects tended to be balanced; (3) SRS descriptions in Section 1, which were expected to include the purpose, overview and general context for SRS, were comprehensive for normal projects but inadequate for projects that finished with overruns; and (4) when the descriptions of SRS in Section 1 were inadequate, while those of the expected functions and product perspective were comprehensive, the project tended to end up with cost overruns.

Keywords: Requirements quality, project success, statistical analysis.

1   Introduction

The importance of requirements engineering (RE) has been gradually recognized by the software industry, but it can hardly be said that RE research results are widely applied [1]. One of the major obstacles may be the lack of evidence that requirements quality really affects the outcomes of software projects.

A report of a survey conducted by the Standish Group in 1994 [2] is repeatedly cited when arguing about the success or failure of software projects.

Although concerns have been raised about whether the report represents reality and whether it is well supported by scientific facts [3,4], it has had a strong impact because the reported failure rate for software projects is a staggering 80% or more. However, another reason for the wide circulation of the report might be that the failure rate, which was based on the number of projects with cost or time overruns, is intuitively appealing and has thus been readily accepted by the management of software development companies.

The Standish report also provides some data on factors affecting project success and failure based on a survey of IT executive managers, who were asked their opinion on why projects succeeded or failed. "Clear statement of requirements" was the third-ranked (13.0%) project-success factor, and "incomplete requirements" was the first-ranked (13.1%) project-impairment factor.

These rankings suggest there is a relationship between requirements quality and project success/failure, but the data reflect only the subjective perceptions of managers. Moreover, there was no detailed analysis of the quality of requirements.

The aim of our research is to fill the gap in evidence concerning the relationship between requirements quality and project outcomes. To ensure that our evidence would be acceptable to practitioners, we made use of data collected by a company located in Tokyo, Japan.

The company had collected two types of data that covered a sufficient number of projects for our purposes. The first type was requirements quality evaluation data, judged by the company's software quality assurance (SQA) teams. As the SQA process had been established and practiced by this company for quite some time, their results can be assumed to be homogeneous with little fluctuation. The second type of data related to project performance monitored by performance review teams. Different teams are responsible for SQA and performance review. The teams are independent of each other and also of the development teams. Performance is measured by cost and time, with the current levels being compared with the levels estimated at the beginning of the project. Several reviews, separated by a defined interval, are conducted during the project. Both requirements quality data and performance data were available for 32 projects. The data were preprocessed and then subjected to a set of statistical analysis methods.

In the following sections, we describe the characteristics of the target projects and the methods used for data preparation and statistical analysis. We also provide an in-depth analysis of some typical projects and the lessons learned from this study.

The contributions of this research are as follows.

1. The research is based on requirements quality data from real business projects. The quality of the software processes was evaluated in real-time and the data can be considered uniform as they were recorded by well-disciplined review teams following established procedures. In addition, the data were rearranged to fit into the framework of the IEEE Standard for "Recommended Practice for Software Requirements Specifications" [5], so that they could be considered within the framework of an objective set of criteria rather than viewed arbitrarily based on the in-house practices of a specific company.

2. The data on project cost and time overruns were collected in a consistent manner within a business organization. They therefore differ from the type of data used in the Standish report. Moreover, the data were analyzed in relation to requirements quality, so that the results shed new light on requirements engineering.
3. There were interesting lessons learned, including some unexpected findings, as a result of the simple but rigorous statistical analysis and individual project investigations.

2   Characteristics of Target Projects and Available Data

2.1   Target Projects

The projects that were investigated in this research were all carried out by a division of a company in Tokyo. The division is in charge of developing business application systems for customers.

Data for 72 projects conducted from 2003 to 2005 were collected. Of these projects, 32 were completed by the end of 2005 and performance data showing whether the projects had incurred time/cost overruns were available. The other projects were either still in the process of development when we started this study or performance data were not available even though the projects had been completed.

All the projects were for external customers in various industries, and the systems developed were middle to large in size. Unfortunately, detailed information on system size, programming language, type of application, etc. cannot be disclosed due to confidentiality, but none of the projects had particularly uncommon features in regard to those factors.

2.2   Requirements Quality Data

For all the projects investigated, requirements were provided by, or elicited from customers, but the requirements specification documents were written by the development teams and approved by the customers. During the development process, three independent teams are established within the organization: the development team, SQA team, and performance review team. The SQA team is in charge of evaluating the quality of artifacts produced at each phase of the process. The company has carried out the evaluation procedure for a number of years using a fixed check sheet. From the set of artifacts evaluated, we focused on requirements specifications.

The company's requirements specification check sheet lists more than 100 check items. Each item is rated by the SQA team with a score ranging from 0 to 5; 0 means there is no description corresponding to the item, while 1 to 5 indicate the quality grade (the higher the better).

2.3   Project Performance Data

Several reviews are conducted for each project, with the number of reviews depending on the size of the project and the perceived risk. Both the development

and review teams participate in a review, with the results being reported by the review team.

From the items listed in the review reports, we selected the evaluation of performance in relation to cost and time. Each review reports on performance between the last review and the current review. In the final review, the overall performance is evaluated. Evaluation scores range from 0 to 4.

We took the final total performance data on cost and time as a measure of project success/failure. In the final overall review, cost and time are evaluated in comparison with the initial estimates made at the beginning of the project. A grade of 0 or 1 for cost or time identifies that the project has incurred overruns. Performance evaluation data from the reviews carried out during the development process were not used for statistical analysis but are referred to in the individual case studies to be described in Section 4.

Based on the evaluation, projects were assigned to one of four categories:

P1 Normal with no cost and time overruns,
P2 Cost overrun but no time overrun,
P3 Time overrun but no cost overrun,
P4 Cost and time overruns.

We decided to characterize project success and failure using only cost and time factors without considering product quality because the data available on product quality were not as reliable as the data for cost and time. As the 32 projects investigated were all delivered to customers and no serious problems were subsequently reported, their product quality can be considered roughly equal.

The exact number of projects assigned to each of the four categories cannot be disclosed but each class had at least five members. This meant that the distribution of the 32 projects among the four categories was relatively balanced, so that it was reasonable to make comparisons between the categories.

3   Statistical Analysis

3.1   Conversion of Requirements Quality Data

The requirements quality data were of good quality because differences between the evaluators were minimal due to the long-practiced and stable evaluation procedure. Moreover, there were no missing entries. However, we decided not to use the raw data directly but to convert them into another form before analysis. There were two reasons for this decision.

Firstly, the number of check items was too large to manipulate and they needed to be projected into a lower dimensional space. Secondly, the evaluation was conducted according to the in-house practices of a private company. To ensure the analysis was objective, we decided it would be better to convert and interpret the data within a widely accepted standard framework. This should also make our results more useful to researchers carrying out similar studies.

1. Introduction
     1.1 Purpose
     1.2 Scope
     1.3 Definitions, acronyms and abbreviation
     1.4 References
     1.5 Overview
2. Overall description
     2.1 Product perspective
     2.2 Product functions
     2.3 User characteristics
     2.4 Constraints
     2.5 Assumptions and dependencies
     2.6 Apportioning of requirements
3. Specific requirements
     3.1 External interfaces
     3.2 Functions
     3.3 Performance requirements
     3.4 Logical database requirements
     3.5 Design constraints
     3.6 Software system attributes
     3.7 Organizing the specific requirements

Fig. 1. Recommended SRS Structure (IEEE Std. 830-1998)
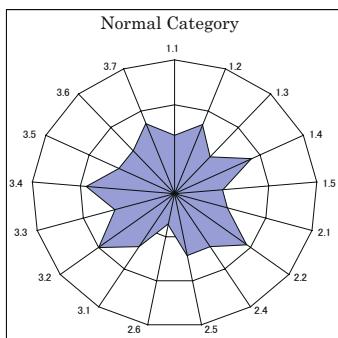
We took IEEE Std. 830-1998 [5] as a reference frame. A recommended structure for software requirements specifications (SRS) given by the standard is shown in Figure 1. The standard provides several alternatives for the contents of Section 3 *Specific requirements* in its appendix, but we adopted the subsection structure stated in the text body. Some subsections, e.g. 2.1, 3.5 and 3.7, have a finer substructure comprising sub-subsections.

We determined the correspondence between the requirements quality check sheet items filled out by the SQA teams and the IEEE Standard SRS entries. The mapping was done as follows.

1. The SRS structure of Figure 1 and its definitions were explained to the members of SQA teams.
2. They discussed and decided on the mapping between the check sheet items and the SRS entries at the lowest level.

Accordingly, each SRS entry at the lowest level is either related to one or more check sheet items or not related at all. In the former case, the score for the SRS entry is defined by the average of the scores for the related check sheet items. The SRS entries in the latter case are simply ignored.

Scores of all entries at the second level shown in Figure 1 are defined directly from the above procedure or obtained by taking the average of the scores for their subentries. As all entries at the second level have at least one subentry
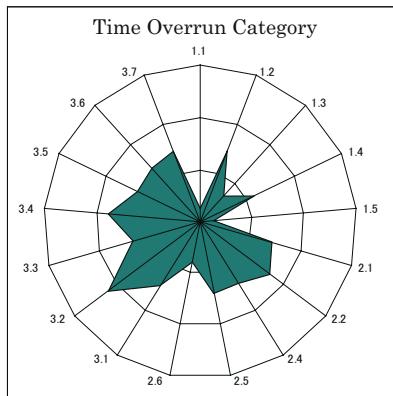
Fig. 2. Spider Chart of Normal Projects

Fig. 3. Spider Chart of Cost Overrun Projects

related to check sheet items, ignoring SRS entries with no corresponding check sheet items causes no problems.
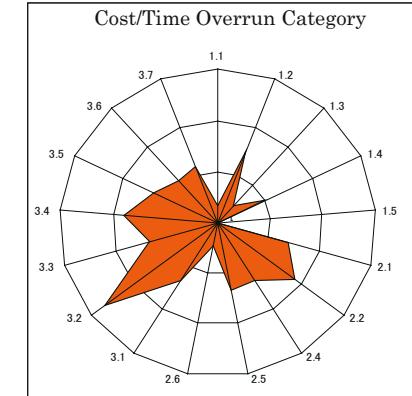
3.2   Requirements Quality Characteristics by Project Category

To provide an overview of the SRS quality data distributed by project category, we drew spider charts of the four categories, as shown in Figures 2 to 5. Each radius corresponds to a SRS item and the score averaged over the projects in the project category is plotted on the radius. As the average scores do not exceed 3 in all cases, the outermost circle corresponds to a score of 3 in all these figures.

When comparing the average scores of the four project categories for each SRS item, the following points are observed.

Fig. 4. Spider Chart of Time Overrun Projects

Fig. 5. Spider Chart of Cost and Time Overrun Projects
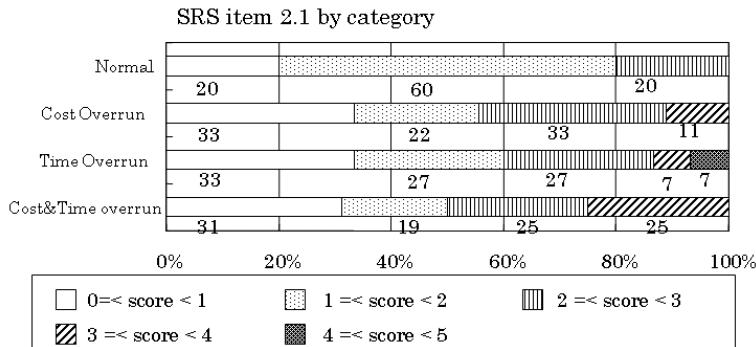
1. SRS items for which a difference in scores of 0.5 or more between at least two project categories was observed are listed below, together with the definition of each SRS item.

   – 1.1 Purpose: delineate the purpose of the SRS and specify the intended audience
   – 1.3 Definitions, acronyms and abbreviations: provide the definitions of all terms, acronyms and abbreviations required to properly interpret the SRS
   – 1.4 References: provide a complete list of all documents referenced in the SRS
   – 1.5 Overview: describe what the rest of the SRS contains and explain how the SRS is organized
   – 2.1 Product perspective: put the product into perspective with other related products and also describe how the software operates inside various constraints such as system interfaces, user interfaces, hardware interfaces, software interfaces, communication interfaces, memory, operations and site adaptation requirements
   – 3.2 Functions: define the fundamental actions that must take place in the software in accepting and processing the inputs and in processing and generating the outputs, including validity checks of the inputs, exact sequence of operations, responses to abnormal situations, effect of parameters, and relationship of outputs to inputs
   – 3.7 Organizing the specific requirements: specific requirements such as system mode, user class, objects, feature, stimulus, responses, and functional hierarchy.

2. The score of P1 was the highest of the four project categories on all SRS items except:

   – 2.1 Product perspective
   – 3.2 Functions
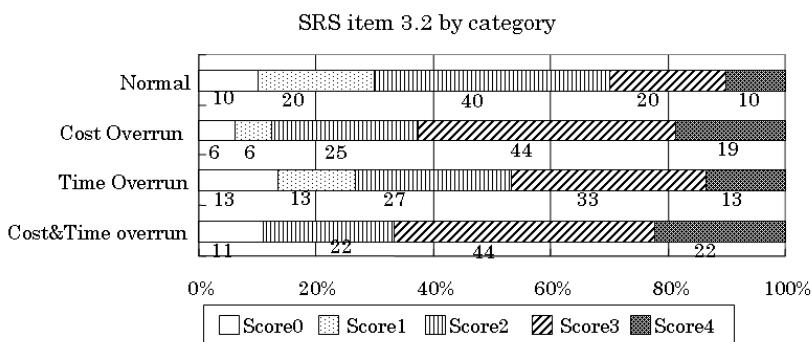
As the visual patterns clearly show, the shape of the normal project category P1 is balanced and almost circular, whereas those of the other project categories are unbalanced. In particular, the inner area of the upper right quadrant corresponding to SRS Section 1 is the largest in the normal project pattern among the four patterns.

The fact that the scores for two SRS items, *Product perspective* and *Functions*, for P1 are not the highest, in fact, they are the lowest of the four, is somewhat counterintuitive and thus interesting. Figures 6 and 7 show the score distribution of these items for the four categories. Compared to other items, these patterns indicate it is not that the scores for normal projects were low, but that the scores for overrun projects were particularly high for these items.

However, these observations are intuitive and not based on analysis of statistical significance. Let us proceed to more rigorous analysis of the relationship between SRS quality factors and project success/failure.

Fig. 6. Score Distribution of SRS 2.1 by Category

Fig. 7. Score Distribution of SRS 3.2 by Category

3.3 Analysis of Variance

We conducted a typical analysis of variance to see how the scores for various SRS items affected project outcomes. Three cases of analysis, each based on dividing the set of projects into two groups, were tested.

1. Normal project group vs. cost- or time-overrun project group
2. Cost-overrun project group vs. cost-within-range project group
3. Time-overrun project group vs. time-within-range project group

For each case, the null hypothesis was that the score distribution of the given SRS item for one group and for the other would be the same. F-tests and t-tests were applied and when the hypothesis was refuted, it implied that the corresponding SRS item could be a factor that affected project performance.

Normal vs. cost or time overrun. Normal projects P1 and cost or time overrun projects, i.e. the union of P2, P3 and P4, are compared. In this analysis, the item 1.1 *Purpose* was found statistically significant by F-test with a 95% confidence range as shown in Table 1.

Table 1. Statistically significant SRS items for normal vs. overrun

No.	Item Name	test	conf.	+/−
1.1	Purpose	F-test	95%	+

conf. = confidence

Table 2. Statistically significant SRS items for cost overrun vs. cost within range

No.	Item Name	test	conf.	+/−
1.5	Overview	F-test	95%	−
2.1	Product perspective	F-test	95%	+
2.6	Apportioning of req.	F-test	95%	−
3.2	Functions	t-test	95%	+

The last column of the table shows whether the factor positively or negatively affected the first group, i.e. the normal project group. In this case, it is positive, which means that normal projects received a better score for the description of SRS *Purpose* than overrun projects.

The hypothesis is not refuted for all the other SRS items either by either the F-test or t-test.

Cost overrun vs. cost within range. Cost-overrun projects are by definition the union P2 and P4 and cost-within range projects are the union of P1 and P3. Four items were found significant between these two groups as shown in Table 2. Both F-tests and t-tests were applied to all SRS items, but only 3.2 *Functions* was found to be statistically significant by the t-test. The other three items were found to be significant by the F-test.

SRS item 2.6 *Apportioning of requirements* is defined in the IEEE Standard as "identify requirements that may be delayed until future versions of the system." Understandably, this arrangement will reduce the risk of a cost overrun.

A particularly interesting result is that statistical significance was found for cost-overrun projects that received higher scores on *Product perspective* and *Functions*.

Time overrun vs. time within range. Time-overrun projects are by definition the union P3 and P4 and time-within- range projects are the union of P1 and P2. Two items were found significant as shown in Table 3.

The interesting point here is that *Purpose* is a strong negative factor to characterize the time-overrun projects. The confidence range for this item is 99%.

Table 3. Statistically significant SRS items for cost overrun vs. cost within estimate

No.	Item Name	test	conf.	+/−
1.1	Purpose	F-test	99%	−
1.5	Overview	F-test	95%	−

3.4   Multivariate Analysis

The analysis described so far is single variate. It is highly plausible that the factors represented by SRS items are mutually related. Thus, some type of multivariate analysis should be applied as well.

As a multivariate analysis method, we chose a tree model because the method is intuitively comprehensible and is suitable for interpreting data that are inherently discrete rather than obtained by measuring physical phenomena.

A decision tree is suitable for representing classification-type knowledge. A variable is assigned to each node and the outgoing edges from that node correspond to possible value ranges the variable can take, so that the selection of an edge at each node, starting from the root node, guides the classification process. The selection path ends at a leaf that determines the group the object is to be classified in [6].
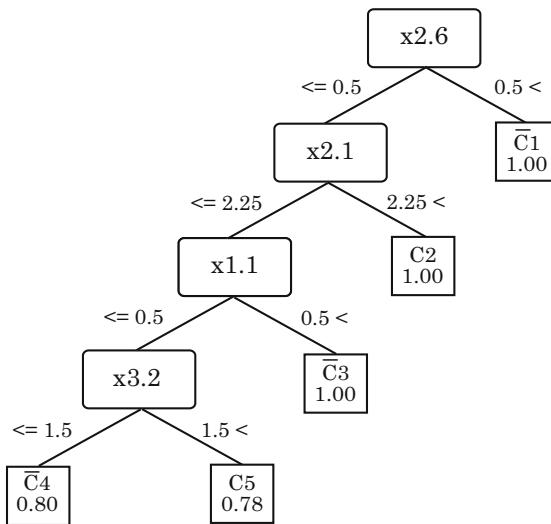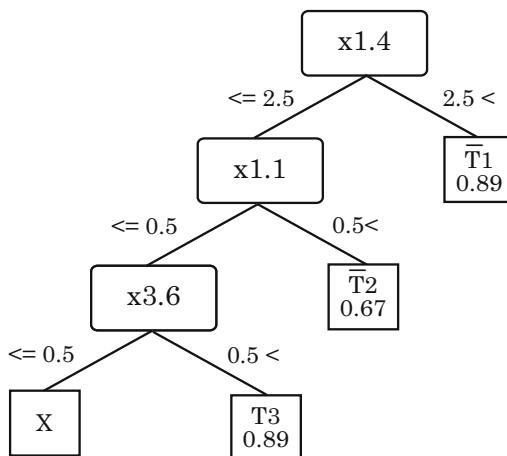
In this case, we use a binary tree, where two edges from a node correspond to two intervals of the value range separated by a threshold value. The threshold is determined statistically to maximize the deviance. To bifurcate a node, a variable that brings the greatest deviance is chosen. When a node sufficiently represents either of the groups, then the node is bifurcated further.

We constructed three tree models, each classifying two groups just as the analysis of variance: i.e. normal vs. cost/time overrun, cost overrun vs. cost within range and time overrun vs. time within range. We used a statistical analysis tool R [7] for generating tree models. Because the analysis of the cost overrun vs. cost within range produced the most striking result, we explain that case first.

Cost overrun vs. cost within range. The result is as shown in Figure 8. A node with label "xn.m" denotes a decision by the score of the SRS item "n.m". The left edge of each node leads to a subset with a score smaller than or equal to the threshold, while the right edge indicates scores greater than the threshold. A leaf node with label "C" represents a subset sorted as a cost overrun, and label "$\bar{C}$" represents a subset sorted as no cost overrun. The number in the box denotes the fraction (1.0 means 100%) of the projects in the sample of 32 classified into this leaf that are actually cost overrun (or no cost overrun for $\bar{C}$ leaf).
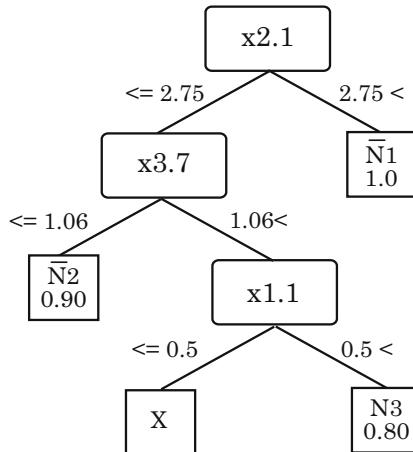
For example, projects with SRS 2.6 score greater than 0.5 are classified as no cost overrun by 100%. We can summarize the results as follows.

1. cost-overrun projects
   - SRS 2.6 score no greater than 0.5 and SRS 2.1 score greater than 2.25 (100%)
   - SRS 2.6 score no greater than 0.5 and SRS 2.1 score no greater than 2.25 and SRS 1.1 score no greater than 0.5 and SRS 3.2 score greater than 1.5 (78%)
2. cost-within-range projects
   - SRS 2.6 greater than 0.5 (100%)
   - SRS 2.6 score no greater than 0.5 and SRS 2.1 score no greater than 2.25 and SRS 1.1 score greater than 0.5 (100%)

Fig. 8. Decision Tree of Cost Overrun Projects

Fig. 9. Decision tree of time-overrun-projects

- SRS 2.6 score no greater than 0.5 and SRS 2.1 score no greater than 2.25 and SRS 1.1 score no greater than 0.5 and SRS 3.2 score no greater than 1.5 (80%)

The negative effect of SRS 2.6 *Apportioning of requirements* to cost overrun can be interpreted straightforwardly as mentioned before but it is still surprising that the item is selected as the first factor and the score just exceeding the value of 0.5 identifies cost-within-range projects out of the sample by 100%.

x2.1

<= 2.75   2.75 <

x3.7 $\overline{N1}$ 1.0

<= 1.06   1.06<

$\overline{N2}$ 0.90   x1.1

<= 0.5   0.5 <

X   N3 0.80

Fig. 10. Decision Tree of Normal Projects

The last case of cost within range is particularly interesting. In this case, all the factors 2.6, 2.1, 1.1, and 3.2, are on the negative side. We will come back to the interpretation of this rather counterintuitive phenomenon later.

Time overrun vs. time within range. Figure 9 shows the tree model that classifies time- overrun projects.In this case, SRS 1.4 *Reference* was chosen first as giving a negative effect, i.e. if references are well described, the project is unlikely to incur time overruns. The next factor is SRS 1.1 *Purpose*, which is also a negative effect. Under the condition that these two scores are low, SRS 3.6 *Software attributes* has a positive effect, i.e. if the item has a good score, the project tends to be delayed.
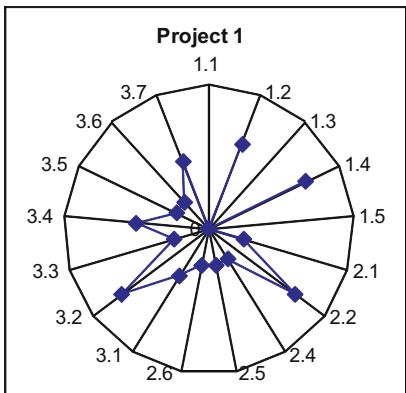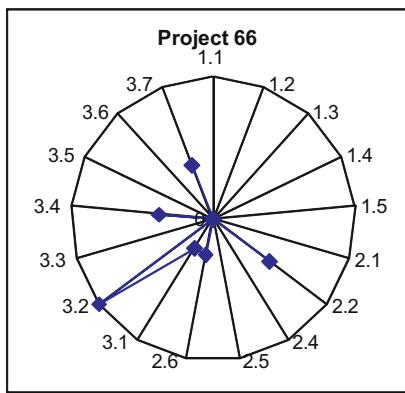
The last leaf labeled by "X" in the tree denotes a case where the decision is inconclusive, i.e. it is hard to decide to which group members in this leaf belong.

Normal vs. cost or time overrun. The decision tree for normal projects is shown in Figure 10.

It is surprising that SRS 2.1 *Product perspective* is selected first and affects negatively. It will be discussed in the next section. Otherwise, the condition that high rating of both SRS 3.7 *Organizing the specific requirements* and SRS 1.1 *Purpose* is required is understandable.

4   Closer Look at Individual Projects

Using multiple statistical analysis methods, we found that there were relationships between SRS quality and project success/failure. The general trend supports the widely accepted assumption that higher requirements quality favors a successful project, but some apparently opposing phenomena were also found.

Fig. 11. Spider Chart of Project #1   **Fig. 12.** Spider Chart of Project #66

To see how requirements factors actually affect software processes and outcomes, we investigated each of the 32 sample projects in detail. Here, we select some typical cases and focus on the possible causes of the *counterintuitive* results. In analyzing the projects individually, we used not only the quantitative data treated in the preceding sections but also the qualitative data available in the form of comments made by the SQA and review teams.
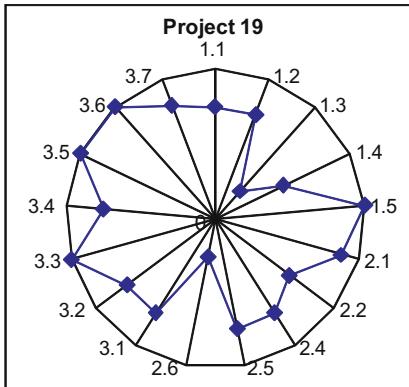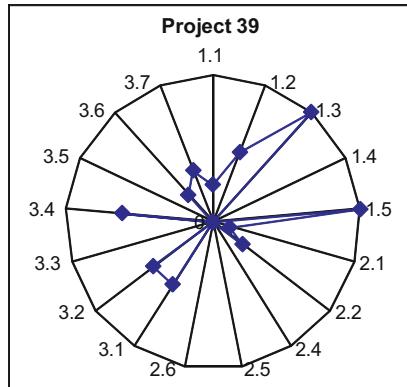
First, we selected project #1[1], which is a cost-overrun project. Figure 11 shows a spider chart of the SRS quality of this project. The outermost circle of all the spider charts in this section corresponds to a score of 4, in contrast with those in Section 3 where the outermost circle corresponds to a score of 3.

In Section 1 of SSR, 1.1 *Purpose*, 1.3 *Definitions, acronyms, and abbreviations*, and 1.5 *Overview* are completely missing, while 1.2 *Scope* and 1.4 *References* are well written. In Section 2 and 3, 3.2 *Functions*, 2.2 *Project functions* (summary of major functions) and 3.7 *Organizing the specific requirements* received high scores.

In fact, the requirements analysis task was not conducted independently in this project but was merged into the design phase, and the SRS and design specifications were written at the same time. That explains why the descriptions of detailed functions in the SRS are comprehensive compared to other items. But the testing phase found that several requirements were missing and there were also errors which required extensive reworking and pushed up the cost.

Another cost-overrun project, #66 went to even further extremes, as Figure 12 shows. Description in Section 1 is nil and the same pattern can be seen for 3.2, 2.2, and 3.7. The requirements definition task of this project was not completed during the requirements phase and the task was moved to the design phase. This situation is quite similar to #1. Detailed functional descriptions without a firm grasp of conceptual and essential requirements tend to require considerable changes in specifications and reworking.

---
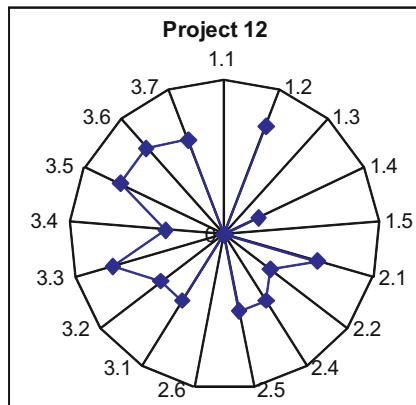
[1] The 32 projects under study were taken from a set of 72 projects and thus the projects are numbered from 1 to 72.

Fig. 13. Spider Chart of Project #19   **Fig. 14.** Spider Chart of Project #39

However, there is a case in which a cost-overrun project produced a balanced and beautifully shaped SRS evaluation spider chart, as shown in Figure 13. This project, #19, did well in the RE phase and no problems were detected during the design and implementation phase. But it turned out that the amount of work required in the testing phase was higher than projected and extra testing staff had to be added. According to the postmortem, the cause was simply a wrong estimate of the testing load.

The next case involves a time-overrun projects.

In general, time-overrun projects show a similar pattern to cost-overrun projects such as #1, i.e. relatively poor scores in SRS Section 1 items. The spider chart shown in Figure 14 for project #39 shows a different shape. It has particularly high scores in SRS Section 1. After a closer look at the documents and thorough interviews, we found that the cause of the project delays was not poor requirements specification, but the use of an inappropriate tool in the

Fig. 15. Spider Chart of Project #12

implementation phase. The functions of the tool were inadequate for the purpose and the development team had little experience in using it.

Our last example is a cost-and-time-overrun project, #12.

In this case, the pattern for Section 1 is also similar to that for #1. The scores for Section 2 and 3 are higher. In particular, the high score for 2.1 is worthy of note. This project failed to complete a study of the current system during the requirements analysis phase, which caused frequent specification changes in the design phase, resulting in cost and time overruns. It also implies that the interface to the current system was complicated and as a result, SRS 2.1 *Product perspective*, which is supposed to describe interfaces to other systems among others, turned out to be relatively highly rated. This case partly addresses the question raised in the last section.

5   Threats to Validity

The first probable threat to validity could be the reliability of the SRS quality evaluation data. Admittedly, there is a risk of dispersion between evaluators and room for human error. However, compared to other similar work, we believe that the data set used in this study has several advantages for the following reasons.

1. Evaluations were done and recorded in real time as projects proceeded. The data are therefore fundamentally different from those collected through questionnaires or interviews after projects have been completed.
2. The evaluation procedure is well established, it has been practiced for a long time, and a fixed check sheet is used. The data are therefore different from those obtained by asking survey responders or interviewees for their subjective views.
3. The data were systematically collected through the daily practices of dedicated SQA teams. They were not especially collected by researchers for the sake of research.

The second probable threat to validity could be the limited data source. Data were collected from a single organization and the characteristics of the projects are basically uniform. This is an advantage in the sense that the data are coherent and well suited to statistical analysis. However, comparison with other kinds of data covering different applications and organization types will be fruitful.

6   Related Work

One of the most important studies relating to our work is the one reported by Damian & Chisan [8]. They conducted an intensive study of a large software development project that had just introduced a requirements process improvement program. The case study lasted 30 months to follow the RE process improvement activities, while the research process was divided into three stages. This approach can be characterized as follows: (1) a single large project was targeted;

(2) questionnaires prepared by the researchers were used to collect data from managers, team-leaders, and senior engineers; and (3) the focus was on RE processes rather than on the quality of RE products. In this sense, their work and ours complement each other.

Another example of an in-depth case study on a single company was reported by Wohlwend and Rosenbaum [9]. It is interesting to note that they adopted a single criterion whether the software was delivered on time to judge project success.

Sommerville & Ransom [10] also focused on RE process assessment and improvement. Nine companies were contacted in this study and the RE process maturity model proposed by the authors was used to assess their processes. Process improvements were then recommended and practiced. The eventual goal of the research was to correlate improvements in RE processes to business performance, which is challenging and hard to achieve, as the authors admit.

The goal of the work by Verner et al. [11] is closer to ours. They tried to find relationships between requirements practices and software project outcomes. The approach they took was to distribute questionnaires to practitioners in the U.S. and Australia. The respondents answered questions that characterized the RE practices of projects they knew, which they considered to have either succeeded or failed. The authors admit that "surveys are of course based on self-reported data which reflects what people say happened, not what they actually did or experienced."

Research on measuring the success of RE processes is reported by Emam & Madhavji [12]. They listed up to 32 measurement indicators and classified them into two major dimensions, quality of RE products and quality of RE service. Their analysis is closed within RE processes and is not related to project success or final product quality.

7   Conclusion and Future Directions

The findings of our study can be summarized as follows.

1. Data indicate there is a relationship between SRS quality and project outcomes. Moreover, a relatively small set of SRS items have a strong impact.
2. Descriptions of SRS in normal projects tend to be balanced. When the SRS item evaluation rating is plotted on a spider chart, the pattern shows a figure approximating a circle.
3. SRS descriptions in Section 1, which covers purpose, overview and the general context of the SRS, are comprehensive in normal projects and sparse in overrun projects. In particular, when the references or purpose in Section 1 are well written, the project tends to finish on time.
4. When the descriptions in SRS Section 1 are sparse, while those of functions and product perspective are comprehensive, the project tends to result in cost overruns because such characteristics often indicate that the RE phase has been neglected, or absorbed into the design phase.

5. Identifying requirements whose implementation may be delayed can be a good way of preventing cost overruns.

In the near future, we plan to collect more data from the same organization to corroborate or enrich our current findings. It will also be interesting to try other types of multivariate analysis, such as principal components analysis. In the long term, it will be valuable to apply a similar approach to other areas, including embedded software systems and COTS products.

The empirical approach of analyzing and mining software engineering data is now widely pursued and bringing fruitful results. There are at least three factors that are enabling and encouraging this approach. Firstly, data analysis techniques have made much progress. Secondly, enhancement of hardware performance has made brute force approaches, which were infeable before, viable. Thirdly, and quite importantly, the proliferation of open source projects is providing a large amount of software engineering data.

However, in the area of requirements engineering, it is still not easy to obtain requirements related data that are significant in amount and quality. It is expected that much efforts will be expended to collect real data concerning requirements and new light will be shed on RE knowledge obtained through analyis of such data.

Acknowledgment

References

1. Berry, D., Damian, D., Finkelstein, A., Gause, D., Hall, R., Simmons, E., Wassyng, A.: To do or not to do: If the requirements engineering payoff is so good, why aren't more companies doing it? In: Proc. 13th International Requirements Engineering Conference (RE 2005), p. 447. IEEE, Los Alamitos (2005)
2. Standish Group International: The chaos report (1994), http://www.standishgroup.com/sample_research/chaos_1994_1.php
3. Glass, R.L.: The standish report: Does it really describe a software crisis? Communications of the ACM 49, 15–16 (2006)
4. Jorgensen, M., Molokken-Ostvold, K.: How large are software cost overruns? A review of the 1994 chaos report. Information and Software Technology 48, 297–301 (1997)
5. IEEE: Recommended practice for software requirements specifications. Technical report, IEEE, IEEE Std 830-1998 (1998)
6. Breiman, L., Friedman, J.H., Ohshen, R.A., Stone, C.J.: Classification and Regression Trees. Wadsworth, Belmont (1984)
7. The R Foundation for Statistical Computing: The R project for statistical computing, http://www.r-project.org/index.html

8. Damian, D., Chisan, J.: An emipirical study of the complex relationships between requirements engineering processes and other processes that lead to payoffs in productivity, quality, and risk management. IEEE Transactions on Software Engineering 32, 433–453 (2006)
9. Wohlwend, H., Rosenbaum, S.: Software improvements in an international company. In: 15th International Conference on Software Engineerig (ICSE 1993), Baltimore, MD, USA (1993)
10. Sommerville, I., Ransom, J.: An empirical study of industrial requirements engineering process assessment and improvement. ACM Transactions on Software Engineering and Methodology 14, 85–117 (2005)
11. Verner, J., Cox, K., Bleistein, S., Cerpa, N.: Requirements engineering and software project success: An industrial survey in Australia and the U.S. Australian Journal of Information Systems 13, 225–238 (2005)
12. Emam, K.E., Madhavji, N.H.: Measuring the success of requirements engineering processes. In: Second IEEE International Symposium on Requirements Engineering, pp. 204–211 (1995)

Designing Value-Based Inter-organizational Controls Using Patterns

Vera Kartseva[2], Jaap Gordijn[1], and Yao-Hua Tan[2]

[1] VUA, Faculty of Sciences, De Boelelaan 1081, 1081 HV Amsterdam, The Netherlands
gordijn@cs.vu.nl
[2] VUA, Faculty of Economics, De Boelelaan 1105, 1081 HV Amsterdam, The Netherlands
{kartseva,ytan}@feweb.vu.nl

Abstract. Networked organizations, consisting of enterprises who exchange things of economic value with each other, often have participants who commit a fraud or perform other actions, not agreed in a contract. To explore such opportunistic behavior, *and* to design solutions to mitigate it, we propose the $e^3 control$ approach. This approach takes the valuable objects, which are exchanged between enterprises, as a point of departure, and proposes a control patterns library to find solutions for various types of opportunistic behavior in network organizations. The practical use of the patterns is illustrated by a case study in the field of renewable electricity supply in UK.

Keywords: Inter-organizational control, value network, control pattern.

1 Introduction

Organizations increasingly organize themselves as networks: Collections of enterprises that jointly satisfy a complex consumer need, each utilizing their own specific expertise, products, and services [1]. These networks are enabled by innovative technologies such as web-services, allowing for timely coordination of enterprises. Due to this innovation, new networks emerge, for instance in the field of energy supply, Internet service provisioning, or digital content [2,3,4,5].

Techniques, such as goal- and value modeling [6,3] play an important role in the early requirements engineering phase for information systems supporting and enabling these networks. For instance, in [3] we report how to explore an IT-enabled network of enterprises from a *business value* perspective using the $e^3 value$ technique, and in [4] we explain how such exploration can be done in combination with multi-actor *i** goal analysis. In brief, $e^3 value$ analyzes *what* objects of *economic value* are exchanged between enterprises, and *what* actors request in return for these objects (usually other objects of value). So, the $e^3 value$ approach is an *early* requirements engineering technique with the aim to understand business *value* requirements in a model-based way. Understanding of the network's business value requirements provides a starting point for analyzing requirements of information systems.

The $e^3 value$ approach deliberately supposes that enterprises behave *honest*, as otherwise resulting models would soon become rather complex, and disturb executive decision making. In $e^3 value$, 'honest behavior' refers to actors who - if they obtain an

object of value from their environment - *always* provide another object of value to their environment in return, as a economic reciprocal exchange. To our experience, it is initially already sufficiently difficult to design a network under such perfect-world conditions. The next step however is to assume *opportunistic* behavior of enterprises. To this end, we have proposed $e^3control$ [7]. As $i*$ and e^3value , the $e^3control$ technique is a requirements engineering technique to understand the context of multi-actor information systems. The $e^3control$ models have close similarities with e^3value models with one important difference: An e^3value model supposes that each enterprise in a network behaves honestly, or *ideally*, whereas an $e^3control$ model allows opportunistic, or *sub ideal*, behavior. An $e^3control$ model however still focuses on the *value objects* exchanged.

To discourage opportunistic behavior, *control mechanisms* can be applied. Such controls are often *value-based*, e.g. penalties and incentives, or reconciliation of valuables. Also, controls may require specific business processes, or rely on information technology (e.g. security protocols). As processes are significantly controlled and executed by IT, understanding of these controls are important for the IS requirements.

To design controls in networked enterprises, $e^3control$ can be used as a general framework, but the design process still requires a vast amount of knowledge on organizational controls *themselves*. To make this knowledge available within $e^3control$, we propose a library of *control patterns*, which describes organizational controls for networked organizations. These patterns are the main contribution of this book chapter. The $e^3control$ approach and the supporting patterns are unique because they are grounded in an *economic value* perspective, while connecting properly to the processes putting the controls into operation. It is the transfer of valuable objects in a network that has to be controlled in first place. This contrasts to existing process-only approaches for controls (see e.g. [8]), or even EDP-auditing (e.g. [9]).

The controls have been collected from agency theory (e.g. [10]), internal control theory (e.g. [8]) and management control theory (e.g. [11]). Examples of such organizational controls are *detective* controls such as monitoring and verification (e.g. quality control, reconciliation of accounting records with material reality), but also *preventative* controls, such as economic incentives and penalties. In addition to literature, the patterns are based on four real-life case studies we performed in the drinks industry [12], international trade [13], the entertainment industry [14], and electricity supply industry [15]. In this chapter, we elaborate on the latter case study.

This chapter first introduces the notion of value-based controls for networked value constellations (Sec. 2). Then, we present three of our control patterns in detail in Sec. 3 as well a summary of the rest of the patterns. In Sec. 4, we show the three patterns can be practically applied in a case study. Finally, Sec. 5 presents our conclusions.
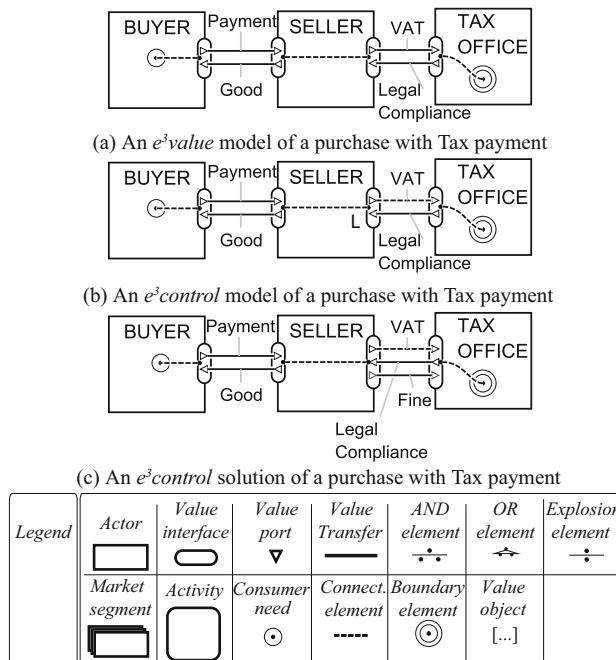
2   Value-Based Design of Controls for Networked Constellations

We illustrate the design of controls for networked constellations by a small example (see also Fig. 1). For this example, we suppose that someone buys a product or a service from a seller, and the seller has to pay Value Added Taxes (VAT) to the Tax office. In $e^3control$, we follow three subsequent steps to analyze networks for sub ideality.

Step 1: Elicit and Model the *Ideal* Network Using an *e³value* Model. We use the *e³value* technique to first understand the network, assuming that all actors would be behave ideally, or, in other words, would be honest. The *e³value* technique allows to represent which enterprises in a network exchange which objects of economic value with which other enterprises. Fig. 1 exemplifies a buyer obtaining goods from a seller and offering a payment in return. Due to the law, the seller must pay a value-added tax (VAT). This can be conceptualized with the following *e³value* constructs.

Actors, such as the buyer, seller, and the tax office are economically independent entities. Actors transfer *value objects* (payment, goods, VAT) by means of *value transfers*. For value objects, some actor should be willing to pay, which is shown by a *value interface*. A value interface models the *principle of economic reciprocity*: only if you pay, you can obtain the goods and vice versa. A value interface consists of *value ports*, which represent that value objects are offered to and requested from the actor's environment. Actors may have a *consumer need*, which, following a path of dependencies will result in the exchange of value objects. Transfers may be *dependent* on other transfers, or lead to a *boundary element*. In the latter case, no transfers are considered anymore.

The important point here is that an *e³value* model *by definition* supposes that all actors behave ideally. This is reflected by the explicit notion of 'economic reciprocity': All agreed transfers are required to happen, or should not happen at all. Performing this step results in understanding of the valuable objects that should be transferred, and thus which objects should be subject to control.

(a) An *e³value* model of a purchase with Tax payment

(b) An *e³control* model of a purchase with Tax payment

(c) An *e³control* solution of a purchase with Tax payment

Fig. 1. Example of an *e³value* model of a purchase with tax payment

Step 2: Analyze *Sub ideality* in a Network Using an e^3control Model. In reality, actors ideal often behave sub-ideally: they commit a fraud or make unintentional errors. In e^3control , these situations are modeled by *sub-ideal value transfers* [7]. These are graphically represented by dashed arrows, and can indicate different risks: e.g. actors not paying for goods, not obtaining the goods, or obtaining wrong goods. For example, Fig. 1 (b) models a situation that the seller does not pay VAT. 'L' is a *liability token* [7], assigned to the responsible actor for the sub-ideal value transfer, here the seller.

Step 3: Reduce *Sub ideality* by Adding Controls. We now add control mechanisms that *reduce* the control problem. Hardly any control mechanism can *remove* a control problem completely. A combination of mechanisms, so-called 'control mix', is usually required [8]. For example, Fig. 1 (c) introduces fining. In case the seller does not pay taxes, he is charged with a high fine. The fine is modeled as a value object, transferred from the seller to the tax office. As can be seen by the dashed transfer, the model is still sub-ideal, but at least the Tax Office receives adequate compensation if the seller behaves sub-ideally, and if such behavior is detected.

3   Control Patterns

The design process in Sec. 2 is general and requires quite some design knowledge on well accepted control problems and solutions. To increase the usability of e^3control , it is therefore important to bring in this *accepted* knowledge; therefore we propose a series of *inter-organizational* control patterns (cf. [16]). These patterns and their use is the main contribution of this chapter.

3.1   Elicitation and Representation of Control Patterns

Elicitation Method. Pattern development usually consists of the *identification*, *collection* and *codification* of existing knowledge [17]. The PattCaR method developed by [18], suggests more specific guidelines for patterns elicitation: (1) analysis of the *domain* and *context* of the patterns, (2) definition of a *vocabulary*, (3) a thorough domain analysis and extraction of *patterns candidates*, (4) a collection of several examples of each *pattern candidate*, (5) *encoding* of patterns by modeling the examples and performing a commonality-variability analysis, and (6) a description of *relations* between patterns.

Domain of Controls for Networks. There are several theories that attempt to describe the domain of controls, including accounting control theory (e.g. [19]), management control theory (e.g. [11]), and agency theory (e.g. [10]). There is also specific work on inter-organizational controls, such as [20,21].

In this chapter, we consider controls in terms of the principal-agent framework. This framework makes a distinction between a *primary* actor (or principal) and a *counter* actor (or agent). The counter actor behaves *sub-ideally* and the primary actor wants to reduce the loss caused by such behavior. The agency theory describes several control problems and mechanisms to mitigate sub ideal behavior, namely *Screening*, *Signaling*, *Monitoring*, and *Incentives*. The Screening and Signaling mechanisms are used to

counter the *hidden information* problem. This occurs if the primary actor does not have enough knowledge about the counter actor, increasing the risk that the counter actor will perform his activities in a sub ideal way. The screening and signaling mechanisms recommend checking the counter actor's abilities and characteristics before signing a contract with him. The Monitoring control is used to counter the *hidden action* problem, which means that the counter actor performs his activities in a sub ideal way. The Monitoring mechanism recommends verifying the counter actor's performance before rewarding him. If monitoring is difficult or costly, the counter actor can be stimulated to behave ideally by Incentives which can be positive (reward) or negative (punishment).

Two additional inter-organizational controls are described in [20]. The Commitment Evidence control applies to a situation in which the counter actor inappropriately denies his commitment to the primary actor. The Execution Evidence control addresses a counter actor inappropriately claiming that the primary actor executed his activities sub ideally. Both commitment evidence and execution evidence controls require the creation of evidence that can be used in (legal) disputes against the counter actor. These two controls stem up from the audit trail principle of the internal control theory.

Usually, a distinction is made between *ex-ante* controls, i.e. controls executed before the contract between two actors is settled, and *ex-post* controls, i.e. controls executed after the contract is settled. The Screening, Signaling and Settlement of incentives are ex-ante controls, while Monitoring, Commitment evidence, Execution evidence and Execution of incentives (actual rewarding or punishment) are ex-post controls. A further distinction can be made between *contractual* controls and *procedural* controls. Contractual controls employ value-based mechanisms to stimulate the counter actor to behave ideally. Procedural controls employ process-level mechanisms to repressively prevent or detect the counter actor's sub ideal behavior. With the exception of incentives, all the groups of controls considered here are procedural.

Pattern Representation and Vocabulary. Cf. [22], a pattern has the following structure: *name*, *context*, *problem*, *solutions*. We consider a *control pattern* as a description of generic and re-usable control mechanism for a recurring control problem. We describe the context, problem, and solution slots by taking a business value (e^3value or $e^3control$) and business process (UML activity diagrams [23]) perspective, thereby following the principles of multi viewpoint requirements engineering [24]. Examples of patterns can be found in the appendix of this chapter.

We use the following vocabulary[1] to describe a control pattern. There are two actors, a *primary actor* and a *counter actor*. From a value perspective, the primary and counter actors exchange value objects: the primary actor transfers a *primary value object* (PO) to the counter actor, and the counter actor transfers a *counter value object* (CO) in return. From a process perspective, the exchange of PO corresponds to execution of a *primary activity*, and the exchange of CO corresponds to execution of the *counter activity*. These activities can also be collections of multiple operating activities.

Sub ideal behavior and, consequently, sub ideal transfers are defined from the point of view of the primary actor, who is the *principal*. Sub ideal behavior is executed by the counter actor, who is the *agent*. The primary actor expects the counter actor to

---

[1] The terminology is inspired by [25].

behave sub ideally with respect to the execution of the counter activity. The result of this opportunistic behavior is a sub ideal transfer of the CO. Obviously, actors can all play the role of principal or agent, depending on the perspective taken.

Furthermore, based on [25,26,8], we have also developed a vocabulary of *control activities* and *control principles*, which form the building blocks for control mechanisms. The activities include e.g. *verify*, *witness*, *testify*, and *authorize*. The control principles are normative rules of relations between activities, objects and actors [26]. As an example of such a rule, segregation of duties requires the party who executes a verification activity to be independent and socially detached from the party who executes the activity being verified. Also, the ordering of activities is motivated by control principles.

Extraction of Pattern Candidates. Due to lack of space, we can not present the pattern extraction process itself (see [27] for more details), rather we focus on a few considerations with respect to this process, and the results.

A first consideration is that our candidate patterns should represent a unique combination of problems and solutions. To do so we require that (1) two different control problems should fall in two different patterns, (2) two different control mechanisms for one control problem should be represented by two different patterns.

A second consideration is that, in line with the approach of e^3value [3], we want the patterns library to be lightweight. This means that the fewer patterns we have to describe all the considered controls, the better it is. To achieve this, we abstract from domain-specific details, which are present in the internal control theory. Firstly, we do not consider any specific roles of actors, such as a supplier or a customer. We describe a transaction in terms of the principal-agent framework. So, we distinguish between a *primary* and a *counter actor*. The counter actor behaves *sub ideally* and the primary actor wants to reduce the loss caused by this sub ideal behavior. The actors can delegate their activities to *trusted parties*. Secondly, we do not differentiate controls if they only involve different types of documents, e.g. a purchase order or a contract.

Our domain analysis resulted in Screening, Signaling, Monitoring, Commitment Evidence, Execution Evidence and Incentives controls. We call these *sub domains* of the control domain, and we consider these to be a good starting point for elicitation of the patterns. We compare the sub domains with each other, and re-group them to select the unique problem-solution pairs. These pairs will form the control patterns. Effectively, this process is about commonality-variability analysis of the domains.

Screening and signaling. The screening and signaling sub domains mitigate the same control problem of hidden characteristics, however they are different control mechanisms in terms of solution. In screening, the primary actor verifies the *activities* of the counter actor. In signaling, the primary actor verifies indirect *signals* and not the activities. Such signals have a historical correlation with the expected performance of the actor in the future. The difference between screening and signaling is that screening is based on information collected by direct observation of the counter actor's activity, while signaling is based on information collected from a third party. However, if we ignore delegation, the difference between the two mechanisms disappears. So, screening and signaling describe the same control problem and the same control mechanism. This results in one pattern, called *Partner Screening*, in which the primary actor screens his partner.

Screening and monitoring. Screening and monitoring mitigate the same control problem: The sub ideal execution of contractual agreements by a counter actor. On the other hand, the screening control also considers the condition of hidden characteristics, and, therefore, employs an ex-ante control. As explained before, screening verifies activities performed by a counter actor in the past with the assumption that the past resembles the future. The monitoring mechanism does not carry the assumption of hidden characteristics. As a result, it is an ex-post control performed in the context of an existing contract and it suggests verification of activities under the contract. Because of this difference, screening and monitoring controls result in different patterns. The pattern for monitoring control is called *Execution Monitoring*, meaning that the execution of the counter actor's activities is monitored by the counter actor.

Positive incentives and negative incentives. Incentives may be positive or negative. Positive incentives stimulate the counter actor to behave ideally by rewarding him while the negative incentives do the same by punishing the counter actor. These two mechanisms require different changes in e^3value models. Namely, positive incentives can be created by adding an incoming value object to the counter actor in the case of ideal behavior, while negative incentives can be created by adding an outgoing value object to the counter actor in the case of sub ideal behavior. We therefore put positive and negative incentives into two different patterns. Positive incentives are described in the *Incentive* pattern and negative incentives are described in the *Penalty* pattern.

Monitoring and incentives. Both the monitoring and the incentive mechanisms mitigate the same control problem - that of a sub deal execution of contractual agreements by the counter actor. On the other hand, the two controls are different, as the former is a procedural control, while the latter is a contractual control. Incentives also require monitoring mechanisms to prove when the reward or punishment has to be issued or not. Such proof can be modeled with the pattern *Execution Monitoring*, while the actions related to the punishment are a part of the Penalty pattern. In fact, the incentive mechanism is a *variation* of the monitoring mechanisms.

Execution evidence and commitment evidence. The execution and commitment evidence controls involve the same activity: The counter actor should provide the primary actor with an evidence document, which can later be used in a legal dispute. As a commitment evidence control, the evidence document contains a testimony of the counter actor's commitment to a future transaction with the primary actor. This evidence document is normally represented by a contract. For the execution evidence control, the evidence document contains the counter actor's testimony that the primary actor executed his obligations as stated in the contractual agreement. An example of such an evidence document is a receipt given as proof of payment.

So, the processes behind these mechanisms are technically the same, only the role of the evidence document is different. This is because the two controls address different control problems. In addition, the commitment evidence control is executed ex-ante, while the execution evidence control is executed ex-post. For these reasons, we describe these two controls in different patterns.

The pattern for the commitment evidence control is described in the *Proper Contracting* pattern. As the name implies, the control provides guidelines on a correct

contracting process. The pattern for the execution evidence control is called *Execution Confirmation*. The name reflects the essence of the mechanism, which is to provide evidence about the execution of primary activities.

Delegation. The agency theory and the work of Bons consider controls in a relationship between two actors: A primary and a counter actor. In addition, Bons also considers some network aspects. He considers networks as being derived from a two-actor network as a result of the *delegation* of activities to other actors. We do not include delegation issues in our patterns (e.g. as variants), but rather factor it out. First of all, a very large number of different networks (and so patterns) can be formed by using delegation, as e.g. third parties can further delegate activities. If we also consider that actors not only execute primary and counter activities, but also other activities associated with controls (e.g. reconciling, witnessing, verifying), even more possibilities for delegation arise. Furthermore, inclusion of delegation in our patterns would describe similar control problems and mechanisms and only differ in the way activities are delegated, which, strictly speaking, is not a control issue.

Therefore, we describe each pattern only for a transaction between a primary and a counter actor. In order to describe delegation situations, we introduce *delegation patterns* (see [27] and http://www.e3value.com/e3family/e3control/patterns), which provide guidelines on how the two-actor model of a control pattern should be properly changed into the multi-actor model. They ensure that when an activity is delegated, the controls prescribed by the control pattern, are still in place.

Examples of Patterns. In addition to the literature review, elicitation and validation of usability of the patterns was done through a series of case studies.

- *Beer Living Lab.* The case study is about an excise collection procedure inside and outside the EU. This case study [12] contains the patterns Execution Monitoring, Partner Screening Certification, and includes multiple situations when activities by a principal or an agent are delegated to other trusted parties.
- *Dutch health care services.* The case study is about processes in Dutch health care system. It contains the patterns Execution Monitoring and Certification and provides a test of patterns for a non-profit sector [15].
- *International trade.* The case study is about a bill of lading procedure in international trade. It contains the patterns Execution Monitoring, Proper Contracting and Execution Confirmation. It demonstrates the application of patterns in a complex situation when control mechanisms in a network are conflicting [13].
- *Internet Radio.* The case study about an Internet service of free radio broadcasts. It contains, by applying the Execution Monitoring pattern, a mechanism for controlling how many listeners the radio station has, using data collected from distributed listeners [14].

Control Patterns. Based on the mentioned literature and case studies, we have identified the following control patterns: Partner Screening, Proper Contracting, Execution Monitoring, Execution Confirmation, Incentive and Penalty. The patterns are summarized in Table 1; for three of these patterns, the e^3value , $e^3control$, and UML activity models are shown in the Appendix of this chapter.

Table 1. Library of Control Patterns

Name	Control Problem	Solution
Partner Screening	Counter Actor executes his commitment sub ideally.	Primary Actor verifies credentials of Counter Actor before making any commitments
Execution Monitoring	Counter Actor executes his commitment sub ideally.	Primary Actor verifies Counter Actor's execution of the commitment, before executing own commitments
Incentive	Counter Actor executes his commitment sub ideally.	Primary Actor provides a reward for the ideal execution
Penalty	Counter Actor executes his commitment sub ideally.	Primary Actor provides a punishment for the sub ideal execution
Proper Contracting	Counter Actor denies to have made a commitment to Primary Actor	Counter Actor provides an evidence document, which confirms his commitment
Execution Confirmation	Counter Actor denies that Primary Actor executes commitments ideally, and refuses to execute his commitments in return, or requires a compensation for executing his commitments	Counter Actor provides an evidence document, which confirms that Primary Actor executes his commitment ideally

4   Case Study: Renewable Energy in the UK

4.1   Introduction

One of the industries with interesting and complicated control problems is the renewable electricity industry. To comply with international environmental agreements, such as the Kyoto protocol, governments must ensure that a sufficient amount of electricity is produced with technologies that do not use fossil fuels. Examples of CO_2-friendly technologies are wind turbines, photovoltaic panels and hydro generators. Such technologies are called *renewable* or *green* technologies. At present, these technologies require high initial investments, meaning that the price of green electricity is higher than the price of electricity produced in the conventional way using fuel-based technologies [5]. Many government regulated schemes have been implemented to make renewable technologies commercially more attractive, e.g. tax cuts and subsidies on initial investments, premiums for generated electricity, etc. In this chapter we examine more closely one such scheme, which was implemented in the United Kingdom (UK).

In the UK, the Renewable Obligation (RO) regulation law was introduced to stimulate the generation of renewable electricity. The first Renewable Obligation regulation in the UK came into force in April 2002. The law places an obligation on electricity *suppliers*, licensed to supply electricity in the UK, to source a certain proportion of electricity from renewable sources [28]. When the regulation was introduced, this portion constituted 10% of the total supply of a UK supplier. In 2006/07 a UK supplier is obliged to generate 6.7% of its supply from renewable sources.
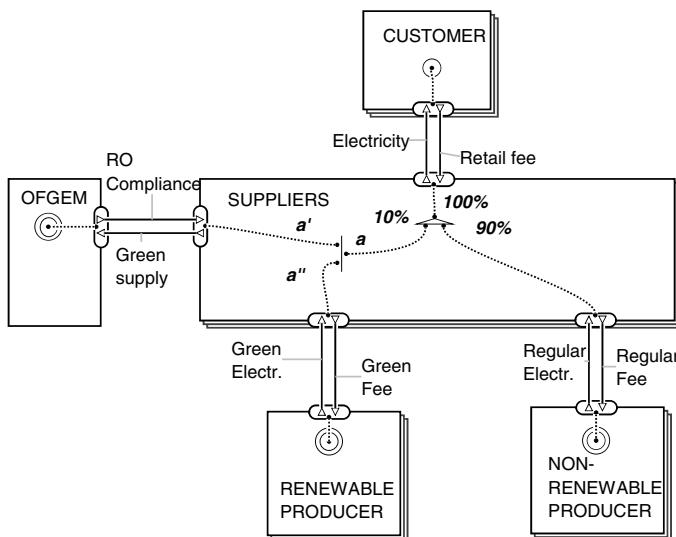
Suppliers prove that they meet their obligations by presenting Renewable Obligation Certificates (ROCs), each representing one Mega Watt/hour (MWh) of produced renewable electricity output. ROCs can be acquired by suppliers from *producers* of green electricity. The producers get ROCs from a government agency, the Office of Gas and Electricity Markets (*Ofgem*) for each MWh of renewable electricity output they produce. In addition, Ofgem maintains a register of all ROCs it has issued.

The suppliers must therefore provide ROCs as evidence of how much MWh of green electricity they have supplied. If a supplier does not have sufficient ROCs to cover his obligation, he must make a deposit into a *buy-out fund*. The buy-out fee is a fixed price per MWh shortfall and is adjusted in line with the Retail Prices Index each year. Premiums from the buy-out fund are paid back to suppliers in proportion to how many ROCs they have presented.
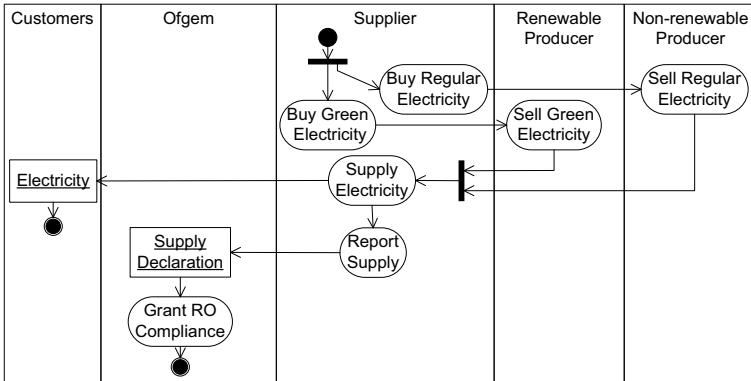
In this case study we apply the $e^3control$ methodology and patterns to understand and find controls. We reverse engineer the ultimate ROC-scheme, by means of the patterns, to show why this scheme is needed from the control perspective. For example, we illustrate how the patterns explain the necessity of introducing ROCs.

We first assume that ROCs do not exist yet. We explain the control problems that may occur in the network. We explicitly take the government's point of view and only describe the problems as perceived by the government, which is represented by Ofgem. Then, step by step, we design the ROC scheme by applying the patterns. As a result, we will demonstrate that the ROC scheme can be explained by means of $e^3control$ patterns.

The case study material is based on participation in the EU BusMod project [5], as well as the Ofgem web site (www.ofgem.co.uk), including [28,29].

Fig. 2. An ideal value model of the ROC case study

Fig. 3. The ideal process model of the ROC case study

4.2   The First e^3 *control* Cycle: Non-tradable ROCs

Step 1: Ideal Situation
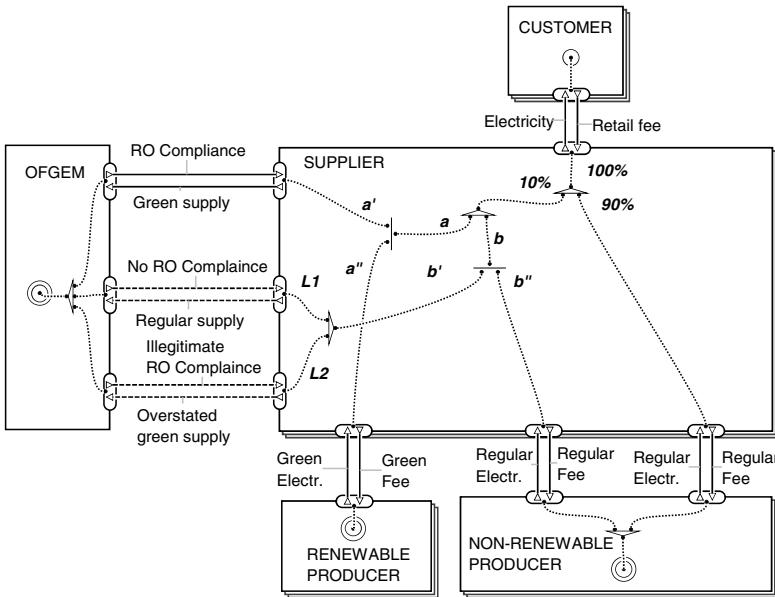
Value view. Fig. 2 presents an e^3 *value* model for the ROC case study. The dependency path in Fig. 2 starts at the *customer*, the final electricity consumers in the UK. The customer buys *Electricity* from the *supplier* and pays the supplier a *Retail Fee* in return[2]. As denoted by the OR-fork at the supplier, the supplier can buy electricity from two sources: from *non-renewable producers* or from *renewable producers*. In the first case, the supplier buys *Regular Electricity* and pays *Regular Fee* in return. In the second case, the supplier buys *Green Electricity* and pays a *Green Fee*. Because green electricity is produced by more expensive renewable technology, the renewable producer asks a higher price for electricity than the non-renewable producer.

   According to the RO regulation, a supplier has to obtain 10% of electricity from renewable sources[3]. In e^3 *value* terms, this means that the electricity delivered by buying *Green Electricity* in Fig. 2 has to account for at least 10% of the *Electricity* supplied to the customers. We also assume that the suppliers behave ideally and always buy 10% of their supply from renewable producers. Therefore, if a supplier buys *Green Electricity*, then he also reports the supply of green electricity to *Ofgem* and receives a statement of compliance with the renewable obligation. This is modeled by the objects *Green Supply* and *RO Compliance* accordingly (see path *a'*).

Process view. In Fig. 3 we represent an ideal process model that corresponds to the ideal e^3 *value* model. The process starts at the supplier who, as in the value model, has the choice of buying electricity from a renewable or a non-renewable supplier. In the first case, the supplier executes *Buy Regular Electricity* , followed by *Sell Regular Electricity* of the non-renewable supplier. In the second case, the suppler executes

---

[2] In this model, the customer buys both green and conventional electricity for the same price.

[3] When the regulation was introduced in 2002, the limit was around 10%. Currently in 2006/07 it is 6.7% and 2.6% in Northern Ireland.
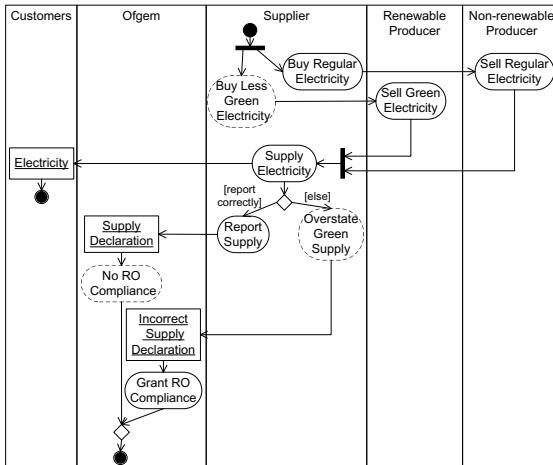
Fig. 4. Sub ideal value model for the problem of not supplying green electricity

Buy Green Electricity , followed by *Sell Green Electricity* executed by the renewable supplier. After that, in both cases *Supply Electricity* is executed by the supplier, which results in a transfer of an object *Electricity* from the supplier to the customer. Further, the supplier reports information about his supply (in MWh) to Ofgem by transferring a statement *Supply Declaration* to Ofgem. In the *Supply Declaration*, the supplier reports how much green electricity was supplied and what part of this electricity was green.

Since this model represents an ideal situation, the supplier is always assumed to behave ideally. In other words, the supplier always buys at least 10% of green electricity. Therefore, at the end of the process the *RO compliance* is always granted.

Step 2: Sub Ideal Situation. There are two types of sub ideal behavior. Firstly, not every supplier complies with the renewable obligation as a supplier can buy a lower percentage of green electricity than the 10% prescribed by the regulation. In this case, the *RO compliance* is not (completely) granted. Secondly, some suppliers can *overstate* the percentage of green supply in order to obtain the RO compliance illegally.

Value view. The sub ideal value model in Fig. 4 models both the ideal and sub ideal behavior of a supplier. The second OR-fork leads to the ideal path *a* and sub ideal path *b*. The ideal path *a* shows the same as in the ideal value model. The sub ideal path *b* corresponds to the two types of sub ideal behavior. In both cases, the supplier buys *Regular Electricity*, instead of *Green Electricity*. This corresponds to the exchanges in the sub path *b''*. Further, the *OR-fork* at the sub path *b'* indicates two possibilities of sub ideal behavior. The sub path, marked with a liability token *L1*, corresponds to a

Fig. 5. Sub ideal process model for the problem of not supplying green electricity

situation in which the supplier reports his low supply of green electricity and does not get the RO compliance. At the sub path, marked with a liability token *L2*, the supplier either overstates his low supply of green electricity or understates the total supply. As a result, he gets the RO compliance illegitimately. The value objects that correspond to this situation are marked as sub ideal with dashed value transfers.

Process view. In Fig. 5 we represent only the sub ideal behavior of the supplier. The supplier buys insufficient green electricity. We model it with a sub ideal activity *Buy Less Green Electricity* instead of the *Buy Green Electricity*, as in the ideal process model. The supplier has the choice to report the true supply of green electricity or to overstate it. In the first case, the supplier transfers a *Supply Declaration* in which he informs Ofgem about insufficient green supply and, as a result, he does not get the RO compliance. In the second case, the supplier overstates the percentage of green supply and transfers an *Incorrect Supply Declaration*. As a result, the supplier gets the RO compliance illegitimately.

Step 3: Reduce Sub Ideality by Applying Control Patterns. In order to solve the control problems, Ofgem should implement one or more control mechanisms. We exemplify how the Penalty and Incentive patterns can be used to motivate suppliers to supply the right amount of renewable electricity. We have a process for selecting appropriate patterns for found controls problems (see [27]), which we do not explain due to lack of space. We illustrate below how the Penalty and Incentive patterns contribute to solving the found control problems.

Process view. Both the Penalty and Incentive patterns require the primary actor Ofgem to check the outcome of the sub ideal counter activity *Buy Green Electricity* and, then to reward or punish the supplier. To model this, we first add *Verify Compliance* to Fig. 6. This is an instance of the *Verify* activity in the patterns (see Figs. 13 and 14).

Furthermore, according to the pattern, *Verify Compliance* requires two inputs: *Supporting Statement* and *To-be-verified Statement* (see the Appendix). The *To-be-verified Statement* gives information about a percentage of electricity supplied from renewable sources. This corresponds to the object *Supply Declaration*, presented in the model.

According to the pattern, the to-be-verified statement *Supply Declaration* must be produced by an activity that *witnesses* the activity *Buy Green Electricity*. In addition, the pattern requires this witnessing activity to be executed either by the primary actor Ofgem or by a party independent and socially detached from the counter actor Supplier. If this is not the case, the *violation of segregation of duties* occurs, since the supplier who is responsible for buying green electricity also reports about it.

In the ideal model, the to-be-verified statement *Supply Declaration* is produced by the Supplier. This violates the requirements of the pattern. Therefore, we explicitly model the witnessing activity *Witness Green Supply*, instead of only the reporting activity. Secondly, we assign this activity to an actor, who is independent from the Supplier and is able to produce trustworthy information about the supply. Ideally such an actor is the primary actor Ofgem. However, as it is just an administrative body, Ofgem does not have the resources to control each supplier. So, Ofgem delegates control of the suppliers to some trusted party.
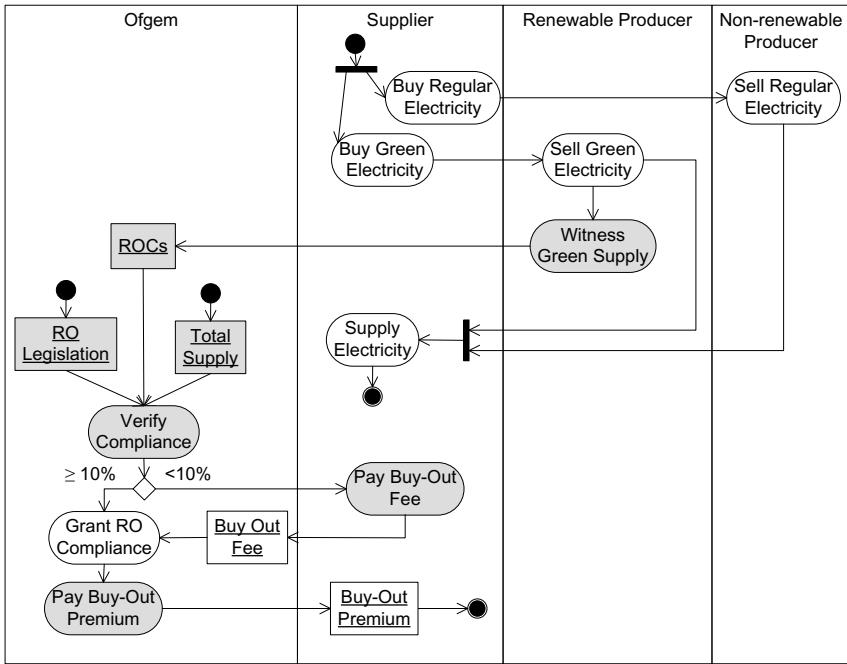
The role of such a trusted party can be played by the renewable producer who supplies electricity to suppliers. This party is therefore physically able to keep track of how much green electricity is bought by each supplier. In Fig. 6, the activity *Witness Green Supply* is assigned to the renewable producer.

The third change we make is to rename the to-be-verified statement. This statement, previously called *Supply Declaration*, is now called *Renewable Obligation Certificate (ROC)*. One ROC is issued for each Mega Watt/hour (MWh) of eligible renewable output. According to the pattern, the ROC is fed into the activity *Verify Compliance*, which compares whether the ROCs of one particular supplier represent 10% of his total supply.

The ROC only represents the amount of green electricity. However, the important criterion for granting RO compliance is the *share* of the green electricity within the supplier's total electricity supply. Therefore, we add another to-be-verified statement *Total Supply*, which represents this information. As with *ROC*, the *Total Supply* must be generated by an actor who is independent and not acting in the interests of the supplier. For instance, the data about the total supply could be retrieved from the final customer or from the supplier's annual accounts, assuming they are trustworthy. In the model we show that the *Total Supply* is generated by Ofgem. Data concerning the total supply of each supplier is easily accessible to a governmental organization like Ofgem.

The *Verify Compliance* activity requires a supporting statement, namely, information, which is needed to decide whether the *RO Compliance* should be granted. Such a document is a *RO legislation*, stating e.g. the required percentage of green electricity (which we assume is 10%), which producers are qualified to hold the status of 'renewable', to which customers should the reported green electricity be supplied, etc.

In addition, the activities are assigned in a proper order, as required by the control principles. The activity *Witness Green Supply* is executed after the *Buy Green Electricity* activity and before *Verify Compliance* activity. In addition, the *Verify Compliance* activity is executed before the primary activity *Grant RO Compliance*.
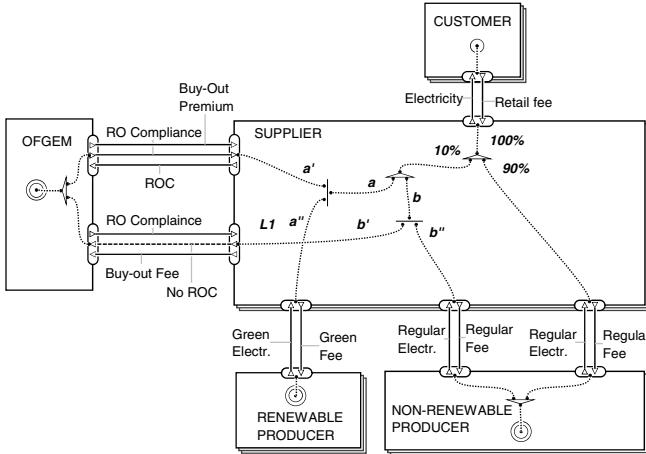
Fig. 6. Solution process model with penalties and incentives

According to the penalty pattern, the object *Penalty* should be added. The penalty should be transferred to Ofgem by a supplier who supplies less than 10% of green supply. According to the RO regulation, the supplier who does not have sufficient green supply to cover his obligations must make a deposit into the *buy-out fund* [28]. Such a payment corresponds to the object *Penalty* of the Penalty pattern. In Fig. 6 the *Buy-out Fee* is paid by the suppliers according to the RO regulation. According to the pattern, if the outcome of the *Verify Compliance* states that the green supply is less than 10%, the RO compliance is granted only after the buy-out fee is paid by the supplier. The *Pay BuyOut Fee* activity corresponds to the *Pay Penalty* activity of the Penalty pattern.

According to the solution given by the Incentive pattern, the *Incentive* object should be added. This object should be transferred to the supplier who supplies at least 10% of green supply. The buy-out fund is paid back to suppliers in proportion to how much green electricity they have purchased [28]. This payment, henceforth called *Buy-out Premium*, represents the incentive. We add it to Fig. 6.

In addition, according to the pattern, we model that if the outcome of the *Verify Compliance* states that the green supply is more than 10%, then RO compliance is granted and the *Buy-out Premium* is paid. The *Pay BuyOut Premium* activity corresponds to the *Pay Incentive* activity of the Incentive pattern.

Note that in the application of this pattern we have also used the Simple Delegation pattern (see http://www.e3value.com/e3family/e3control/patterns). This is needed

Fig. 7. Solution value model with penalties and incentives

to model that the *Witness Green Supply* activity is delegated by Ofgem to the trusted actor Renewable Producer.

Value view. We now make appropriate changes in the ideal value model. The changes have been caused by the introduction of penalties and incentives as well as by the delegation of the witnessing activity.

We add a new value transfer, indicating a penalty, as an outgoing value object of the counter actor Supplier in the sub ideal path *L1*. We add it to the transfer of *No RO Compliance* and *Regular Supply*, and change *No RO Compliance* to *RO Compliance*.

Also, since we have renamed the *Supply Declaration* to *ROC* in the process model, the ROC also appears as value object in the value model. We model *ROC* instead of the value object *Green Supply*, and *No ROC* instead of the value object *No Green Supply*.

The resulting value model is presented in Fig. 7 and this corresponds to reality. The penalty is represented by the value transfer *Buy-out Fee*. So, at the sub ideal path *b*, where the supplier does not supply enough green electricity, he is obliged to pay a buy-out fee in order to cover the RO.

The Incentive pattern requires that an incoming value object *Incentive* should be added to the Supplier in the ideal value transfer. The incentive is the buy-out premium paid by Ofgem to compliant suppliers. The incentive value object *BuyOut Premium* is added to the transfer of *Green Supply* and *RO Compliance*.

4.3   The Second $e^3control$ Cycle: Tradable ROCs

The process model in Fig. 6 represents only a part of the actual ROC scheme. Due to the nature of the electricity business, suppliers can buy and sell electricity several times to other suppliers before it reaches the final customer. According to RO regulation, ROCs can be claimed by the supplier who delivers the associated green electricity to final customers. If a supplier sells green electricity to another supplier, the ROCs should also
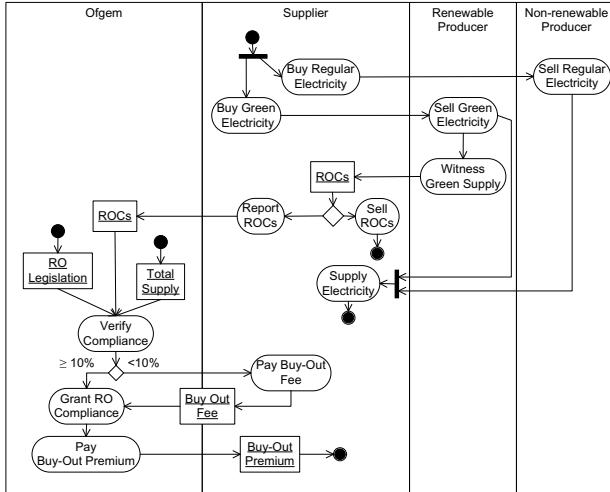
Fig. 8. An ideal process model with tradable ROCs
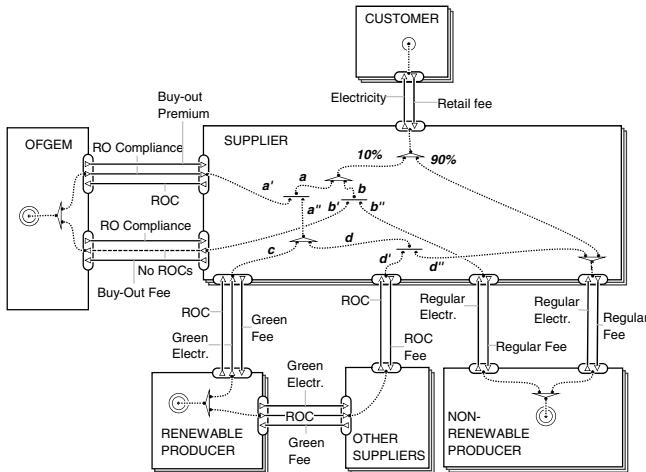
Fig. 9. An ideal value model with tradable ROCs

be transferred to this other supplier. In addition, ROC's can be traded themselves on the market. The ROC is in fact a security similar to stocks and bonds. As will be explained later, the ROC market was created to stimulate green electricity production. Therefore, additional controls are required.

Step 1: Ideal Situation. The ideal process model of the scenario with tradable ROCs is shown in Fig. 8. Unlike in the solution process model in Fig. 6, the ROCs are transferred to the supplier before being transferred to Ofgem to comply with the required

percentage of supplied renewable energy. The supplier has the choice of selling the obtained ROCs or of reporting them to Ofgem. This choice is denoted by the UML decision element at the supplier. If the supplier reports the ROCs to Ofgem, the verification process of charging the buy-out fee or the paying buy-out premium remains the same as before. However, if the supplier sells the ROCs, they are not presented to Ofgem.

The corresponding value model is shown in Fig. 9. The following changes have been made compared to the situation without tradable ROCs in Fig. 7. Firstly, suppliers obtain ROCs from the renewable producers. Secondly, unlike in the solution value model in Fig. 7, the ideal dependency path now offers a choice between (1) obtaining ROCs for free while buying the (more expensive) green electricity in path *c* or (2) buying ROCs separately and purchasing the (cheaper) regular electricity in the path *d*. Thus, because ROCs can be traded, they are modeled as value objects, and not as process objects only.

Step 2: Sub Ideal Situation. The new ideal models in Figs. 8 and 9 do not comply with the prescriptions of the Penalty and Incentive patterns in Figs. 7 and 6. Specifically, to perform the *Verify Compliance* activity, Ofgem has to rely on information received from the supplier. This implies that the supplier performs the *Witness Green Supply* activity. As already explained, this contradicts with the pattern, since the supplier should not report his own activities. For example, the supplier can forge ROCs and overstate the number of supplied green electricity.
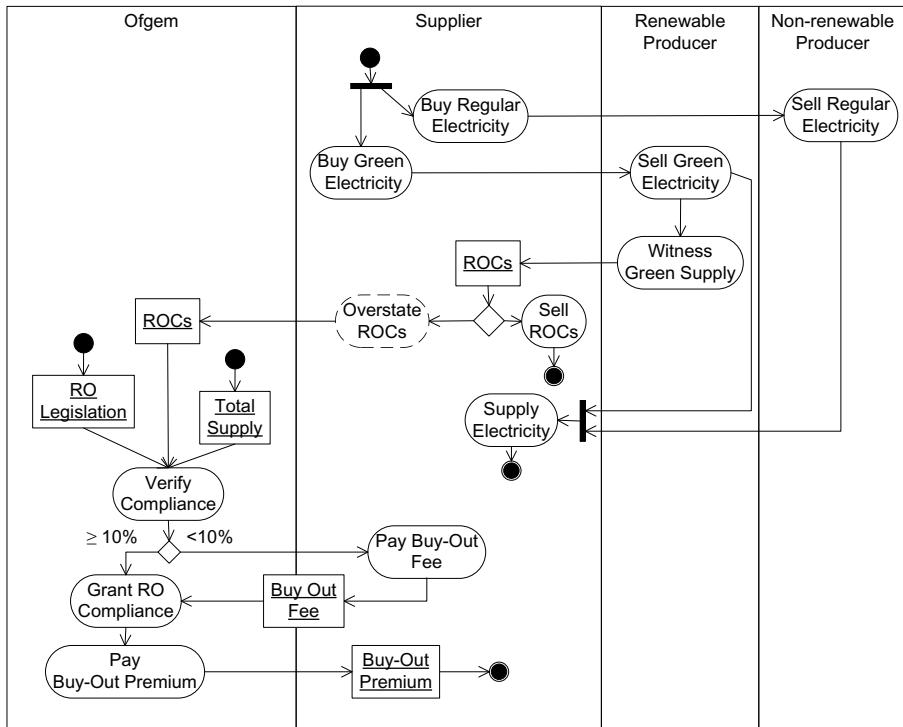
In Fig. 10 we show the sub ideal process model of the scenario with tradable ROCs. The supplier overstates the number of ROCs he has, which is modeled by the activity *Overstate Green Supply*. This corresponds to the transfer of *No ROCs* by the supplier in the sub ideal value model in Fig. 11. Because the overstatement remains undetected, the supplier receives the *RO Compliance* and even gets the *BuyOut Premium* in return. This path is marked with the liability token L3.

Note that the supplier has an illegitimate interest in overstating the number of ROCs, which does not depend on whether he can cover the RO obligation or not. If the supplier has enough ROCs to cover the obligation, he may overstate ROCs to receive the buy-out premium (see path *a*). If the supplier has not enough ROCs to cover the obligation, he is motivated to overstate ROCs to avoid the buy-out fee penalties (see path *b*).
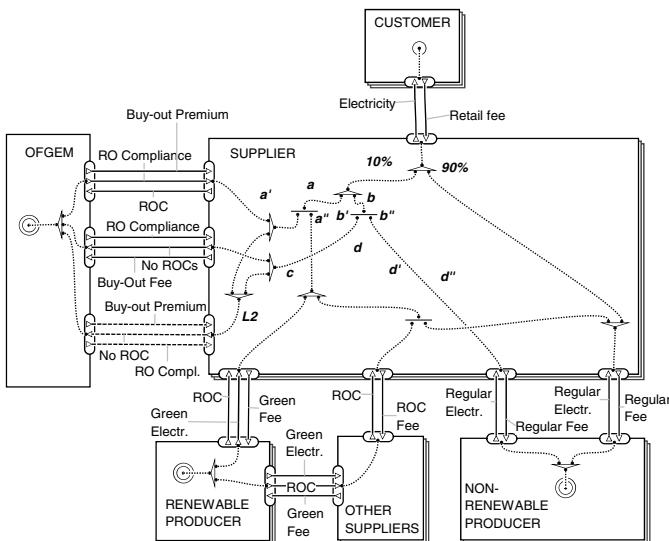
Step 3: Reduce Sub Ideality by Applying Control Patterns. We now apply the Execution Monitoring pattern to reduce sub ideality.

Process view. Following the solution given by the Execution Monitoring pattern, we add a new verification activity *Verify ROC* to Ofgem. It verifies the *Present ROCs* activity of the Supplier. To do so, it checks if a *ROC*, submitted by a supplier, corresponds to a ROC reported by the renewable producer.
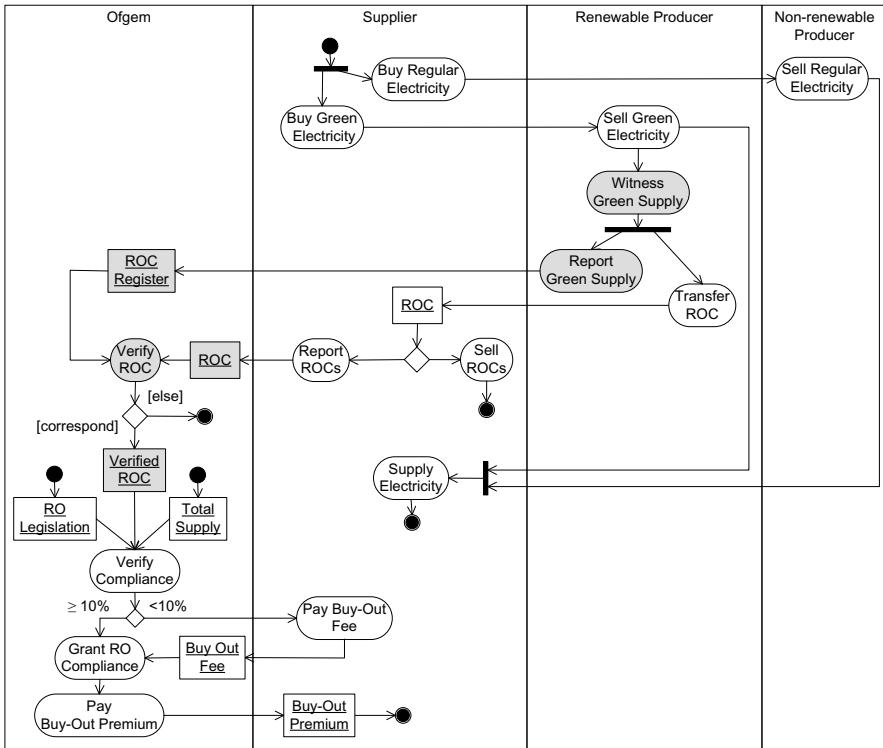
In addition, unlike in the ideal model with ROCs in Fig. 8, the renewable producer not only issues a ROC to the supplier, but also reports the number of issued ROCs to Ofgem. This is modeled with an object *ROC Register*. The ROC register is an electronic, web-based system, supported by Ofgem, which allows generators and suppliers to view the ROCs they hold and to transfer ROCs to other parties. In this way, Ofgem can verify the authenticity of each ROC.

Fig. 10. A sub ideal process model with tradable ROCs

Fig. 11. A sub ideal value model with tradable ROCs

Fig. 12. Controls in a process model with tradable ROCs

Note that the activity *Report ROCs* is added not because of a pattern's prescription. It is added because the UML language restrains modeling the exchange of the object (*ROC Registry* in this case) directly from the AND-join (the black thick bar). We therefore add an activity in between.

In this solution, an *ROC* plays the role of the to-be-verified statement, while the *ROC Register* plays the role of the supporting document. Thus, the renewable producer plays the role of the provider of a supporting document.

After the verification of an ROC, the *Verified ROC* object is used. As in the previous $e^3control$ cycle, the *Verified ROC* plays the role of the to-be-verified statement for the verification activity *Verify Compliance*.

The value model does not change and is the same as the ideal value model in Fig. 9.

5   Conclusion and Discussion

In this chapter, we have presented a series of patterns for the value-based design of inter-organizational controls, and demonstrated three patterns in a case study. The

proposed patterns take an explicit *economic value* perspective on the control problem: It is important to understand first the valuable objects to be safeguarded, before designing controls ensuring proper transfer of objects.

We have contributed a structure for stating control patterns in terms of ideal value&activity models, sub ideal value&activity models representing the fraud, and possible solutions expressed using similar model types. The patterns themselves stem from two sources: (1) accepted theory on the principal-agent relations, accounting, and auditing, and (2) four industrial case studies we have performed to design inter-organizational controls.

For $e^3control$, many further research issues can be identified, and we present two of them. First, there is the issue of the cost of controls. So far, we have studied controls that have inherent economic value aspects (e.g. incentives and penalties), and therefore have an impact on the e^3value model of a networked organization. However, implementing controls usually comes with a price. This 'cost of control' should also be considered when analyzing control issues in value networks. Second, controls may interact. If we first analyze and select controls for actor A, and thereafter actor B, the resulting set of the controls for the *network* can be different compared to first considering actor B, and thereafter actor A. Actually, this can be seen as an example of feature interaction, and a more structured approach is needed to deal with this interaction.

More in general, the $e^3control$ methodology is a member of the e^3value modeling suite, focusing on understanding the exchange of valuable objects in networks of enterprises. Although value modeling has proven to be useful in numerous case studies, there exist also a number of research challenges to be addressed. First, there is the issue of *valuation*. Value objects reflecting money are relatively easy; the value of such objects coincides with the amount of money exchanged. However, in value modeling it is sometimes needed to assign economic value to non-money objects also. For instance, a final customer of a service should value the service outcome obtained. Essentially, we are then interested in the economic utility function of the customer, which is difficult to obtain. But even objects directly reflecting money pose interesting problems. Usually, a 'money object' refers to a price to be paid for a product or service outcome. However, the pricing scheme can only be partly based on market considerations (e.g. the amount of money the customer is willing to pay). Another factor is the cost needed to produce the product or service outcome. As in our field, the services are usually *ICT* services, there should be a clear, and understandable, relationship between the valuable ICT service offered, and the costs of the ICT service. In addition to that, the value network takes a commercial perspective on a network of actors. But, in the case of ICT, other perspectives are required, such as perspectives on the supporting ICT, and the business processes that should be carried out, to develop the value network at hand. One of the key questions is then how to properly align these perspectives during the design of the actor network, and how to ensure that these perspectives remain aligned.

References

1. Weill, P., Vitale, M.R.: Place to Space: Migrating to eBusiness ModelsEconomy and Society. Harvard Business School Press, Boston (2001)
2. Gordijn, J., Akkermans, J.M.: *e3-value*: Design and evaluation of e-business models. IEEE Intelligent Systems, Special Issue on e-Business 16(4), 11–17 (2001)
3. Gordijn, J., Akkermans, J.M.: Value-based requirements engineering: Exploring innovative e-commerce ideas. Requirements Engineering 8(2), 114–134 (2003)
4. Gordijn, J., Yu, E., van der Raadt, B.: E-service design using i* and e3value modeling. IEEE Software 23(3), 26–33 (2006)
5. Gordijn, J., Akkermans, J.M.: Business models for distributed energy resources in a liberalized market environment. The Electric Power Systems Research Journal 77(9), 1178–1188 (2007), doi:10.1016/j.epsr.2006.08.008
6. Castro, J., Kolp, M., Mylopoulos, J.: Towards requirements-driven information systems engineering: the tropos project. Information Systems 27, 365–389 (2002)
7. Kartseva, V., Gordijn, J., Tan, Y.-H.: Developing a modelling tool for designing control mechanisms in network organisations. International Journal of Electronic Commerce (2005)
8. Ronmey, M.B., Steinbart, P.J.: Accounting Information Systems, 10th edn. Prentice Hall, New Jersey (2006)
9. ISACF: Control objectives for information and related technology (cobit). Technical report, Information Systems Audit and Control Foundation (ISACF) (2005)
10. Eisenhardt, K.M.: Agency theory: An assessment and review. Academy of Management Review 14(1), 57 (1989)
11. Anthony, R.N., Govindarajan, V.: Management Control Systems, 11th edn. McGraw Hill Higher Education, New York (2003)
12. Kartseva, V., Hulstijn, J., Baida, Z., Gordijn, J., Tan, Y.-H.: Towards control patterns for smart business networks. In: Vervest, P., van Heck, E. (eds.) Proceedings of the Smart Business Networks Initiative Discovery Session. Springer, Heidelberg (2006)
13. Kartseva, V., Hulstijn, J., Tan, Y.-H., Gordijn, J.: Towards value-based design patterns for inter-organizational control. In: Waldena, P., Gricar, J. (eds.) Proceedings of the 19th Bled Electronic Commerce Conference on eValues, Bled, Slovenia. University of Maribor (June 2006)
14. Kartseva, V., Gordijn, J., Tan, Y.-H.: Designing control mechanisms for value webs: The internet radio case study. In: Vogel, D.R., Walden, P., Gricar, J., Lenart, G. (eds.) Proceedings of the 18th BLED conference (e-Integration in Action), pages cdrom, Maribor, SL (2005)
15. Kartseva, V., Hulstijn, J., Gordijn, J., Tan, Y.-H.: Modelling value-based inter-organizational controls in healthcare regulations. In: Proceedings of the 6th IFIP conference on e-Commerce, e-Business, and e-Government, Turku, Finland (2006)
16. Alexander, C.: The Oregon Experiment. Oxford University Press, New York (1975)
17. Fowler, M.: Analysis Patterns: Reusable Object Models. Addison-Wesley, Reading (1997)
18. Seruca, I., Loucopoulos, P.: Towards a systematic approach to the capture of patterns within a business domain. The Journal of Systems and Software (67), 1–18 (2003)
19. Hollander, A.S., Denna, E., Cherrington, J.O.: Accounting, Information Technology, and Business Solutions, 2nd edn. McGraw-Hill, Irwin (1999)
20. Bons, R.W.H.: Designing Trustworthy Trade Procedures for open Electronic Commerce. PhD thesis, University of Rotterdam (1997)
21. Bons, R.W.H., Dignum, F., Lee, R.M., Tan, Y.-H.: A formal analysis of auditing principles for electronic trade procedures. International Journal of Electronic Commerce 5(1), 57–82 (2000)

22. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley, Boston (1995)
23. Fowler, M., Scott, K.: UML Distilled, 2nd edn. Addison Wesley, Reading (2000)
24. Nuseibeh, B., Kramer, J., Finkelstein, A.: A framework for expressing relationships between multiple views in requirements specification. IEEE Transactions on Software Engineering 20(10), 760–773 (1994)
25. Bons, R.W.H., Lee, R.M., Wagenaar, R.W.: Designing trustworthy interorganizational trade procedures for open electronic commerce. International Journal of Electronic Commerce 2(3), 61–83 (1998)
26. Chen, K.T., Lee, R.M.: Schematic evaluation of internal accounting control systems. EU-RIDIS Research Monograph, RM-1992-08-1, Erasmus University Rotterdam (August 1992)
27. Kartseva, V.: Designing Controls for Network Organization: A Value-Based Approach. PhD thesis, Vrije Universiteit Amsterdam (2008),
    http://docs.e3value.com/bibtex/pdf/Kartseva2008.pdf
28. OFGEM. The renewables obligation: Annual report 2003-2004. Technical report, Office of Gas and Electricity Markets, United Kingdom (2004)
29. OFGEM. Renewables Obligation - Atlantic Electric and Gas Limited and Maverick Energy Limited. Technical Report R/65, Office of Gas and Electricity Markets, United Kingdom (2004)

Appendix: The $e^3control$ patterns

Below we present three control patterns that we use in the case study of this chapter. Other patterns can be found at http://www.e3value.com/e3family/e3control/patterns.
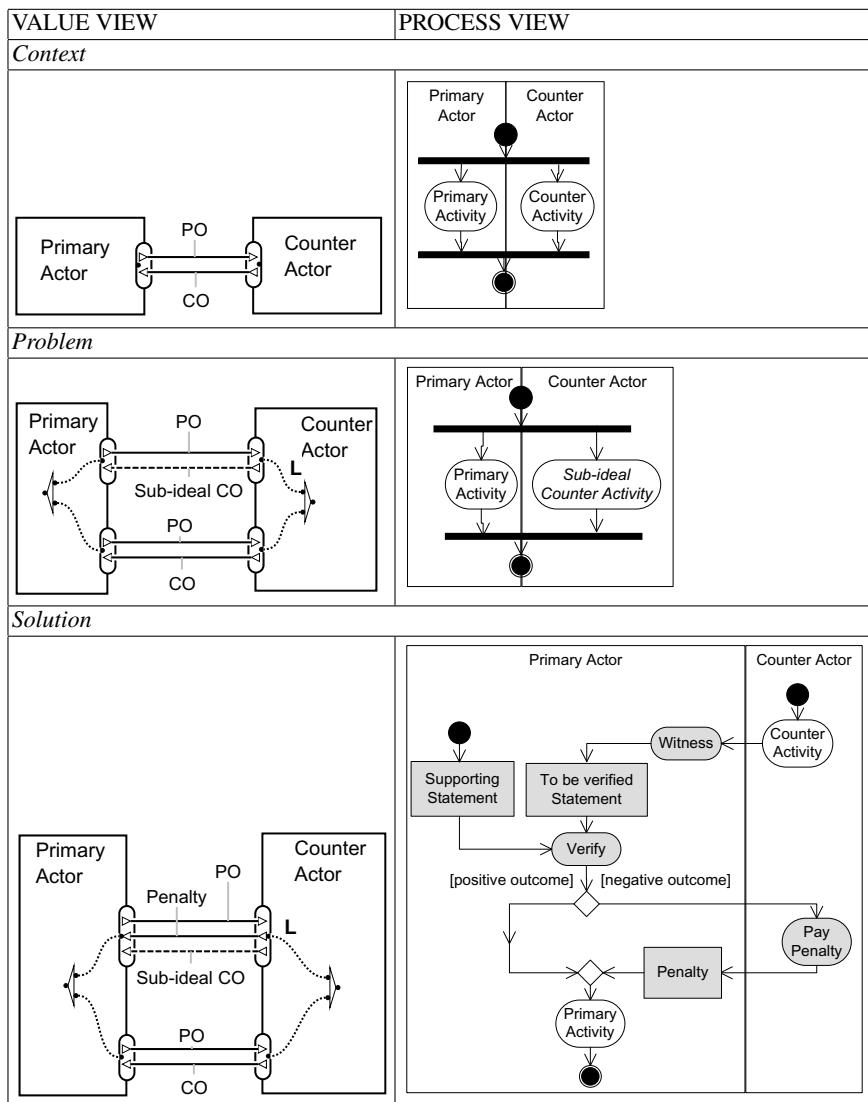
5.1   Penalty Pattern

The penalty pattern punishes the counter actor for sub ideal behavior by introducing a value object that decreases the accumulated value of the counter actor in the *ideal path*.

5.2   Incentive Pattern
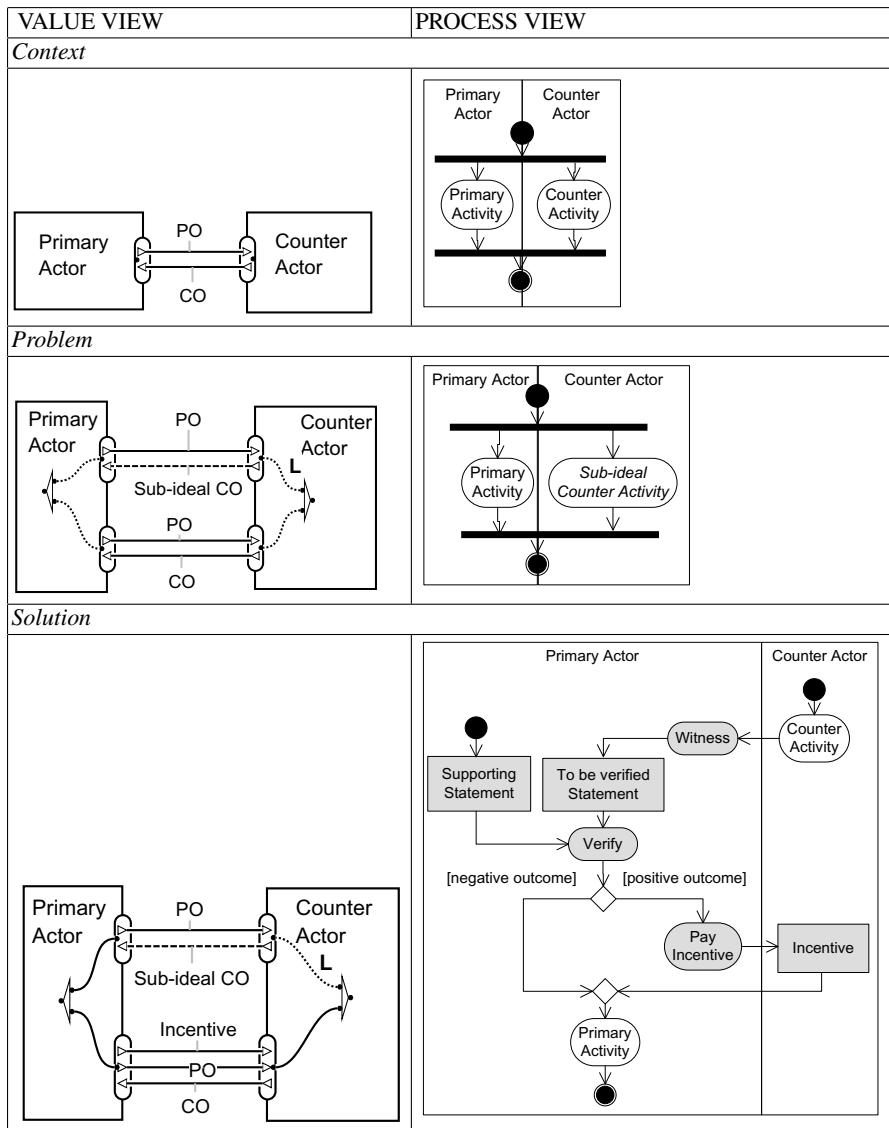
The incentive pattern rewards the counter actor for ideal behavior by introducing a value object that increases the accumulated value of the counter actor in the *ideal path*.
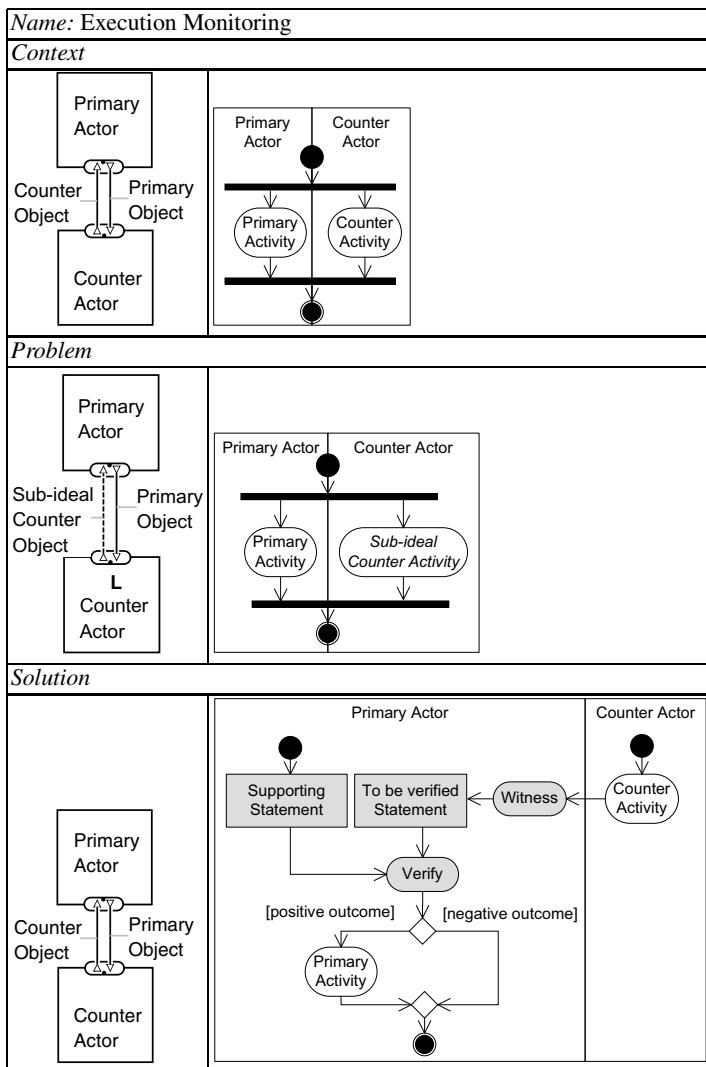
5.3   Execution Monitoring Pattern

The *Execution Monitoring* pattern describes the control problem in which a counter actor does not execute his commitments or executes them in a sub ideal way (e.g. not as agreed in the contract). The control mechanism requires the primary actor to monitor counter activity. This solution is very similar to the Partner Screening pattern, the only difference being that verification concerns the counter activities under the contract and not an actor's past counter activities.

Fig. 13. Penalty pattern

VALUE VIEW	PROCESS VIEW
Context	
Problem	
Solution	

Fig. 14. Incentive pattern

Fig. 15. Pattern 'Execution Monitoring'

Section 4: Requirements Intertwining

Pericles Loucopoulos

Business analysts are being asked to develop increasingly complex and varied business systems that need to cater to the changing and dynamic market conditions of the new economy. This is particularly acute in today's turbulent business environment where powerful forces such as deregulation, globalisation, mergers, advances in information and telecommunications technologies, and increasing education of people provide opportunities for organising work in ways that have never before been possible. Enterprises attempt to create wealth either by getting better at improving their products and services or by harnessing creativity and human-centred management to create innovative solutions. In these business settings, requirements become critical in bridging system solutions to organisational and societal problems. They intertwine organisational, social, cognitive, and implementation considerations and they can provide unique insights to change in systems and their business context. Such design situations often involve multiple stakeholders from different participating organisations, subcontractors, divisions, etc., who may have a diversity of expertise, come from different organisational cultures and often have competing goals. The success or failure of many projects depends, to a large extent, on understanding the contextual setting of requirements and their interaction amongst a diverse population of stakeholders.

In this section we review a set of key issues that emerge in Requirements Engineering paying particular attention to two factors, (a) the role of requirements in a business context and (b) the treatment of requirements in multi-stakeholder settings. The first four articles deal with issues relating to the interaction between business and requirements concepts whereas the subsequent three deal with issues relating to stakeholders' requirements interaction.

The first article by Colette Rolland titled "Exploring the Fitness Relationship between System Functionality and Business Needs" examines the issue of fitness relationship, by considering the twin problems of establishing such a relationship and preserving it in the face of change. To this end, the article investigates three key issues. First, the conceptual mismatch between business and system languages is considered and a solution is put forward in using a common language whereby the functionality of a system is abstracted at an intentional level and business constructs are also formalised in intentional terms. Second, a formal approach to modelling and measuring the fitness relationship is considered through the definition of correspondences between a set of components. Third, the preservation of the fitness relationship at different levels of detail is considered by using a refinement mechanism as a way to control the quality of the refinement.

The second article by Varum Grover and Samuel Otim titled "A Framework for Business Change Requirements Analysis," examines the management of requirements change in the context of business change. It argues that the pressures for enterprises to remain competitive requires substantial efforts in business process change and that IT is normally at the centre of such changes. The authors argue that a design orientation helps to manage the complexity of business change. Within such a design paradigm, it is recognised that requirements definition at the business task level may be

time consuming and counterproductive in most business situations where the pace of change is fast. To meet the challenge of requirements evolution, this article puts forward a framework that focuses on the process of change and the decisions that need to be made. This framework highlights the importance of managing the process of change by maintaining contextual knowledge about process models and process redesign rules.

The third article by Peri Loucopoulos and Joy Garfield titled "The Intertwining of Enterprise Strategy and Requirements", examines the issue of the potential impact that a set of requirements are likely to have on the strategy of a business. It raises the point that system requirements are not simply derived linearly through an analysis of business goals but often these business goals are influenced through alternative implementation choices. This feedback, from system requirements to business strategy, leads to an improved dialogue between 'problem' and 'solution' worlds, which fits very well the design paradigm espoused by the book.

The fourth article by Paul Otto and Annie Anton titled "Managing Legal Texts in Requirements Engineering" examines the effect that laws applicable to specific business and societal settings influence Requirements Engineering. This article examines the many challenges presented by legal documents given the complexity of such documents as well as many other challenges faced by designers such as domain-specific definitions, ambiguity, cross references and frequent amendments. Legal compliance is considered by many practitioners as a major driving force in information security. The challenge for requirements engineers seems to be on how to determine the regulation that is applicable to the case in hand and how to specify requirements to achieve compliance with these regulations. The article provides a comprehensive survey of research efforts in modelling and using legal texts in the systems development lifecycle.

The fifth article by Mark Bergman titled "Requirements' Role in Mobilizing and Enabling Design Conversation" considers the issue of design conversation, arguing that stakeholders need to engage into meaningful conversation towards desired results. Such a conversation needs to be reflective upon a set of shared design representations and to this end the author puts forward a research agenda and approach for improving the role of requirements in promoting design conversation.

The sixth article by Davide Bolchini, Franca Garzotto, and Paolo Paolini titled "Design Requirements for Communication-Intensive Interactive Applications" focuses on requirements issues that arise due to complexity in computer-mediated communication applications. The objective of such applications is to be informative and persuasive, termed in this article 'infosuasive' applications. This article addresses the early design phase of infosuasive applications when requirements are normally vague, unfocused and possibly inaccurate. During this phase there are many different types of stakeholder involved from both the user community and the designer community. The article proposes a conceptual framework aimed at supporting members of both communities in order for them to be able to share their mental models, to integrate their different viewpoints, and to plot a direction amongst the many different design options.

The final article in this section by Yijun Yu, Nan Niu, Bruno Gonzalez-Baixauli, and John Mylopoulos titled "Requirements Engineering and Aspects" is concerned with issues of validation of stakeholders' requirements. In this article, the authors

explore the idea of system implementation assisting the validation of early requirements. The idea of 'requirement aspect' is used as a stakeholder concern that permeates across other requirements concerns. An approach is presented whereby aspects are discovered and tested within an intentional framework.

We believe that these seven articles provide useful insights on issues relating to one of the key principles discussed in the introduction of this book, namely the intertwining of requirements. Hopefully, this section will stimulate the reader to exploit further some of the research questions raised here or to apply some of the proposed approaches so that through further theoretical and empirical work the issue of requirements intertwining can become an integral part of contemporary work in Requirements Engineering.

Exploring the Fitness Relationship between System Functionality and Business Needs

Colette Rolland

Université Paris1 Panthéon Sorbonne, Centre de Recherche en Informatique,
90 rue de Tolbiac, 75013 Paris, France
`Colette.Rolland@univ-paris1.fr`

Abstract. Fitting information systems to business needs is considered equally important by both, the Requirements Engineering and MIS communities. Even though alignment/fit clearly appears as desirable, a number of issues still remain unsolved as for example: (i) the achievement of alignment (ii) its management over time, (iii) the identification of non fit and, (iv) its evaluation. This paper gives to fitness a central position and introduces the notion of a *fitness relationship* and its *measurement*. In doing so, this paper tackles the social context and requirements intertwining as pointed out in the book framework. It highlights two sets of issues, one involved in *understanding* this relationship and the second in *engineering* it. It also points out broad directions and trends in resolving these issues.

Keywords: Business/IS alignment, fitness modelling, fitness measure, strategic alignment, intentional modelling.

1   Introduction

Requirements Engineering (RE) tries to ensure that system functionality matches business needs. It does this by establishing a relationship between the "whys" and the "whats" of the system To-Be. The former deals with the objectives of the organisation and the way they influence the requirements of the system. The latter refers to the functionality that meets these requirements. RE inherently deals with business-system alignment. Potts [1] identified 'fitness for use as a system quality that matters most'. However, the notion of fitness, its properties and characteristics have been peripheral issues for the RE community. Through the issue of 'Intertwine Requirements and Contexts' identified in its framework, this book recognises the growing importance of mastering the fitness relationship between business, organisation and community contexts and requirements.

In the MIS community, alignment between IT systems and the business they support has been considered as a key issue for several years by both researchers and professionals. As [2] [3] [4] [5] or [6] already showed it, strategic alignment between IS and business objectives is a top priority for CIOs and IT executives. There is also, a large corpus of empirical and theoretical evidence that more tactical alignment between information systems and business processes improves organizational performance (e.g. [7], [8], [9], [10], [11]). Furthermore, studies

highlight the lack of alignment as a major cause for business processes failure in providing return on investments. However, even though alignment clearly appears as a desirable, a number of issues still remain unsolved. For example: (i) the achievement of BP-IS alignment, (ii) its management over time [12], (iii) the identification of non fit and, (iv) its evaluation [13].

This paper explores what we refer to as the *fitness relationship* (inspired by [1] so as to reveal its nature and its engineering process. In this exploration, we shall consider the twin problems of establishing the fitness relationship and preserving it in the face of change. The presentation is organized around two sets of issues related to (a) *understanding the fitness relationship* and (b) *engineering the relationship*. The former deals with characterizing, conceptualizing, and modelling, the fitness relationship whereas the latter deals with establishing and preserving it in the face of change. Issues relating to (a) above, represent the static point of view whereas those of (b) above follow the dynamic viewpoint. Whereas the subject of this paper in its whole relates to the 'Intertwine Requirements and Contexts' principle of the book framework, issues (b) above relates to the 'Evolve Designs and Ecologies' issue pointed out in this book framework. We discuss them in turn.

2   Understanding the Fitness Relationship

It has been recognized that there is a *conceptual mismatch* between the *business* and the *system* levels. It is considered necessary to minimize this mismatch. This leads to our first issue, **Issue1: Conceptual mismatch**. Despite alignment/fit being mentioned in many approaches, it is rarely a concept per se and there is no agreement on its definition. *Defining* and *modelling the fitness relationship* seem to be a prerequisite to support *measurement* of the degree of fit. Issue 2 considered below is therefore: **Issue 2: Modelling and measuring the relationship**. It may be inconvenient to view the fitness relationship as one monolithic flat structure. A layered approach may facilitate understanding and mastering its complexity. Therefore, in order to present the fitness relationship at different levels of detail, it is necessary to have a *refinement* mechanism as well as a means to control the *quality* of the refinement. **Issue 3: Dealing with the complexity of the relationship**.

To sum up, it can be surmised that an understanding of the fitness relationship requires a position to be adopted on the following issues

Issue 1: Conceptual mismatch;
Issue 2: Modelling and measuring the relationship;
Issue 3: Dealing with the complexity of the relationship.

Issue1: Conceptual mismatch. It has been recognized that there is a *conceptual mismatch* between the business and the system languages [14][15] and that it is considered necessary to minimize this mismatch. One way to do this is to leverage the functional system view to a requirements view and to express

both with the *same language*. This shall allow expressing the fitness relationship in a more straightforward manner. Goal centred languages seem to be the most adequate for this purpose as they explicitly capture the *why* and *how* of both system functionality and business process. With this position, the resolution of the mismatch requires on one hand, to abstract intentionality from the functionality and, on the other hand, to formalize the business in intentional terms.

Our experience is based on the use of the *Map* representation system for a *uniform representation of business goals and system functionalities*. See [16] for a detailed description. A brief overview is as follows: A map is a labelled directed graph (see Fig. 1) with *intentions* as nodes and *strategies* to achieve them as edges. The directed nature of the graph shows which intentions can follow which one. An edge enters a node if its strategy can be used to achieve the corresponding intention. The key element of a map is a *section* defined as a triplet $\langle I_i, I_j, S_{ij} \rangle$ and represents a way to achieve the target intention I_j from the source intention I_i following the strategy S_{ij}. Each section of the map captures the condition to achieve an intention and the specific manner in which the process associated with the target intention can be performed. Sections of a map are *connected* to one another:

(a) When a given intention can be achieved with different strategies. This is represented in the map by several sections between a pair of intentions. Such a map topology is called a *multi-threa*d.

(b) When a goal can be achieved by several combinations of strategies. This is represented in the map by a pair of intentions connected by several sequences of sections. Such a topology is called a *multi-path*. In general, a map from its *Start* to its *Stop* goals is a multi-path and may contain multi-threads

As an example, the map of Fig. 1 contains eleven sections C0 to C11. It can be seen that C1, C2 and C3 together constitute a multi-thread whereas C2, C5, C10, C11 and C6, C11 are two paths between the *Start* and *Stop* intentions of the map constituting a multi-path.
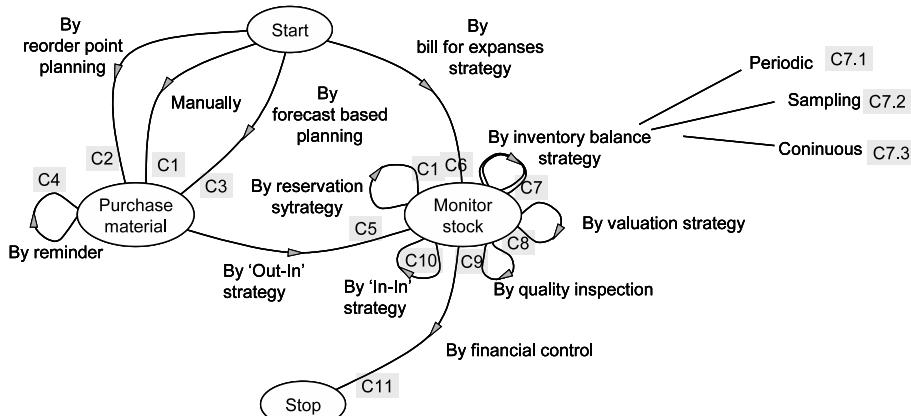
Fig. 1. Material Management Map (inspired from SAP Material Management module)

One advantage of the Map representation system is that it allows us to capture in a single representation what is common to the business and the system, namely their shared purpose. As a result, any section can be regarded from two viewpoints: the *business viewpoint* and the *system viewpoint*. For example, the map in Fig. 1 shows how the SAP Material Management (MM) module can be abstracted into sections of a map [17]. Every section in the map represents both (i) a SAP MM function, and (ii) the business goal that can be satisfied by using this function.

From the *business viewpoint*, material management deals with supplying materials in the right quantity, at the right place and time, and at the minimum cost. The map identifies through two intentions that this involves two tasks: to *Purchase material*, and to *Monitor stock*. It also makes explicit the different manners by which each intention can be achieved. For example there are four strategies to P*urchase material*, namely *Manually*, *By forecast based planning*, *By reorder point planning* and *By reminder*.

From the *system viewpoint*, the map indicates which SAP function helps to achieve the *Material purchase* and *Monitor Stock* goals, and how. For example, the SAP MM module contains a "Create purchase order" function (or 'transaction' in SAP terms). At the operational level, this function entails the identification of material requirements. The material requirements are defined by references to the needed materials, their vendors, their prices, the dates and plant at which they should be delivered, etc. At the business level, the issue is the one of purchasing material to satisfy material needs of the organization. The function contains variants depending of the purchase situation. These are referred to in the four sections C1, C2, C3, and C4, as documented in Table 1. For each of the four sections, the table outlines the variant of the SAP function that is to be used.

Our experience confirms what is illustrated in the SAP example above: the multi-thread topology of the map is useful to reason about alternative fitness relationships. The multi-thread

(a) makes explicit the different business strategies to achieve an intention and,
(b) identifies the variants of the system functionality that can be selected depending on the situation at hand thus highlighting the alternative (ORed) fitness relationships.

We found that when eliciting the desired state (To-Be model) the multi-thread helps envisioning multiple business strategies and identifying the corresponding required system functionality. In a customizing process the multi-thread helps exhibiting the panel of business strategies embedded in the product family and their related software variants.

Issue2: Modeling and measuring the relationship. Alignment deals with establishing relationships between two sets of entities to represent the business and the system, respectively. This relationship has been referred to by different terms such as "alignment", "fitness", "match", "adaptation", "correspondence". It reflects the system/context intertwining as highlighted in the framework put

Table 1. Documenting map sections

Code	Name	Description
C1	Purchase material	Create a purchase order based on a purchase requisition manually defined with information about the material, vendor, date, price, etc. If the information is correct the purchase order is created with a unique identification number.
C2	Purchase material based on reorder points	Automatically generate purchase requisitions any time a stock event that causes the stock of a given material to fit the reorder point criteria occurs. The purchase requisitions can then be transformed into purchase orders.
C3	Purchase material based on forecast	Automatically generate purchase requisitions at the dates defined in the forecast scheduling the purchases that shall be made for a given material. The forecast is computed based on former purchases of the material. Oncegenerated, the purchase requisitions can be transformed into purchase orders.
C4	Purchase material by reminder	Automatically remind of a purchase order for which no delivery has been noticed within due date.

forward in this book. In this paper we will use the term *fitness relationship* [1]. Despite the fact that the relationship is mentioned in many approaches, it is rarely a concept per se. Our view is that the fitness relationship should be precisely defined to support explicit modelling and measurement. We adopt Regev's view [18] that defines this concept "as the correspondence between a set of components". Following Nadler [19] we consider fitness to be "the degree to which the needs, demands, goals, objectives and/or structure of one component are consistent with the needs, demands, goals, objectives and/or structure of another component". The foregoing suggests (a) a precise identification of the types of correspondence between components of the business and system models and (b) a measurement of the degree of correspondence.

A survey of literature shows that different types of links have been proposed. *Traceability* links as used by [20] or [21] to reason about the impact of change, are one option leading to a *loose coupling* between entities. Another variation are *derivation links* such as in [22] who proposes a technique for deriving event-based specification from KAOS specifications. Similarly [23] identifies links between concepts of i* and those of Z in order to transform changes of an i* model into modifications of a Z specification. Derivation rules also used in GORE (Goal Oriented Requirements Engineering) approaches in RE correspond to a more *direct coupling*. Etien et al [24] define two types of correspondence links namely, *maps* and *represents* between constructs belonging to two different models. These are discussed below.

A construct X of a given model *maps* ($\mathcal{M}$) a construct Y of another model if there exists an isomorphism between the set of properties of X (p(x)) and the set of properties of Y, p(y). In other terms, each property of X corresponds to one of Y (even if the domains are different). The existence of an isomorphism

($\approx$ symbol) allows one to specify that the constructs X and Y play the same role. For example, the construct of business class maps the concept of system class.

$$X\mathcal{M}Y \Leftrightarrow p(x) \approx p(y)$$

This relationship is reflexive and thus if a construct X *maps* to another construct Y then, Y also *maps* X.

A construct X of a given model *represents* ($\Re$) a construct Y of another model if the existence of the former affects the behaviour, the value or the existence of the latter.

$$X\Re Y \Leftrightarrow p(x) \triangleright p(y)$$
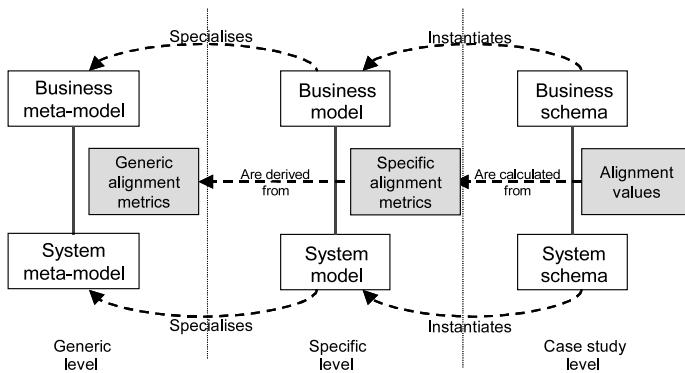
Where $\triangleright$ signifies that X affects the behaviour of Y.

It is important to notice that two constructs of different nature can be linked through the *represents* link. For example, a class can *represent* a property. Clearly, the **represents** link is weaker than the *maps* link. Thus, two constructs that *map* each other also *represent* each other. The opposite is not true because two constructs of different nature can be linked through a representation link even though a mapping link cannot exist.

A more advanced solution completes links with *metrics*. For instance, [25] proposes to define consistency using *metrics*. Soffer [13] suggests that identification of unfit requires the application of a fit measurement method. Etien [24] proposes fitness criteria and associated metrics to quantify the extent to which there is a fit between related business and system models. All possible *maps* and *represents* links (see above) constitute the baseline elements to measure the fit or unfit between a business model and a system model.

As shown in Fig.2, the process of generating the appropriate set of metrics for a given pair of models is based on the adaptation of a set of *generic metrics*. Metrics are based on the *maps* and *represents* correspondences established between constructs of the two meta-models representing the business and system *ontologies* respectively. Indeed, Etien et al base their proposal on the use of separate ontologies for representing business and system components respectively. For the former the ontology of Soffer and Wand [26] is used whereas at the system level the Wand and Weber ontology [27] is employed. The use of ontologies is a way of becoming independent of specific models, thereby leading to a generic expression of the component correspondence and associated metrics. There are a number of advantages of introducing the generic level, (1) generic metrics are based on the solid theoretical foundation provided Bunge's ontology (2) generic metrics serve as a guide to define the specific ones: the latter are just a specialisation of the former, (3) the process of producing the specific metrics is easier and less error prone and, (4) specific sets of alignment metrics are consistent with each other as they are generated from the same mould and this facilitates comparison across methods.

Table 2 sums up the fitness measurement framework. The table shows that the framework comprises ten criteria and ten metrics grouped along four factors. The framework follows the approach of Cavano and McCall [28] who use a multi-level organisation for their framework for software quality measures. They

Fig. 2. Fitness metrics generation

use a 3-level framework, the levels being, *factors*, *criteria* and *metrics*. Etien identifies four factors along which the fitness can be measured namely, the *intentional factor*, the *informational factor*, the *functional factor*, and the *dynamic factor*. These factors reflect the four perspectives that have been reported in IS literature, namely, the holistic view brought by the goal-actor-resource-process perspective; the information perspective; the functional perspective; and the dynamic perspective. They can be used to aid in specifying fit objectives. Each factor has associated *criteria* corresponding to characteristics of the fit. They are in turn, related to *metrics* that allow the actual computation of the degree of fit.

Finally, some authors advocate the use of a *common language*. This approach is recommended by Clarke to address the misalignment of design and code [20] and used by SysML [29] in order to understand relationships between different domain dependent models expressed in UML. This is the position we adopted

Table 2. Fitness measurement framework

Factors	Criteria	Metrics
Intentional Fit	Support Ratio	Activity representation count
	Goal Satisfaction	Goal mapping count
	Actor Presence	Actor mapping count
Informational Fit	Information Completeness	Business/System class mapping count
	Information Accuracy	Business/System state mapping count
Functional Fit	Activity Completeness	Business/System class mapping count
	Activity Accuracy	Business/System state mapping count
Dynamic Fit	System Reliability	Law mapping count
	Dynamic Realism	Path mapping count
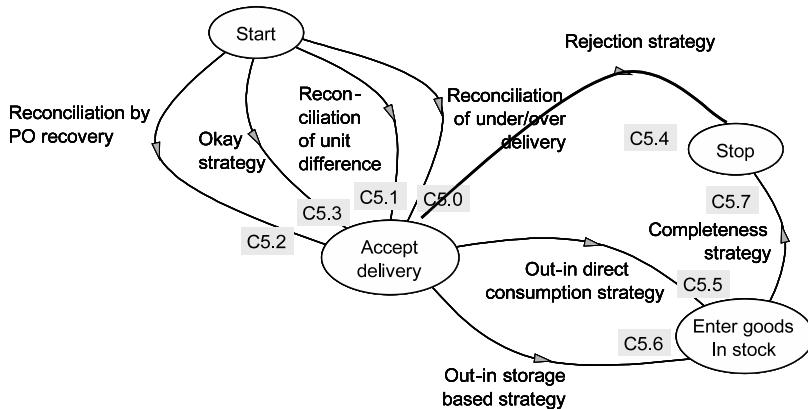
using the MAP formalism as the common language. The ideal fit between system and business occurs when there is a perfect match between the business map and the system map. As the world is not always ideal, we defined in [30] distance measures to evaluate the extent to which two sections of system and business maps respectively, deviate from one another. In dealing with strategic alignment we use a strategic map as the common description of the business and system purpose. We defined in [31] links from sections of the map to business strategy entities on one hand and system elements on the other. Links, such as "is necessary to", "is useful to", "is sufficient to", "is constrained by", "is contradictory with", have been postulated to support the measurement of fit and unfit.

Issue3: Dealing with the relationship complexity. It may be inconvenient to view the fitness relationship as one monolithic flat structure. A layered approach may facilitate understanding. Therefore, in order to present the fitness relationship at different levels of detail, it is necessary to have a *refinement* mechanism as well as a means to control the *quality* of the refinement. 'Recognise Complexity' is the fourth key issue which emerged from the Workshop and is presented as one of the four principles of the framework for this book. This principle evidently applies to the business/system fitness issue and we develop our view on it in the sequel.

Refinement is an abstraction mechanism by which a given entity is viewed as a set of inter-related entities. Refinement is known as a means to handle complexity. Our belief is that a such *refinement mechanism* is required for handling the fitness relationship in a systematic and controlled manner. Indeed, it would be inconvenient to view in one shot, a fitness relationship as one monolithic, flat structure. A layered approach may help mastering progressively the complexity of the relationship. This confirms our experiences which show that the refinement ratio (see Table3) is around 20, meaning that a relationship, initially seen as a whole, finally leads to a complex organization of about 20 sub-relationships.

As far as we are aware, there are very few attempts to provide a refinement mechanism of the fitness relationship [32]. In goal driven approaches, it is known that goals can be used to capture the objectives of a system at various levels of abstraction and goal decomposition is traditionally used to relate high level goals to low level, operationalisable goals. These are leaves of the goal graph that point out to the required functionalities of the system. One can therefore see that goal decomposition does not support top-down reasoning about the fitness relationship. Instead goal decomposition is a mechanism leading to the establishment of the fitness relationship as the link between system functionality and leaves of the goal graph.

The possibility of refining a goal graph was one of our motivations for defining MAP. In the map approach it is possible to *refine a section* of a map at level i into an entire map at a lower level $i+1$. Therefore, a fitness relationship (captured in a section of the map) is refined as a complex graph of sections, each of them corresponding to sub-relationships between the business and the system. Therefore, what is refined by the refinement mechanism offered by maps is in fact the fitness relationship itself. We found this mechanism helpful in

Fig. 3. Refined map M5 of section C5 of the SAP MM map shown in Fig. 1

understanding the fitness relationship at different levels of detail. Let us exemplify this mechanism by refining the section C5, *Monitor stock by Out-In strategy* of the SAP map presented in Fig. 1. The refined map of C5 is shown in Fig. 3. From the *business viewpoint*, this map explains how the *Monitor Stock by 'Out-In' strategy* is refined as a graph of intentions and associated strategies. The map tells us that stock entries shall not be permitted unless they have been checked. This is reflected in Fig. 3 by the ordering of the two intentions *Accept delivery* and *Enter goods in stock*. The map also shows that there are several ways to achieve each of these two intentions. For example, there are four strategies to *Accept delivery*: the *Okay* strategy (when the delivery matches the order) and three reconciliation strategies, namely the *Reconciliation by PO recovery*, the *Reconciliation of unit difference*, and the *Reconciliation of under/over delivery*. Each of these provides a way of accepting deliveries that don't match the order but are within specified tolerances, missing order reference, unit difference, under/over quantity, respectively. From the *system viewpoint*, this map explains how the *complex function C5* is made of other functions and the manner in which these functions cooperate to collectively achieve the intention, C5. The map shows that there are seven sub-functions C5.1 to C5.7 of the C5 function that shall co-operate as indicated by the multi-thread and multi-path topology of the M5 refined map.

Since refinement results in a map, it produces a multi-thread, multi-path structure at level *i+1*. As a result, for a given section at level *i*, not only (1) multiple threads describe *alternative sub-sections* at level *i+1*, but also (2) the multi-path structure introduces several different *combinations of sub-sections*. Therefore, section refinement is a more complex structure than a simple composition structure such as AND/OR goal decomposition Indeed, it provides at the same time (a) several alternative decompositions of the initial fitness relationship into its constituents, and (b) different alternatives to its constituents themselves. We found this mechanism useful in practice as a means to fine tune the selection of the adequate system sub-functions in a customizing process.

As an example, let us consider two companies who have selected the *Out-In strategy* of the SAP map to *Monitor stock*. The refined map M5 (Fig.3) helps each of them to refine their selection in different ways. For example, while one company may want to *Accept delivery* by *reconciliation of over/under delivery*, the other may be driven by the *reconciliation by purchase order recovery* strategy. Similarly, one company may want to *Enter goods in stock* using the *Out-in direct consumption* strategy that allows one to consume goods even though they are not yet entered in stock, whereas the other company may select only the *Out in storage* strategy, i.e. systematically enter goods in stock before they are consumed. Therefore, while the two companies have the same fitness relationship at the level of abstraction of the MM map, the refinement mechanism allows us to differentiate the sub-relationships relevant for each company.

If refinement is necessary as a way to master complexity, it also generates its own difficulties. One is to control the *level of abstraction* in a given refinement. This has been found for example, in using refinement in use case driven approaches [33]. Cockburn reported [34] that the application of his refinement mechanism (by which an action in a scenario can be seen as a goal attached to a new use case and associated scenarios) led to mix up of abstraction levels in the same scenario. It seems that several levels of abstraction are a source of difficulty, in particular with respect to consistency of the entities belonging to the same level. This issue is related to the *black box/white box* principle [33] [35] [36]. According to this principle, it is useful to see an entity as a black box at a level of abstraction i and then as a white box at level $i+1$. When a system is seen as a black box, its internal properties are hidden and the emphasis is on the relationship between the system and other systems. When it is seen as a white box, the internals of the system are, on the contrary, apparent. The problem arises when the white box analysis shows that the content of the box covers different levels of abstraction.

We encountered this problem when applying the map refinement mechanism in projects, and we believe that complementing the *refinement mechanism* of the fitness relationship with *refinement quality assurance* is an issue.

In a project with the Renault company [37], we defined a set of *refinement quality* rules and gained experience in their application. The aim was to ensure a unique level of abstraction in a given level of map refinement. Given a map at a refinement level i the rules helped, (a) to detect and move up sections into maps at level i-1; (b) to detect and move down sections into maps at level i+1 and, (c) to improve sections at the same level.

In order to reinforce the importance of the *refinement quality checking* issue, we present in Table 3 data gathered from four projects in industry [17] [38] [39] [40]. The table shows that the top map has a limited number of goals and strategies. The table also reflects the fact that systematic section refinement could rapidly lead to a combinatorial explosion of the number of maps to document. There is therefore, a need to control the refinement. It was also found necessary to identify when the refinement is needed and when it is not. In the DIAC project for example, we achieved the latter through a consensus based

Table 3. Practice data
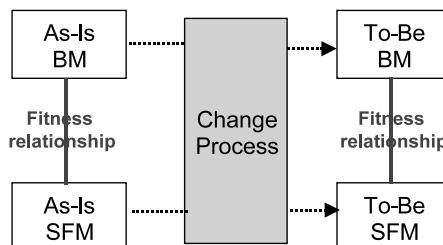
	Goals in top level map	Sections in top level map	Refinement levels	Total number of maps	Number of transactions or screens
PPC	2	6	3	37	200 transactions
DIAC	3	14	3	36	2000 screens
SAP MM	2	11	2	14	About 50
SNCF	4	27	5	55	300 transactions

process: each section was subject to a vote and the refinement was considered unnecessary when the fitness between the business requirement and the selected product was agreed upon by the stakeholders.

3   Issues in Engineering the Relationship

Whereas the three previous issues were dealing with understanding the fitness relationship we are now moving to the issues related to its production, particularly in an evolutionary perspective. The broad framework used in this paper is in Fig. 4. It accepts the prevalent view of change as a move from the *As-Is* to the *To-Be* situation [41]. However, it departs from the traditional view in highlighting the fitness relationship itself (linking the Business Model (BM) and the System Functionality Model (SFM)) and its engineering through the change process.

Our experience in a wide range of change handling projects shows that change arises in a number of different ways. Each of these can be characterized by a specialization of the framework of Fig. 4 that influences the way of preserving the fitness relationship. The first issue in engineering the fitness relationship, namely **Issue 4**, is the **identification of the different change engineering classes**. Each class has its own engineering process. Despite the diversity of processes, we found some common underlying strategies that we discuss as **Issue 5:change engineering strategies**. Assuming that the change requirements have been stated the next question is that of propagation of these change requirements to preserve the fitness relationship. This is raised through **Issue 6: preserving the**

Fig. 4. The broad change framework

fitness relationship. These four issues evidently relate to the second principle 'Evolve Designs and Ecologies' pointed out in the framework that constitutes the core position of this book.
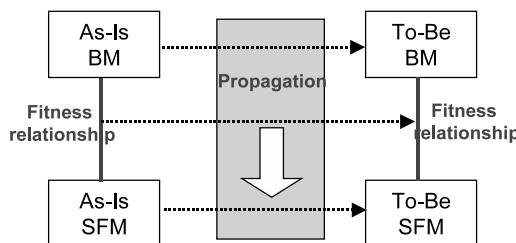
Issue 4: Identifying different change engineering classes. As mentioned in the introduction of section 3, our experience showed that there are different change situations leading to specific ways to engineer the fitness relationship. Our suggestion is to *classify* these change situations and change processes as a means to better understand the different engineering ways to produce the fitness relationship. We propose four engineering classes

- Direct change propagation;
- Customization from product family;
- Adaptation of a baseline product and
- Component assembly.
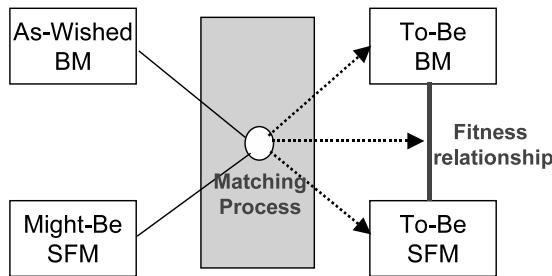
Direct change propagation. Fig. 5 is the customized version of the generic change framework presented in Fig.3. It corresponds to the case of *direct change propagation*. In this situation, the change is led by the move from the *As-Is BM* to the *To-Be BM*. The relationship between the *As-Is BM* and the *As-Is SFM* is used to propagate the business changes onto system functionality changes and to produce the *To-Be SFM* fitting the *To-Be BM*.

This first engineering class corresponds to the traditional evolutionary change of an 'in house' system that is driven by organisational change. However today, with an increase of the 'to-buy' policy over the 'to-do', the fitness relationship is established through rather more complex engineering processes shown in Figures 6, 7, and 8 respectively.

Customization from product family. Fig. 6 displays the case of *customising a product family*. Here, the requirements of the organisation are expressed in the *As-Wished BM*. The *Might-Be SFM* (Might-Be System Functionality Model) reflects the functional capability of the product family. The *To-Be BM* and its counterpart, the *To-Be SFM* result from a *model-match centred process* which searches for the best match between the organisational requirements (expressed in *As-Wished BM*) and what is provided by the Product Family, (*Might-Be SFM*).
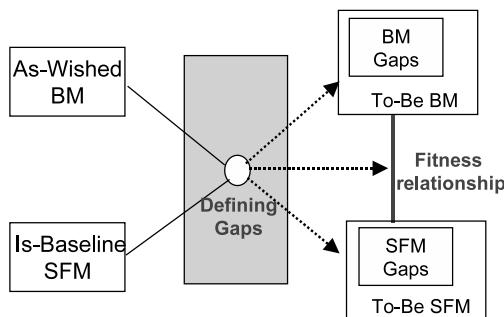
Fig. 5. Direct change propagation
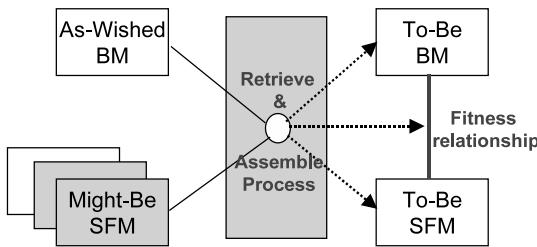
Fig. 6. Customization from product family

Adaptation of a baseline product. The third engineering class shown in Fig. 7 corresponds to a case of *system adaptation*. Such an adaptation is caused by changes in the organizational context in which a legacy software system is now to operate. This typically occurs because of mergers/take-overs, globalisation, standardisation of practices across branches of a company etc. Several legacy software systems are already running when such events occur. In this context, the question is not to develop a new system from scratch. Rather, it should be possible to integrate the legacy systems or to select one of these for adaptation and uniform deployment across the organization. The *Is-Baseline FM* (Is-Baseline Functionality Model) models the functionality of the selected system for uniform deployment across the organization. The *As-Wished BM* expresses the requirements of the organization that wants to adapt its requirements to the new situation at hand. The *To-Be BM* and its counterpart the *To-Be SFM* result from a gap *centred process* that focuses on eliciting the gaps between models. Indeed, eliciting the differences (or gaps) between the current baseline-functionality model and its future version seems to be the most efficient way to perform the change.

COTS-based system development. Finally, Fig. 8 displays the case of *COTS based system development*. Here the process of establishing the fitness relationship is centred on the retrieval and assembly of system components that match

Fig. 7. Adaptation of a baseline product

Fig. 8. Component assembly

organizational requirements. The *Might-Be SFM* models the different COTS components available. The *As-Wished BM* reflects the organizational needs. The *To-Be BM* and its counterpart the *To-Be SFM* result from a process where components matching the organization needs are retrieved and then assembled together.

We show in the following that these different classes call for different types of engineering techniques.

Issue 5: Change engineering strategies. Despite the diversity of approaches dealing with each of the four aforementioned classes, reflection from experiences and literature surveys led us to identify three common underlying strategies.

The first of these, which we call *'from scratch'* strategy, is prominently used in the first engineering class, namely 'direct change propagation'. It does not make *change requirements* (i.e. requirements for change) explicit but lets them remain *implicit* i.e. change requirements are not formulated per se. Instead, the focus is (i) on formulating the To-Be requirements or (ii) on writing a new release of the requirements document. As evidence, consider methodologies such as Merise, i*, Kaos or the RUP that do not require an explicit description of changes of current models. They help to construct new ones focussing on the To-Be situation. Type (i) reflects an ad-hoc practice inherited from the way legacy systems have been maintained over time. In (ii) type of approaches the concern on the change transition leads to keep track of requirements that have been removed, added, changed using a configuration mechanism of requirements documents. The preservation of the fitness relationship is not considered as such but assimilated to its establishment from scratch as it was to be freshly established.
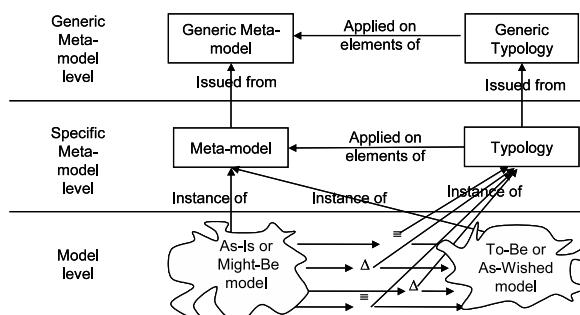
The second and third strategies emerged in order to address the three other engineering classes of figures 6, 7, and 8. On one side of the spectrum we found that *similarities* between pairs of models are useful to support the matching process whereas at the other end of the spectrum focusing on differences or *gaps* is the most relevant technique. For example, 'customising' and 'component assembly processes' call for a similarity based technique whereas a gap measurement technique is more suitable for processes of the type 'adaptation from a baseline product'. Similarity modeling and gap modeling are not easy tasks if they are not achieved in a systematic way. The challenge is to maximize the knowledge gained, while limiting the amount of effort needed to gain it.

Different gap and similarity based languages have been defined in different domains and could be adapted to the fitness context; for example, similarity formulae to reuse source code [42], or find resemblances among object oriented models [43] or UML models [44]; languages for expressing requirements of database schema evolution [45], [46], and [47] or workflow models evolution [48], [49], or [50]. We proposed in [39] & [51] a generic typology of gap operators and similarity predicates that can be specialised for any modelling formalism. Once specialised for the MAP formalism, our language can be used to express for example, how goals and strategies shared by the business and the system shall change, e.g. a goal can have a change of name, a strategy can be replaced by another one, sections can be merged or split, etc. Combined with maps of the current situation, change requirements expressed as gaps can be used to automatically control the consistency of the requirements actually released for the future system under the form of goal/strategy maps.
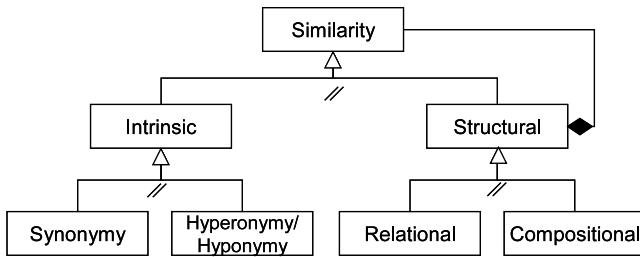
The overall approach is depicted in Figure 9. As the figure shows, gaps (represented by the symbol Δ) and similarities (represented by the symbol $\equiv$) are specified at the model level. These models conform to different meta models such as Use Case, E/R, Workflow, Business Process, Goal graphs, etc. that are themselves viewed as instances of a generic meta-model (or meta-meta model). This hierarchical framework allows us to define generic gap operators and similarity predicates at the generic meta level. These typologies and measures are in turn, used to evaluate gaps and similarities at the model level.

The generic meta-model aims to make explicit the main elements and their relationships of any model [39].It indicates that any model is made of *elements* characterised by *properties*. The elements are specialised depending on whether they are *link* or *non link* on one hand, and on whether they are *atomic* or *compound*, on the other hand.

Based on the generic meta-model, the *generic typology of similarity predicates* emphasises that given a pair of elements, (i) their properties can be similar, and (ii) their structure can be similar. We therefore, identified two classes of similarities, namely intrinsic similarities and structural similarities (Fig.10). A pair of elements has an *intrinsic* similarity if the elements have similar properties

Fig. 9. Overview of the approach for defining gap & similarity typologies

Fig. 10. Generic typology of similarity predicates (main categories)

and therefore, have close semantics. We related intrinsic similarity to synonymy and hyponymy/hyperonymy. *Structural* similarity deals with (i) the composition of elements (Compositional similarity), and (ii) their organisation within structures (relational similarity). Whereas intrinsic similarity only involves the two compared elements, structural similarities also imply comparisons between other elements that are related to the two compared ones. As shown in Fig. 10 by the aggregation link from the structural similarity class to the similarity class, a structural similarity may be complex and may involve other similarities. The generic typology includes thirty-three similarity predicates [51] organised according to the similarity types shown in Fig.10.
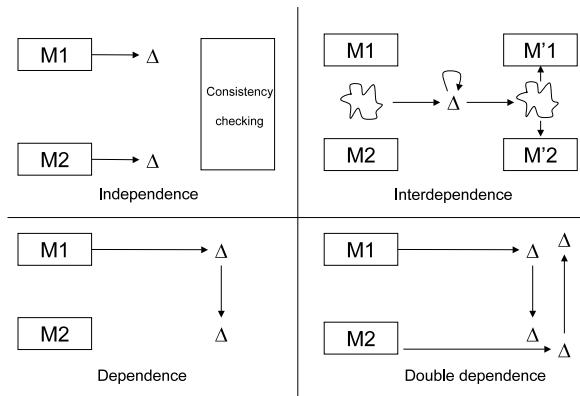
Again, the generic meta-model helps us to identify a *generic typology of gaps* composed of a set of *operators* applicable to model elements. Each operator identifies a type of change that can be performed on an As-Is model. The operator identifies the difference between the As-Is model and the To-Be model. For example, as *Rename* is an operator (see Table 4), *Rename Element* will be a change that characterizes the transformation of an As-Is element in the To-Be model. The generic gap typology identifies three major types of change: *naming* changes, *element* changes and *structural* changes.

- *Naming changes* are defined with the *Rename* operator.
- *Element changes* affect elements and are circumscribed to the elements themselves, i.e. adding an *attribute* to an *entity type* is an example of such localized change.
- *Structural changes* are the most important as they correspond to a modification of the set of elements which composes the model. For example adding or removing *Relationship types* and *Entity types* in an *As-Is* E/R schema to form the *To-Be* schema is a structural change.

Issue 6 : Preserving the fitness relationship. Once change requirements have been identified, for instance through a set of change requirements expressed as gaps, the question raised through issue 6 is that of propagation of these change requirements taking into account that several entities linked through the fitness relationship have to evolve at the same time. This is sometimes referred to as co-evolution by analogy with biology and aims to study the reciprocal evolution

Table 4. The generic gap typology.

Object	Operator	Description
Element	Rename	Change the name of the element in the To-Be model
	Add, Remove	Add/Remove an element of the As-Is in the To-Be model
	Merge	Two separate As-Is elements become one in the To-Be model
	Split	One As-Is element decomposes into two To-Be elements
	Replace	An As-Is element is replaced by a different To-Be one
Link	Change	The source or target of the link is changed
Compound	AddComponent	A component is added in the To-Be element
	RemoveComponent	An As-Is component is removed in the To-Be element
	MoveComponent	A component is repositioned in the structure of the To-Be element
Property	Give	Add a property to the To-Be element
	Withdraw	Remove an As-Is property in the To-Be element
	Modify	Change the property of the To-Be element
	Retype	The As-Is element changes its type in the To-Be

Fig. 11. Co-evolution classes

of systems and other entities such as organisations [52], business processes [37], or environment [53]. Fig. 11 proposes four classes of co-evolution engineering to preserve the fitness relationship, namely *independence*, *interdependence*, *dependence*, and *double dependence*. Each class reflects a different ordering to handle the co-evolution of involved entities.

Co-evolution is engineered *independently* (left upper corner of Fig. 11), when there is no dependency between the change engineering processes of each evolving

entity. Independent co-evolution implies the need for checking alignment after the changes have been made.

In a *dependent* approach to propagation of change requirements, the change requirements of one entity are inferred from the change requirements elicited for the co-evolving entity. Changing the software system as a consequence of organisational changes is a typical example of this class. Similarly, when dealing with a portfolio of projects, managers select first 'flagship projects' and change projects dependent on these accordingly. Another way is to maintain the alignment using a set of rules which express the dependence between the two entities [23].

There is a *double-dependence* when each co-evolving entity can play the role of master in the propagation of change requirements. For example, [54] proposes rules to evaluate the impact of change requirements specified at the business level on the system-level and vice-versa. A double dependence approach can be considered as the combination of two one-way dependence approaches.

An *interdependent* approach is more balanced. Each change requirement specifies how all co-evolving entities shall evolve simultaneously. Then, propagation is performed with a single collection of change requirements. The gap approach that we developed using maps as a means to represent both business and system belongs to this category. Change requirements are expressed as gaps and are propagated simultaneously on business and system models [55].

4   Conclusion

Our experience of Requirements Engineering and Information System Engineering has been obtained over a number of industrial and European research projects. The common point we found in all these projects was the prime importance of establishing and preserving the fitness relationship between businesses and systems. This position is conforming the 'Intertwine Requirements and Contexts' set as one of the key issues in Requirements Engineering that emerged from the workshop. A number of issues directly derive from this point. However, the prevalent view is that fit is achieved through a process that is often poorly specified. We depart from this traditional view by proposing to specify the fitness relationship itself and to engineer it through the change process. This entails a number of issues that were reported and discussed in this paper.

These issues reflect our beliefs on how to address the fitness problem. We formally define the fitness relationship through two kinds of links, *maps* and *represents*. We suggest that resolution of the fitness problem is facilitated by using a common conceptual language to represent both the business and the system. Thereafter we propose that the multiplicity of change engineering situations can be dealt with by three basic strategies namely, *from scratch*, *similarity* and *gaps*, respectively. To get a quantitative estimate of the extent of fit, the paper proposes the generic metric approach.

Taken together, it can be seen that we have given centre place to the notion of the fitness relationship. By doing so, we have highlighted the need for more

investigation into the several points of view from which the fitness problem can be looked upon.

References

1. Potts, C.: Fitness for use: The system quality that matters most. In: 3rd International Workshop on Requirements Engineering: Foundation for Software Quality (1997)
2. Luftman, J., Maclean, E.R.: Key issues for it executives. MIS Quarterly Executive 3, 89–104 (2004)
3. Reich, B.H., Nelson, K.M.: In their own words: Cio visions about the future of in-house it organizations. The DATA BASE for Advances in ISs 34, 28–44 (2003)
4. Tallon, P., Kraemer, K.L.: Executives Perspectives on IT: Unraveling the Link between Business Strategy, Management Practices and IT Business Value (2002)
5. Watson, R.T., Kelly, G.G., Galliers, R.D., Brancheau, J.: Key issues in iss management: an int. perspective. Journal of Management ISs 13, 91–115 (1997)
6. Brancheau, J.C., Janz, B.D., Wetherbe, J.C.: Key issues in information systems management: 1994-1995 sim delphi results. MIS Quarterly 20(2), 225–242 (1996)
7. Chan, Y.E., Huff, S.L., Copeland, D.G., Barclay, D.W.: Business strategic orientation, ISs strategic orientation and strategic alignment. ISs Research 8(2), 125–150 (1997)
8. Kefi, H., Kalika, M.: Survey of strategic alignment impacts on organizational performance in international european companies. In: 38th Hawaii International Conference on System Sciences. IEEE Computer Society, Los Alamitos (2005)
9. Chan, Y.E.: Why haven't we mastered alignment? the importance of the informal organization structure. MIS Quarterly Executive 1(2), 97–112 (2002)
10. Sabherwal, R., Chan, Y.E.: Alignment between business and is strategies: A study of prospectors, analyzers and defenders. ISs Research 12(1), 11–33 (2001)
11. Croteau, A.M., Bergeron, F.: An information technology trilogy: business strategy, technological deployment and organizational performance. Journal of Strategic IS 10, 77–99 (2001)
12. Hirschheim, R., Sabherwal, R.: Detours in the Path toward Strategic Information Systems Alignment. California Management Review 44(1), 87–108
13. Soffer, P.: Fit measurement: How to distinguish between fit and misfit. In: Persson, A., Stirna, J. (eds.) CAiSE 2004. LNCS, vol. 3084, pp. 253–254. Springer, Heidelberg (2004)
14. Arsanjani, A., Alpigini, J.: Using grammar-oriented object design to seamlessly map business models to component-based software architectures. In: Symposium of Modelling and Simulation, pp. 186–191 (2001)
15. Alves, C., Finkelstein, A.: Challenges in cots decision-making: a goal-driven requirements engineering perspective. In: Workshop on Software Engineering Decision Support in conjunction with SEKE 2002, pp. 789–794 (2002)
16. Rolland, C.: Capturing intentionality with maps. In: Krogstie, J., Opdahl, A.L., Brinkkemper, S. (eds.) Conceptual modelling in information system engineering. Springer, Heidelberg (2007)
17. Rolland, C., Prakash, N.: Matching erp system functionality to customer requirements. In: 5th IEEE International Symposium on Requirements Engineering, pp. 66–75. IEEE Computer Society, Los Alamitos (2001)

18. Regev, G., Wegmann, A.: Remaining fit: On the creation and maintenance of fit. In: Persson, A., Stirna, J. (eds.) CAiSE 2004. LNCS, vol. 3084, pp. 131–137. Springer, Heidelberg (2004)
19. Nadler, D., Tushman, M.L.: A congruence Model for Diagnosing Organizational Behavior. In: Miles, R. (ed.) Resource Book in Macro Organizational Bahavior, pp. 30–49 (1980)
20. Clarke, S., Harrison, W., Ossher, H., Tarr, P.: Subject-oriented design: Towards improved alignment of requirements, design and code. In: Object-Oriented Programming, Systems, Languages and Applications (1999)
21. Pohl, K., Jacobs, S.: Oncurrent engineering: Enabling traceability and mutual understanding. Journal of Concurrent Engineering Research and Application, Special Issue on Concurrent Engineering and Artificial Intelligence 2(4), 279–290 (1994)
22. De Landtsheer, R., Letier, E., van Lamsweerde, A.: Deriving tabular event-based specifications from goal-oriented requirements models. In: 11th IEEE International Conference on Requirements Engineering, p. 200. IEEE Computer Society, Los Alamitos (2003)
23. Krishna, A., Ghose, A.K., Vilkomir, S.A.: Co-evolution of complementary formal and informal requirements. In: 7th International Workshop on Principles of Software Evolution in conjonction with IWPSE 2004, pp. 159–164. IEEE Computer Society, Los Alamitos (2004)
24. Etien, A., Rolland, C.: Measuring the fitness relationship. Requirements Engineering Journal 10(3), 184–197 (2005)
25. Bodhuin, T., Esposito, R., Pacelli, C., Tortorella, M.: Impact analysis for supporting the co-evolution of business processes and supporting software systems. In: BPMDS Workshop in connection with The 16th Conference on Advanced Information Systems Engineering, pp. 146–150 (2004)
26. Soffer, P., Wand, Y.: Goal-driven analysis of process model validity. In: Persson, A., Stirna, J. (eds.) CAiSE 2004. LNCS, vol. 3084, pp. 521–535. Springer, Heidelberg (2004)
27. Wand, Y., Weber, R.: An ontological model of an information system. IEEE Transactions on Software Engineering 16(11), 1282–1292 (1990)
28. Cavano, J., McCall, J.: A framework for the measurement of software quality. In: Software Quality and Assurance Workshop, ACM SIGMETRICS and SIG-SOFT, pp. 133–139 (1978)
29. Sysml, http://www.sysml.org
30. Etien, A., Rolland, C.: A process for generating fitness measures. In: Pastor, Ó., Falcão e Cunha, J. (eds.) CAiSE 2005. LNCS, vol. 3520, pp. 277–292. Springer, Heidelberg (2005)
31. Thevenet, L.H., Salinesi, C.: Aligning is to organization's strategy: The InStAl-Method. In: Krogstie, J., Opdahl, A.L., Sindre, G. (eds.) CAiSE 2007 and WES 2007. LNCS, vol. 4495, pp. 203–217. Springer, Heidelberg (2007)
32. Potts, C., Hsi, I.: Abstraction and context in requirements engineering: Toward a synthesis. Annals of Software Engineering 3, 23–61 (1997)
33. Weidenhaupt, K., Pohl, K., Jarke, M., Haumer, P.: CREWS Team. Scenario Usage in System Development: a Report on Current Practice. In: 3rd International Conference on Requirements Engineering (1998)
34. Cockburn, A.: Writing Effective Use Cases. Addison-Wesley, Reading (2000)

35. Ben Achour, C.: Extraction des Besoins par Analyse de Scénarios Textuels. Phd Thesis, Univ. Paris 6 - Pierre et Marie Curie (1999)
36. Kamsties, E., von Knethen, A., Reussner, R.: A Controlled Experiment on the Understandability of Different Requirements Specifications Styles. In: 8th International Workshop on Requirements Engineering: Foundation for Software Quality (2002)
37. Salinesi, C., Rolland, C.: Fitting business models to system functionality exploring the fitness relationship. In: Eder, J., Missikoff, M. (eds.) CAiSE 2003. LNCS, vol. 2681, pp. 647–664. Springer, Heidelberg (2003)
38. Rolland, C., Grosz, G., Kla, R.: Experience with goal-scenario coupling in requirements engineering. In: 4th IEEE International Symposium on Requirements Engineering, p. 74. IEEE Computer Society, Los Alamitos (1999)
39. Rolland, C., Salinesi, C., Etien, A.: Eliciting gaps in requirements change. Requirement Engineering Journal 9(1), 1–15 (2004)
40. Rolland, C.: Aligning Business and System Functionality Through Model Matching, Systèmes dInformation et Management (SIM). Editions ESKA (2006)
41. Jackson, M.: Software Requirements and Specifications. Addison-Wesley, Reading (1995)
42. Prechelt, L., Mahpohl, G., Phlippsen, M.: Jplag. finding plagiarisms among a set of programs. Technical Report 2000-1, Universität Karlsruhe (2000)
43. Sarireta, A., Vaucher, J.: Similarity measure in the object model. In: Aksit, M., Matsuoka, S. (eds.) ECOOP 1997. LNCS, vol. 1241. Springer, Heidelberg (1997)
44. Blok, M.C., Cybulski, J.L.: Reusing uml specifications in a constrained application domain. In: 5th Asia-Pacific Software Engineering Conference, pp. 196–202. IEEE Computer Society, Los Alamitos (1998)
45. Delgado, C., Samos, J., Torres, M.: Primitive operations for schema evolution in ODMG databases. In: Konstantas, D., Léonard, M., Pigneur, Y., Patel, S. (eds.) OOIS 2003. LNCS, vol. 2817, pp. 226–237. Springer, Heidelberg (2003)
46. Lautemann, S.E.: Schema versions in object-oriented database systems. In: 5th International Conference on Database Systems for Advanced Applications, pp. 323–332. World Scientific, Singapore (1997)
47. Banerjee, J., Kim, W., Kim, H.J., Korth, H.F.: Semantics and implementation of schema evolution in object-oriented databases. In: Association for Computing Machinery Special Interest Group on Management of Data 1987 Annual Conference, pp. 311–322. ACM Press, New York (1987)
48. Casati, F., Ceri, S., Pernici, B., Pozzi, G.: Workflow evolution. In: Thalheim, B. (ed.) ER 1996. LNCS, vol. 1157, pp. 438–455. Springer, Heidelberg (1996)
49. Kradolfer, M.: A Workflow Metamodel Supporting Dynamic, Reuse-based Model Evolution. Ph.d. thesis, Department of Information Technology, University of Zurich (2000)
50. Reichert, M., Rinderle, S., Dadam, P.: A Formal Framework For Workflow Type And Instance Changes Under Correctness Constraints (2003)
51. Salinesi, C., Etien, A., Zoukar, I.: A systematic approach to express is evolution requirements using gap modelling and similarity modelling techniques. In: Persson, A., Stirna, J. (eds.) CAiSE 2004. LNCS, vol. 3084, pp. 338–352. Springer, Heidelberg (2004)
52. Lehman, M.M., Ramil, J.F., Kahen, G.: Evolution as a noun and evolution as a verb. In: Workshop on Software and Organisation Co-evolution (2000)

53. Mitleton-Kelly, E., Papaefthimiou, M.C.: Co-evolution & an enabling infrastructure: a solution to legacy? In: Systems engineering for business process change. Springer, Heidelberg (2000)
54. Kardasis, P., Loucopoulos, P.: Aligning legacy information systems to business processes. In: Pernici, B., Thanos, C. (eds.) CAiSE 1998. LNCS, vol. 1413, pp. 25–39. Springer, Heidelberg (1998)
55. Etien, A., Rolland, C., Salinesi, C.: Measuring the business/system alignment. In: Requirements Engineering for Business Need and IT Alignment (2005)

A Framework for Business Process Change Requirements Analysis

Varun Grover and Samuel Otim

Department of Management, 101 Sirrine Hall
Clemson University, Clemson, SC 29634
vgrover@clemson.edu

Abstract. The ability to quickly and continually adapt business processes to accommodate evolving requirements and opportunities is critical for success in competitive environments. Without appropriate linkage between redesign decisions and strategic inputs, identifying processes that need to be modified will be difficult. In this paper, we draw attention to the analysis of business process change requirements in support of process change initiatives. Business process redesign is a multifaceted phenomenon involving processes, organizational structure, management systems, human resource architecture, and many other aspects of organizational life. To be successful, the business process initiative should focus not only on identifying the processes to be redesigned, but also pay attention to various enablers of change. Above all, a framework is just a blueprint; management must lead change. We hope our modest contribution will draw attention to the broader framing of requirements for business process change.

Keywords: Business process change, transformation, redesign, requirements analysis, functional coupling, process synthesis, process decomposition.

1 Introduction

The pace of change in the new digital economy puts demands on organizations to keep reinventing themselves in order to remain competitive. To be competitive, businesses must respond better and quicker. Recognizing this, many companies have made significant attempts to improve their business processes. In doing so, organizations have evolved from traditional functional hierarchies to business process-centered structures [1]. This is because business processes are crucial as organizations adapt themselves to changing conditions. A business process, such as a loan appraisal system used in a bank, performs a specific business function by transforming input data to a form that is useful for decision making, thereby enabling an organization to achieve a business goal [2]. The continued interest in business processes can be attributed to the realization that when properly designed, they can help an organization achieve efficiency and effectiveness in its business operations.

Business process change management is the way to approach the transformation of traditional bureaucratic organizations into market-oriented process organizations.

Management strategies, such as business process reengineering (BPR), process improvement, process innovation, and business process redesign have emerged to help organizations change their business processes promptly and dynamically according to changing environments [3].

Despite the continued interest in business process change, it should be pointed out that the scheme started on a wrong footing and roughly 70 percent of the initiatives through the 1990s failed [4]. As, a result, the surge in business process reengineering (BPR) initiatives in the early to mid 1990s was followed by fall in the late 1990s and early 2000s [5]. Several factors contributed to the alarmingly high rate of failure of BPR efforts that discouraged further initiatives [6]. First, the idea of "radical" change dominated the earlier years. However, many organizations were focused on cost-saving and undertook massive restructuring and downsizing under the guise of process reengineering. These initiatives tried to optimize how work is done at the cost of people and did not suit organizational norms and culture; thus, they created more problems than they solved. Little attention was paid to a more tampered approach to process change in which change is introduced incrementally.

Second, while many companies stressed process reengineering, they neglected the necessary complementary changes in other organizational factors [7]. Complex process change decisions affect different interacting and interrelated dimensions of an organization: its processes, people, strategy, culture, information policies, etc. Thus, questions of strategy, structure, information technology (IT), incentive systems, people, roles, etc., have to be concurrently considered and deliberately aligned for process change success. Earlier initiatives that adopted a narrow view of process change resulted in processes that were not reinforced by other organizational factors and created tension that resulted in failure.

Third, change management was under-emphasized [8]. Little attention was paid to the institutionalization process because often, a hired consultant guided process change initiatives with a hand-picked team that did not communicate effectively with the rest of the organization. Due to poor process change management, "buy-in" from the rest of the organizational members was poor and the downsizing image associated with many initiatives created fear of job loss, which engendered resistance to change [1]. Appropriate process change management ought to embrace the institutionalization of these changes, as well as undertake continuous assessment of processes and their performance, and their impacts during the institutionalization process.

Fourth, while requirements elicitation is a vital phase of process change, overemphasis on existing processes often led organizations down a path of excessive documentation of processes that did not work [7]. As a result, several initiatives got entangled in the details and ultimately organizations lost sight of the goal. This, in turn, led to project scope creep due to too many vested interests deliberating on existing processes, which culminated in the inevitable project expansion and ultimate failure.

Fifth, IT was being thrown at process problems, with the hope that they would disappear. Instead of making processes the center of change, IT (e.g., ERP) was at the center of most initiatives without adequate consideration for organizational adaptations [8]. Despite an ongoing discussion about aligning business and IT goals, process change received lip service and buy-in by organizational members was not adequately secured [1]. In fact, in one study of more than 100 process change

projects, IT issues were considered most important, but had the lowest correlation with success. On the other hand, people and change management issues were considered the least important and had the highest correlation with success [9].

Sixth, earlier BPR efforts were focused on the redesign of operational processes; with limited value potential as efficiency was the major driver of change. This emphasis on back-office processes paid little attention to front-office and customer-facing processes, which play an important role in creating and sustaining competitive advantage [10, 11].

While even the major supporters of BPR such as Tom Davenport had declared it dead and over [12], the original narrow concept of BPR has been recast to take a broader perspective. This broader process management perspective embraces a continuum of approaches to process transformation (process reengineering, redesign, change, transformation, etc.), with more focus on the incremental approaches and value-creating processes [3]. With this broadened view, trend analysis by [5] on the BPR phenomenon indicates signs of revival after a dip in the late 1990s to early 2000s. Business process improvement is once again a top business priority [13].

Despite the broadened perspective on business process change, few comprehensive frameworks exist to guide practice in successfully managing the multifaceted nature of organizational change engendered by transforming business processes [14]. Moreover, some frameworks adopt a narrow view on business processes, focusing either on physical flows [15], information flows [16], or people's roles and relationships [17]. Gavin [18] develops a more comprehensive view that encompasses three types of organizational processes – business (work) processes, behavioral processes (decision-making processes, communication processes, and organizational learning processes), and change processes. He argued that the way business processes are performed is shaped by behavioral processes. Behavioral processes should therefore be taken into account for successful business process improvement. Earl and Khan [19] differentiated between dissimilar types of processes based on value-chain concepts (core processes, support process, and management processes). *Given the diversity of views, we propose a holistic framework that integrates the various views, with specific attention to the design and management aspects of business process change.*

Since business process change is a complex undertaking, a design orientation helps to manage this complexity in order to achieve successful adaption of processes to changing organizational conditions and market environment. We argue that process-based organizational analysis for the purpose of business change is primarily a design problem. According to information processing [20] and decision making [21] views of organizational design, processes can be viewed as collections of decision models, each of which is identified by a type of decision which contains a sequence of information processing tasks [22]. Tasks are the smallest identifiable unit of analysis and the critical design variable determining the efficiency of the resulting structures is the optimum arrangement of tasks [23].

However, detailed requirements gathering at the task level can take many months, or even years. This is undesirable because the pace of change requires delivering benefits immediately by aligning outcomes with a series of intermediate business needs, each contributing to the overall business strategy. The challenge is then to deliver the known requirements while still leaving open the ability to deliver the future requirements as the market unfolds. Put more concisely, the challenge is how to

minimize up-front business requirements but maximize future options. Often, there is really no right or wrong solution when processes are redesigned, rather there exists a wide range of potentially well-suited redesign strategies and more than one strategy may need to be implemented to respond adroitly.

Therefore, trying to achieve completeness, consistency, and correctness by working at the task level is insufficient to address business challenges in a rapidly changing environment. To circumvent these limitations, we propose a higher-order framework that focuses on the process of change and the type of decisions that have to be made. This is the input-process-output (IPO) framework. We feel that such a broad-based framework is more useful to senior managers in the organization in order to develop a high-level strategic perspective on the multifaceted business process change phenomenon at both the business process level and overall organizational level. Given the complexity of business process redesign, such a framework may help guide the strategic perspective of managers. While broad in nature, the IPO framework proposed here is sufficiently rich since it highlights how the various functional activities involved in a business process may be reconfigured through a process transformation initiative. To reflect the broadened view of BPR, we use business process *change* throughout, instead of business process *reengineering*.

The next section discusses the logic of business process change, followed by a presentation of the IPO framework we propose to guide business process change analysis and design efforts. The final section provides some concluding comments.

2   The Logic of Business Process Change

Business process change integrates different views from quality, information technology, organizational change, innovation, and work redesign [24]. As such it represents an input-output activity view of business, as opposed to a functional, responsibility-centered structural view. As summarized in Table 1, the horizontal view of the business engendered by business process change practice represents a paradigm shift from the traditional hierarchical organizational structure. The hierarchy with its focus on efficient command and control works well: (1) in environments characterized by stability, limited uncertainty, and limited "consumerism"; (2) when people are subservient to the structure and can follow rules defined by their position; and (3) when markets do not change rapidly and the focus is not on flexibility, quality, service, and innovation [24].

The hierarchy and function-based organization are vertical views of organizations that often involve decisions that translate down the hierarchy and result in choices that are best for the function, not the organization. Cross-function linkage is achieved through work and responsibility handoffs to other functions. However, with time, functions and specialists multiply, as do the rules and bureaucracy to handle increasing contingencies. A pyramid management structure is often required to tie all the pieces together, with the result that many companies are paying more for the glue than for the real work [24].

A process orientation adopts a horizontal view of organizations by emphasizing notions of processes, process owners, teams, and empowerment, and de-emphasizing

Table 1. A comparison of Hierarchical and Process-Oriented Firms

Dimension	Hierarchical Firms	Process-Oriented Firms
Organizational Structure	• Hierarchical organizational structure (based on functions/products) • Linear and sequential processes • Rigid bureaucracy • Organizational integration through structure • Protective organizational culture	• Networked organization based on cross-functional teams (process teams) • Parallel processes • Flexible adhocracy • Organizational integration through information • Productive organization culture
Human Resources	• Fragmented, individual-performed tasks • Functional specialists • Expertise as a functional specialty • Compensation for skill and time spent • Advancement based on ability	• Holistic processes accomplished by teams • Case manager and process manager • Knowledge as organizational resource • Compensation for results • Advancement based on performance
Information Technology/ Systems	• Fragmented, function-oriented information systems • IT as a driver of business process change	• Integrated, process-oriented cross-functional information systems • IT as an enabler of business process change
Management Practices	• Executives as scorekeepers • Managers supervise and control • Management by internal objectives • Function-wide sub-optimization	• Executives as leaders • Managers coach and advice • Management by external objectives • Organization-wide global optimization

Source: Adapted from [7].

hierarchical structures. The key concepts associated with process orientation are outlined below [25].

(1) *Process/customer focus*: Every process has a customer who is either internal or external to the organization. Focusing on the processes and assigning people to them leads to a reduction in confusion and suboptimization; this results in enhanced customer responsiveness, and increased accountability and performance of the entire process.

(2) *Empowerment*: Horizontal organization reduces the up-and-down information flow for a process and empowers workers by giving them decision making rights (or moving decision rights closer to them). This leads to a reduction in approval delays, compressed lead times, and improved customer service.

(3) *Interaction*: This is engendered by use of cross-functional teams that work on common processes. Tighter integration across functions, joint responsibility for a process, and compensation schemes that are based on performance instead of position enhance job satisfaction and yield efficient and effective processes.
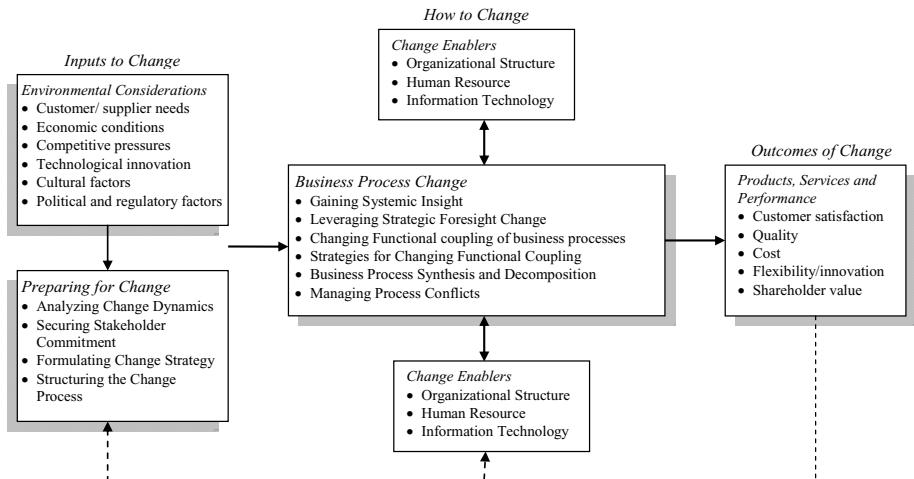
(4) *Top Management Leadership*: Leaderships plays a critical role in BPR initiatives [26]. Top management must formulate and communicate the vision of business process change, and through their transformative leadership, create a sense of mission among organizational members [27, 28]. Top management leadership is necessary to

effectively mobilize resources across functions, organize cross-functional process teams, and communicate the goals and objectives throughout the organization [24].

In the next section, we discuss the IPO framework as a basis to structure business process change initiatives. While some argue that the IPO view oversimplifies the messiness of real-world processes (e.g., [29]), our goal here is to provide a higher-level framework with the view to draw attention to broad-based considerations on the *process* of business process change at the strategic level.

3   The IPO Framework of Business Process Change

We chose to use the input-process-output (IPO) model as a basis for our framework of business process change design. This model is so robust that it can apply to any context in which some inputs are converted to outputs (e.g., manufacturing, service design, systems analysis and design, strategic planning, etc.). Furthermore, a business process represents a stream of activities, their inputs, and their results, which ties in with the IPO framework, even at the process level. Figure 1 presents the IPO model we propose to guide process change design and management. The remainder of the chapter will be organized around this framework. Environmental factors serve as *inputs* to change, which the organization should take into account in preparing for change. The *process* of change (how to change) recognizes that successful business process change initiatives require changing not only business processes, but also other organizational factors and information technology, which serve as enablers. *Output* represents the organization's desired outcomes from business process change effort. While the IPO framework suggests a linear movement from inputs, to processing, to

Fig. 1. The IPO Model of Business Process Change
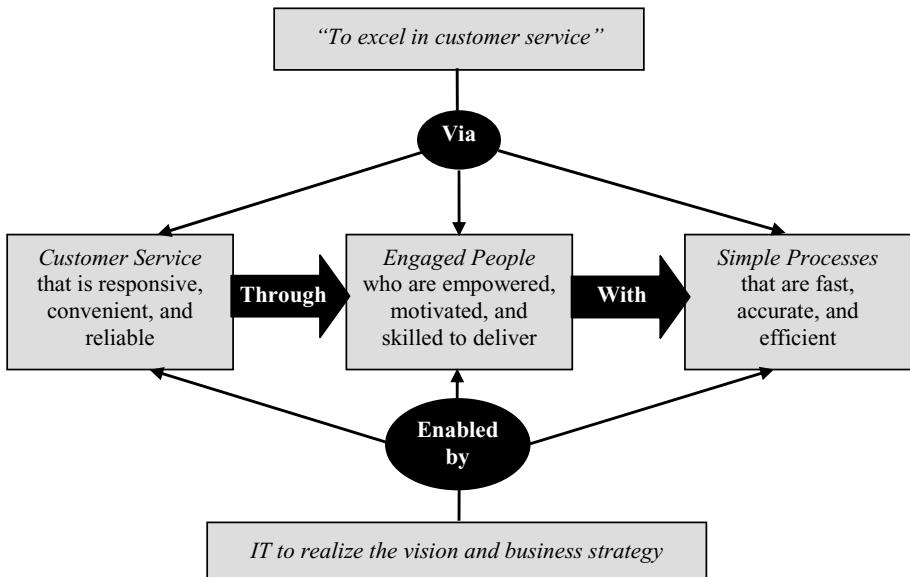
outputs, we argue that there is need-to-know desired outputs before the process of change is executed (this is shown by the dashed link from outputs in Figure 1). Otherwise, change will not be goal-driven. Accordingly, our discussion starts with the outcomes of business process change. Thereafter, we address inputs and the process of change.

3.1 Outcomes of Business Process Change

The anticipated outcomes of business process change should be driven by business strategy. An organization's business strategy can be defined as "the understanding of an industry structure and dynamics, determining the organization's relative position in that industry and taking the action either to change the industry's structure or the organization's position to improve organizational results" [30]. For instance, in most industries with commoditized products, customer service is a key differentiator. Based on this understanding, an organization may develop a strategic vision "*To excel in customer service*". Take for instance the case of banking service. Such an overall vision may be broken down to more concrete objectives, such as to provide superior customer service by:

1. Providing a comprehensive single view of customers' holdings, accounts, and relationships, so the bank can understand them and their business when they talk to a customer service representative.
2. Offering a more consistent experience by recording each interaction, so that the staff can continue with the next step at each customer contact.
3. Anticipating customers' needs by providing more services to match their financial situation with direct requests and referrals between channels and divisions.
4. Saving customers' time by electronically storing customer signatures, investment, and home loan documents and data, so that they do not have to give the same information twice.
5. Following through on customer requests by allocating and scheduling tasks for completion now and in the future.

Focusing on strategy-driven outcomes ensures that the resulting business processes enable the organization to remain well aligned to the requirements of the business, social, and political environments in which it operates. Furthermore, strategy-driven outcomes provide a basis for identifying specific activities and processes through which the organization achieves its business objectives. Following through with our customer service example, Figure 2 summarizes the organization's overall strategy to bring about this desired change in customer service. It highlights the *requirements* for business process change design to achieve the desired organizational objectives. While requirements defined at the level of Figure 2 are likely to be too abstract to begin designing and implementing a solution, the framework serves as a powerful foundation for further transformation of business strategy into desired requirements. This is achieved via a process of domain context analysis and decomposition to refine requirements to increasingly lower levels of abstraction [31]. A higher level of abstraction is desirable because customer needs and the business environment are

Fig. 2. Customer Service Strategy Visualization

dynamic. Trying to include all potential requirements is impractical. There is need to recognize that not all requirements can be known up-front, i.e. be *complete*. As stated before, the challenge then becomes how to deliver the known requirements while still leaving open the ability to deliver the future requirements as conditions change.

3.2   Preparing for Business Process Change

Before embarking on business process change analysis and redesign tasks, it is vital to prepare for change through assessment of the organization's ability to efficiently and effectively deliver the new business process requirements, while at the same time maintaining agreed upon levels of performance within the current environment. This requires paying special attention to change dynamics, securing stakeholder commitment, formulating a change strategy, and structuring the change process with considerations of how tasks, people, and knowledge will be managed. A formal change management process should be fostered to reduce or eliminate disruptions to business activities.

Analyzing change dynamics. A careful analysis of change dynamics is necessary to assess the level of resistance to change engendered by the business process change initiative. Resistance to change may arise due to interaction among several antecedents. From a system's perspective, [32] explains resistance in terms of interaction between the system being implemented and the context of use. She posits that a group of actors will be inclined to use a system if they believe it will support their position of power. However, if they think it might cause them to lose power, they will resist. Power struggle leads to other problems, such as: (1) a disagreement about the nature

of the problem that the proposed change is supposed to solve, and (2) lack of faith in the proposed change's ability to solve the problem [33].

Generally, people weigh how change is going to affect their status quo in the organization. They go through this process by evaluating a given change on three levels [34]. First, they assess the variation in their equity status brought about by the system. Second, they evaluate how change affects their equity relative to that of other members of their reference group. Finally, they compare the equity status of their reference group to that of their organization. They will resist if they perceive inequity. Therefore, careful analysis of the possible causes of resistance to change and political power dynamics is necessary in order to generate guidelines for formulating a specific change strategy and securing management commitment.

Securing stakeholder commitment. A stakeholder is anyone with a vested interest in the business process, including organizational employees, suppliers and customers [35]. Commitment from stakeholders must be secured to build a solid foundation for the change program. A powerful way to secure stakeholder commitment is to get all those concerned involved in the change process. For instance, using the search conference technique, all stakeholders can be assembled into the same room to discuss the need for change and how to best achieve it [36]. During such a meeting, commitment to change may be nurtured through active and open participation by all the stakeholders. It is important that even after the meeting, open and honest communication should be maintained at all levels [35]. Furthermore, resistance to change can also be mitigated through education, facilitation and support (e.g. training), negotiation and agreement, manipulation and co-optation (e.g. assigning resisters to key roles in the change process), and explicit and implicit coercion [37]. However, for long-term effectiveness, manipulation and coercion should be avoided as much as possible because they increase the likelihood of undesirable consequences in the form of poor morale and high likelihood of sabotage. Whenever possible, prototyping the change process is encouraged to help the various stakeholders develop a more concrete appreciation of how the new process works, and how it fits in the overall context of change [7].

Formulating change strategy. After assessing change dynamics, a specific change strategy should be formulated. The change strategy adopted can either be evolutionary (the choice of either technical or social system first, or gradual, staged socio-technical change), or revolutionary (simultaneous change of both technical and social systems) [38]. If forces of resistance are perceived to be high, evolutionary change is preferable because change can be introduced gradually, tempered with frequent and open communication, and adapted to the pace and capabilities of the people [37]. Revolutionary strategy, however, requires a paradigm shift [39] and a dismantling and reconfiguration of the existing structure to prevent the tendency to revert to the status quo once the change program is over. Careful managerial maneuvering through intervention, participation, persuasion, and edict is necessary [40]. Generally, the more tempered approaches of participation and persuasion work better, especially when proper benchmarking and setting of process goals is done.

Structuring the change process. Leadership plays a critical role in BPR initiatives [26] and projects initiated without proper leadership often fail [41]. Specific roles must be assigned to provide some degree of control and structure for the change endeavor [35]. The roles of business process change project sponsor, process owner, and change agents are especially significant. A project sponsor, such as a senior business or IS executive with transformational leadership abilities, is required to legitimize and drive the change process [27, 28]. For the role of a process owner, a senior manager among employees affected by the project is desirable. Such a process owner should possess leadership skill, as well as operational competence to achieve the process performance goals. The members of a cross-functional team serve as the change agents who actually carry out the detailed redesign work and implement the change. Such a team should be facilitated through training in teamwork and group dynamics. A smaller separate team dedicated to organizational change management may also be needed to ensure timely attention to change management [35].

3.3   How to Change (Process of Change)

While the main goal is the change of business processes, the organization context and the role of information technology need to be taken into account in order to achieve successful business process change outcomes. Trying to bring about business process change without corresponding plans for organizational change is likely to increase the risk of failure [7]. Thus, below we discuss not only business process change considerations, but also the complementary changes in organizational factors and the enabling role of information technology.   Table 2 describes questions and activities that enable requirements analysis in this stage for change in process, organization, technology, and management.

Business Process Redesign. Again, business process redesign should be aimed at achieving the overall business strategy. Below, we highlight pointers to what and how to change business processes. Having systemic insight and strategic foresight can provide guidance on what to change in business processes and how to design change [42]. Systemic insight is the ability to visualize connections between business processes and the state of their alignment with supporting information systems to meet business goals. Systemic insight plays a significant role in being able to change the functional coupling of business processes efficiently and effectively. Strategic foresight is the ability to anticipate discontinuities in the business environment and opportunities for business process innovation [42].

Gaining Systemic Insight for Business Process Redesign. Knowing which functional coupling of business processes to changes requires deep systemic insight. An analysis of value-adding activities (i.e., of visible importance to customers) and process mapping can be a source of systemic insight, which then enables the diagnosis of process problems and opportunities. In fact, most of the activities of automated business process management (BPM) are geared toward providing systemic insight. For instance, process mining and simulation can lead to the discovery of how to change in existing processes [43, 44, 45]. Due to the use of automated BPM, many business processes leave their "foot prints" in transactional information systems in the form of event

Table 2. Dimensions of What to Change during Business Process Reengineering

Change Dimension	Major Considerations	Key Activities
Process Design	• What are our major business processes? • Who are their customers? • What are our strategic/value-added processes? • What processes get highest priority from change initiative? • What are our subprocesses, activities and steps? • How do resources and information work through processes? • Why do we do things this way? • What the key strengths and weaknesses of our processes? • Can we benchmark? How? • How would we like these processes to work? • What are our stretch goals for these processes?	Model processes, model customers and suppliers, define and measure performance, define entities or "things" that require information collection, identify activities, map organization, map resources, understand process structure, understand process flow, identify value adding activities, estimate opportunity, benchmark performance, prioritize processes
Organizational Design	• Who is likely to resist these changes and why? • What changes to organizational structure will be required? • What will the new organization look like? • How will the social elements interact with the technical elements? • What human resource practices will be required for the reengineered processes?	Evaluate organizational environment, specify organizational structures, identify job clusters, define jobs/teams, define skills/staffing, design incentives, design transitional organization
Technical Design	• How can IT be used to transform these processes? • What technical resources will we need? • What changes to existing IT resources will be required? • How will all the technical elements work? • How will the technical elements interact with the social elements?	Brainstorm IT possibilities, examine process linkages, model entity relationships, develop performance metrics, consolidate interfaces, consolidate information, and envision technical systems design.
Management Design	▪ How do we ensure that the transition goes smoothly? ▪ Who should be represented on the process change team? ▪ What mechanisms should be established for unanticipated problems? ▪ How do we monitor and evaluate progress? ▪ How do we build momentum for ongoing change?	Setting corporate and process change goals, development of a change plan, identifying and motivating team, training staff, evaluating personnel, monitoring progress, and continuous improvement

Source: Adapted from [24].

logs. By mining these event logs, process, control, data, organizational, and social structures can be discovered [45], which could be used to improve existing processes.

In addition to computer-based (quantitative) simulation tools for business process analysis and redesign, qualitative simulation approaches have also been suggested [46] as a complement to the quantitative approaches. Nissen [46] argues that qualitative simulation provides useful capabilities for the support of business process redesign because many aspects of business processes are inherently qualitative and not well

understood. Using qualitative simulation, complex problems can be abstracted through the development of ontology to capture key entities and relationships about a phenomenon, based only on a few dominant features. This makes it possible to model and codify process entities and relationships, even with limited information and minimal understanding of the details. Qualitative simulation provides visioning through a description of possible behaviors for the modeled process in the early stages, which can then be probed further through quantitative analysis in the latter stages.
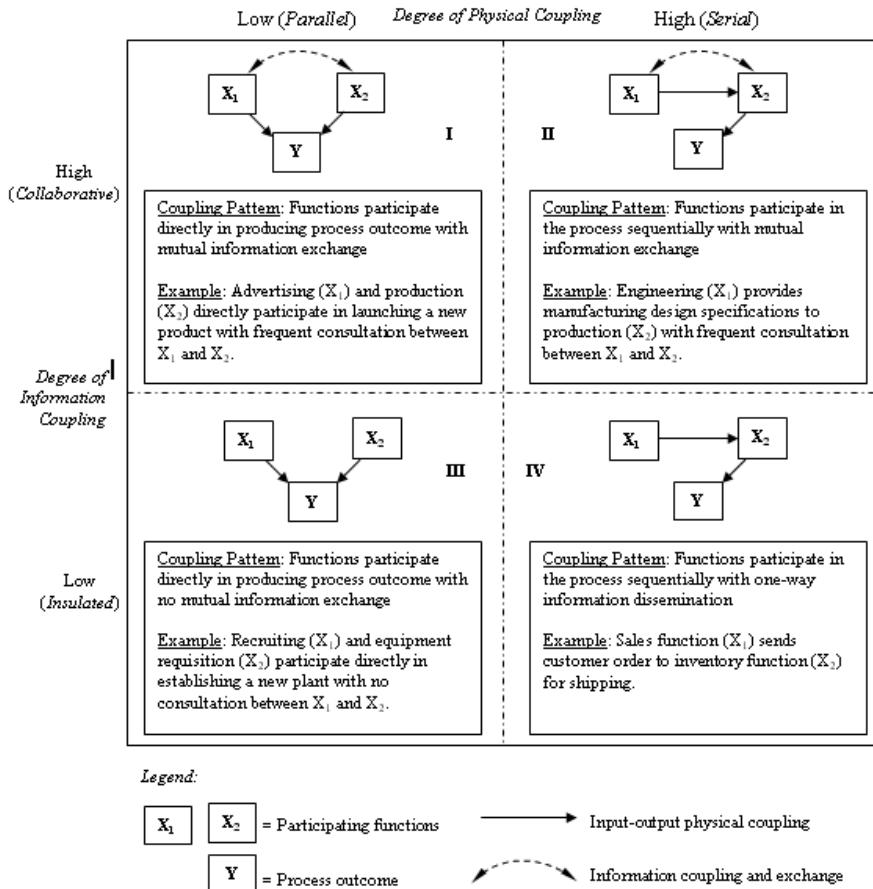
Leveraging Strategic Foresight for Business Process Change. Strategic foresight is "*the ability to anticipate discontinuities in the business environment, marketplace, or the information technology space, the threats and opportunities in the extended enterprise chain, and impending disruptive moves of the competitors*" [42, p. 250]. Through foresight, organizations develop a vision about not only how to improve the existing processes, but also which new ones to introduce in order to effectively compete in the market place. While systemic insight can be largely automated, organizations develop foresight mainly through personal intuition and experiences of managers, as well as organizational intelligence about future market outcomes. Foresight is critical because it reflects the ability to anticipate and visualize market imperfections and opportunities for business process innovation. Through foresight, organizations may be able to formulate answers to the following four fundamental questions about business process change:

- Which of the processes that the industry or marketplace takes for granted should be *eliminated*?
- Which processes should be *reduced* well below the industry's standard?
- Which processes should be *raised* well above the industry's standard?
- Which processes should be *created* that the industry or marketplace has never offered?

Developing concrete answers to these questions can serve as the starting point for business process innovation. The next subsection outlines how organizations can achieve business process innovation by changing functional coupling of business processes.

Changing Functional coupling of business processes. One of the main goals of process analysis and design should be to change functional coupling of business processes to develop processes that are efficient and effective. Systemic insight can yield valuable information about the functional coupling of various business processes. The concept of functional coupling discussed here draws from [7, 47]. *Functional coupling* is the pattern in which various functions are orchestrated while participating in a particular business process. As depicted in Figure 3, this pattern has two components: *physical coupling* and *information coupling*.

Physical coupling arises from input-output (I/O) relationships between a function and other participating functions, involving either transfer of physical objects or handoff of documents from one function to another when a function is included in a business process. The extent of this *flow of input and output* among the participating functions is referred to as the *degree of physical coupling* of a business process.
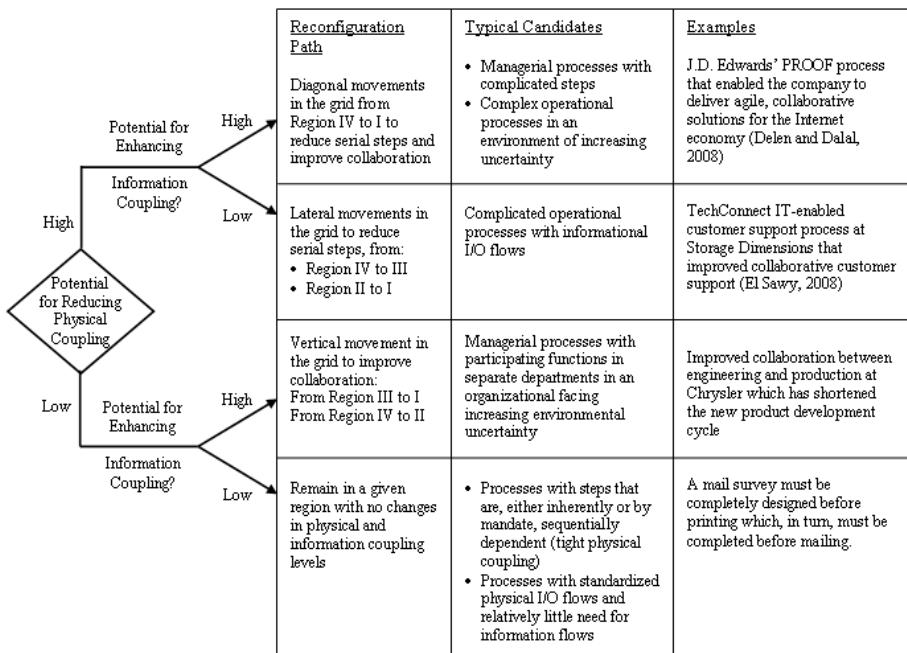
Fig. 3. Dimensions of Functional Coupling of Business Process Adapted from [7]

Physical coupling entails several *intermediate steps* that must be performed before the process is completed. The pattern of these steps can be either *serial* (sequential steps performed by different functions) or *parallel* (several functions contribute *directly* to the process outcome without intermediate steps). On one hand, processes such as business expense processing which require many layers of management approvals, auditor evaluation, and filing of receipts tend to be serial. On the other hand, processes which consist of fairly modular and independent activities tend to be parallel (e.g., process of launching a new product consisting of both production and advertising functions). Nevertheless, serial and parallel patterns are not mutually exclusive since it is possible to have a mixture of both serial and parallel patterns in a process.

Information coupling entails information exchange instead of physical I/O flows. Informational coupling between functions may be either formal or informal, and the frequency and intensity of information exchange between two functions can range from none (completely *insulated*) to extensive (highly *collaborative*).

From Figure 3, four coupling patterns emerge when physical and information coupling dimensions are integrated: *parallel-collaborative* (Region I), *serial-collaborative* (Region II), *parallel-insulated* (Region III), and *serial-insulated* (Region IV). These patterns are illustrated with two functions, X_1 and X_2, which constitute a business process outcome, Y. Moving processes vertically from Region III to I and from Region IV to II is the goal of business process redesign (lateral moves from Region II to I and IV to III, as well as diagonal moves that change physical and information coupling simultaneously (moving from Region IV to I) are also possible).

Strategies for Changing Functional Coupling. Based on the relationships depicted in the grid of Figure 3, some practical strategies can be followed to change the functional coupling of business processes. Figure 4 presents a decision tree that summarizes possible strategies for changing the functional coupling of business processes. Based on the potential for reducing physical coupling, information coupling, or both, typical candidate processes are shown in the middle column, with illustrative examples in the right column of Figure 4. Processes with limited opportunities for reducing physical coupling and limited potential for enhancing information coupling should probably be left as they are. Trying to redesign these processes may lead to process conflict, which could render the redesign effort self-defeating.

Fig. 4. Strategies for Changing Functional Coupling of Business Processes Source: Adapted from [7]

Consider the case of developing new drugs in the pharmaceutical industry in the US. Before the drug can be marketed, complete Federal Drug Administration (FDA) approval is necessary, which may take many years. New drug development also entails many processes with physical I/O flows that are inherently sequential (e.g., proof of concept, sequential clinical trials, etc). If such processes operated in a stable environment without great need for collaboration, they could remain in Region I of Figure 3. However, new drug development is surrounded with a lot of uncertainty and because of this, there is often significant collaboration between pharmaceutical companies, biotech companies, university R&D departments, etc. Thus, typically for pharmaceuticals, these processes should be moved to either Region II or IV.

A process involving information flows, such as the release of a document from one function to another, may be reconfigured by storing the information being transferred in common information resources, such as digitized images or databases. This can facilitate *parallel* execution of operations of the various functions in the process. As indicated by the third branch of Figure 4, these processes (e.g., some operational processes) typically have intricate serial steps, making them good candidates for vertical movement in the grid [48, 9]. Relatively unstructured processes (e.g., managerial processes) require lateral movement to improve collaboration and mitigate uncertainty (branch 2 of Figure 4). Straight lateral or vertical movements in the grid may be sufficient for managerial processes with limited processing steps or operational processes with little or no uncertainty. However, a diagonal path should be considered for managerial processes with complicated serial steps, or operational processes that function in highly uncertain environments (e.g., the new product development process). For instance, members of the design team of Ford's new product development process can simultaneously access a common design database across the Atlantic using computer aided design systems, obviating the need for serial I/O of design documents circulating among the designers [48]. Moreover, networking members can exchange and critique their opinions, even though they have no face-to-face interaction.

Business Process Synthesis and Decomposition. Once strategies for changing the functional coupling of processes have been mapped out, the actual redesign effort of the business process can be implemented by making two types of structural changes to a business process: *synthesis* and *decomposition* [49]. Process synthesis involves combining multiple processes into a single composite process, while process decomposition entails disentangling components of a process that could be better organized as separate processes [49]. Both process synthesis and decomposition can enhance the manageability of a process by removing redundant tasks. For example, a single synthesized order fulfillment process at eBay integrates a variety of partner processes that include payment processes (e.g., Paypal), shipping processes (e.g., FedEx), and other partners' internal processes (e.g., online retailers that sell through eBay). It is important to recognize that process synthesis involves more than simply linking the component processes together. For instance, order status on eBay's web page may be different from that on the web pages of the integrated partners, and may also link to FedEx's tracking process in a way that is different from FedEx's own tracking web page. If the original processes were left in place, potential confusion could ensue, and also redundancies and inefficiencies could result.

It is possible to combine both synthesis and decomposition iteratively during process design and analysis to produce an integrated process. For instance, disparate processes could first be synthesized to remove redundancy and inconsistency, and then decomposed to enhance manageability through process simplification and delegation [49]. Delegation is particularly important in interorganizational settings since processes cut across organizational boundaries.

Business process change involves inherently complex processes, with many tasks, information elements, and resources. Therefore, process redesign through synthesis and decomposition should generate integrated processes that are not just a collection of incomplete, incompatible, or wasteful processes [2]. To circumvent potential problems, as the above discussion highlights, process redesign should be preceded by a careful analysis of the potential for business process redesign, with opportunities and limitations explicitly mapped out.

Managing Process Conflicts. Even if there may be a high potential for reducing physical coupling and enhancing information coupling, it is important to bear in mind that in reality process conflicts may arise. Process conflict is unexpected or contradictory interaction between process functions that has a negative effect on the performance of the process. This may arise when functions of a process controlled by two or more stakeholders cause an inconsistency. In other words, function conflicts are interactions and dependencies between process functions that can lead to negative or undesired process outcome. Three types of process function conflicts can arise:

(1) *Activity conflicts*: Activity conflicts arise when components belonging to different processes achieve the same action, or the components of the same process perform opposite actions at the same time. As a result, these conflicts have a negative effect on the functioning of the process [50]. Activity conflicts may suggest the need to carry out further process synthesis and decomposition to eliminate these conflicts.

(2) *Resource conflicts*: Resource conflicts arise when processes that have different targets attempt to use a limited resource at the same time. For example, a network may have limited resources, which affects concurrent information access by different processes. It may also be due to serial coupling discussed earlier. For instance, a customer may fail to view their account balance at the same time the bank is updating account information (concurrency problem). Resource conflicts can be circumvented through innovative design of the enabling role of information technology and systems [50].

(3) *Control conflicts*: These may arise when several departments or organizations are involved, or when decomposition results in outsourcing parts of the process. A variety of legal issues may arise, such as intellectual property protection, accountability, security concerns, and liability [49]. Furthermore, human resource management issues are also a major consideration, especially when process synthesis or decomposition results in the transfer of parts of the process to new functional areas of the organization (e.g., from manufacturing to distribution). These potential problems should be kept in mind when implementing process redesign and appropriate safeguards should be put in place [51].

Business Process Change Enablers. Accomplishing business process change is facilitated by deploying a number of enablers of the desired change. In addition to IT,

organization structure, human resources, and management practices are a significant consideration in process change initiatives. Sometimes the focus is on IT only, which limits the prospects of success of the overall change initiative [52]. As discussed below, all the various enablers of business process change should be taken into account in order to achieve overall success of the business process change initiative.

Organizational structure enablers. Successful business process redesign requires adopting a process-based organizational structure instead of the functional departmental structure of the traditional hierarchy. A process structure includes the entire set of functions in a process needed to complete at entire process [53]. Cross-functional cooperation can be facilitated by modifying the hierarchical structure through structural enablers such as cross-functional teams, case managers, and process generalists. For example, IBM reorganized its consulting practice away from functional departments (e.g., hardware, software) to consumer teams responsible for the complete order fulfillment process [54]. Kodak also benefited form the use of cross-functional teams in its black-and-white film operation. Previously, the operation was running 15% over budgeted cost, took up to 42 days to fill an order, and was late a third of the time. The redesigned process centered on cross-functional teams resulted in cost savings of about 15%, cut response time by half, and significantly reduced late delivery to one out of 20 times [7]. Moreover, with the help of IT, it is possible to have structurally diverse teams that transcend location or organizational boundaries (i.e., virtual teams).

Majchrzak and Wang's [53] study found that process-based organizational structures generally lead to greater efficiencies than do functional departmental structures, but only if coupled with practices that create collective responsibility. This requires special consideration of human resource management and development issues [55].

Human resource enablers. Successful implementation of business process change requires complementary changes in human resource management practices, since it is through people that these changes are carried out [56, 57]. While a traditional functional organization relies on specialists, cross-functional team membership requires employees to have some rudimentary knowledge of other functions in order to communicate effectively with personnel from other departments. Greater demands for cross-functional knowledge are put on case managers and process generalists. Therefore, mechanisms to foster multiple skill development and reward team performance should be put in place. Mechanisms such as overlapping job tasks among members, work procedures that encourage collaboration over individual performance, and providing rewards for team (not individual) performance have been found to be related to higher performance [54].

In a study of 54 process-based virtual teams, [56] found that practices of inclusion in decision making fostered team spirit and led to success of the teams. In another study of a process-based new product development virtual team, [57] also found that frequent virtual team meetings, co-creation of boundary objects, and information sharing were associated with team success. In companies like GE and the Government Electronics group in Motorola, peers and others above and below the employee evaluate the performance of an individual in a process, sometimes involving as many as 20 people. This has fostered a reward system that is based not only on individual

performance, but also on team performance [7]. In several companies now, cross-functional team orientation has shifted the emphasis from behavioral control to output control (e.g., at JP Morgan Chase, team leaders are paid based on performance).

Generally, many successful business process change efforts require external focus with emphasis on external performance objectives such as customer satisfaction and overall product/service quality. For example, Marshall Electronics transformed its commission-based sales system to better meet its goal of achieving a customer-focused global chain operation. With the commission-based system, salespeople often timed shipments to gain their commissions rather than to meet customers' needs. When Marshall eliminated the commission-based system and replaced it with a profit-sharing system, this change boosted the company's sales by 200 percent, making Marshall the fourth-largest electronics distributor in the world [58].

Information technology enablers. Besides its direct role in automating a number of business processes, IT can be used to support the operational aspects of conducting business process redesign [59]. IT can, for example, enable: the recoding of various business processes, analysis of current and proposed processes, keeping track of deadlines, balancing capabilities with demands, and the flow of information and documentation among the various participants. Given our emphasis on process analysis and design, we focus on IT's role in supporting this process, instead of IT's role in automating business process, which often receives a lot of attention.

The management of knowledge related to the business process initiative is sometimes overlooked as a business process change enabler [60, 61]. Process designers should be able to acquire and use knowledge about business environment, design decisions, and alternatives during process change activities. Also it is important to recognize that over a period of time, not only does the context of a business process change, but also the information needs of the process designers change. Therefore, mechanisms should be put in place to maintain contextual knowledge to keep it up-to-date. Furthermore, practice, the way in which work gets done, is often different from the process described formally in manuals, training programs, organizational charts, and job descriptions [62]. Practice needs to be managed by facilitating or supporting people in getting their work done in particular contexts [63]. Table 3 summarizes the various aspects of process-related knowledge and the corresponding capabilities of a good knowledge management system to support business process change efforts. Nissen [61] discusses the integration of knowledge-based systems into the transformation process and demonstrates the utility of two diagnostic knowledge-based tools for process analysis.

Discussion. According to the framework discussed above, business process change entails making complex design decisions that may affect different, but interacting and interrelated dimensions of an organization: its processes, people, strategy, culture, information policies, and its environment. A change in one of these aspects may have unknown or unexpected consequences on others. For example, a new strategic direction or the adoption of a process structure may not be favored by the workforce and may have detrimental effects on staff morale and productivity. Therefore, it is important that alternative organizational aspects be taken into account and aligned when

Table 3. Business Process Change Knowledge Management Requirements

Knowledge Aspect	Required Capabilities
Knowledge of business environment	• A knowledgebase with the information on various contextual factors to guide users on process redesign • Mechanisms for users to represent contextual information – including process elements, business rules and policies influencing process elements, decisions and explanations
Access to process adaptation knowledge	• A knowledgebase with relevant information on process behavior, process pathology detection, and guidance for process adaptation and improvement
Flexibility in the range dimension	• The ability to provide users with multiple alternatives for process adaptation and improvement, based on process pathologies identified
Acquisition, use, and evolution of contextual process adaptation knowledge	• Facility to access and review existing contextual information during process design activities • Facility to update the existing contextual information during process design deliberations • Facility for multiple stakeholders to actively participate in the process change deliberation

change is designed or introduced. This need makes the process change design problem complex, demanding, and laborious.

The IPO framework takes into account the various aspects of the change initiative and yet is general enough to avoid getting lost in the details. To maintain focus on the goal, we propose that organizations address three fundamental questions that underpin the IPO framework: *what to change*, *how to change*, and *what to change to*. Organizations should address the last question first before moving forward with the change effort.

What to Change to? "What to change to" is the anticipated outcome of the change effort and should be guided by the organization's strategic vision. As the traditional distinction between products and services increasingly becomes irrelevant, customer focus is becoming a major strategic driver for many organizations [64, 65]. Organizations are moving closer to their customers, continuously trying to find new ways to create customer value, and transforming the customer relationship from one of selling and order taking into one of solution finding and partnering. Thus, customer-oriented processes are among today's most critical core business processes that can benefit from improvement. For example, a vendor of high-availability disk and tape storage for client/server environments with a customer-focused strategy transformed its customer support process through TechConnect [64]. As a result the average response time to a customer problem dropped significantly from about 2-3 hours to only 15 minutes. Furthermore, TechConnect had analytical capabilities that enabled staff to uncover patterns and take proactive action for further prevention of any apparent problems.

What to Change? While the main focus of change is the business processes being transformed, selecting proper change enablers lays a solid foundation for the success of a business process change project. Several case studies done by [52] showed that change implementation requires flexibility and a multidisciplinary perspective, with appropriate considerations for several human, social, cultural, and political issues. These components should not be overshadowed by a rigid business process change methodology and technical issues. Appropriate management of process change implementation requires addressing implementation problems, politics, and tactics, as we have highlighted above.

How to Change? We have drawn attention above to the practice of business process analysis to uncover process redesign and transformation requirements, as well as the need to pay attention to various enablers of change. Rather than go through the details already discussed above, drawing from [60], we provide a summary of the following guidelines that can be drawn from our framework:

1. *Focus transformation efforts on critical business issues that have payoff and are aligned with organizational strategy.* Identify critical business issues for which the change is intended to provide concrete improvements. Furthermore, recognize that the ability to deploy appropriate business processes requires that the fit between those processes and organizational strategy be continuously maintained and evolved.
2. *Establish enterprise-level support.* The process change initiative should enable all relevant stakeholders to participate. All stakeholders should be made to understand the problems and the rationale behind chosen improvements. This will motivate them to "own" change and the initiative will profit from their knowledge about the context (e.g., which suggestions will work).
3. *Take a broader view of process change.* Process change is a social process involving several different process participants with diverse set of goals, requirements, assumptions, and constraints. All these should be carefully considered in order to achieve success. If the context for which a process has been designed changes, it is important to reevaluate its applicability and identify potential options for redesign.
4. *Avoid "ownership" of the change efforts by a specific function.* Complex patterns of interrelationships among processes, people, and technology need to be addressed in a balanced manner. Change efforts must draw from a range of skills, independent from functional influence. Success comes from integrating a range of skills in cross-functional teams.
5. *Recognize that information technology is an enabler, not a driver of process change.* IT should not drive or be the cause of change; rather emphasis should be first on the business processes; then, IT can be creatively used to enable those processes.
6. *Strive for parsimony, not excess.* When change is rapid, trying to achieve completeness, consistency, and correctness is not always efficient. Known requirements should be delivered immediately while still leaving open the ability to deliver future requirements as the market unfolds. Otherwise, minimize upfront business requirements but maximize future options because quite often, a wide range of potentially well-suited redesign strategies exist.

7. *Prototype fast and frequently.* Prototyping can help uncover the inconsistencies and conflicts which may still exist in the redesigned processes due to functional coupling or other causes. This suggests that further process synthesis and decomposition may be required.

8. *Do not forget disciplined project management.* Members of the senior management must be committed to the initiative and must demonstrate their commitment by being visibly involved with the project. Managers need to make careful use of "signals" (clear and explicit messages), "symbols" (actions that indirectly reinforce the signals), and "reward systems" to manage the change process [28]. Management should not lose sight of the goal.

When followed closely, these guidelines can help in achieving the desired outcomes. For example, Nortel Networks transformed the front end of its new product development (NPD) process based on these guidelines. Nortel transformed the previously ills-structured and ad hoc front-end NPD process to one that was consistent over time and across people [60].

4   Conclusion

Business process redesign and improvement is especially important in dynamic, complex business environments. The ability to quickly and continually adapt business processes to accommodate the evolving requirements and exploit opportunities is critical for success. Without appropriate linkage between redesign decisions and strategic inputs, identifying processes that need to be modified will be difficult. In this paper, we proposed an IPO framework to guide the analysis and design tasks of business process change.

The high-level IPO framework proposed in this study provides strategic perspective on business process change analysis and design, with emphasis on both the business process level and the overall organizational context. This framework provides guidance on how various functional activities involved in a business process may be redesigned. By focusing on process dependencies, this approach leverages workable assessments to identify and establish process improvements. For example, analyzing the functional coupling of processes makes it possible to devise cogent strategies for process redesign. The framework can also serve as a starting point for identification of practices that can be introduced in one step through process synthesis and decomposition techniques.

The framework also underscores the significance of managing knowledge on process analysis and design requirements. Even the most-capable and experienced process engineers face difficulties in redesigning processes without access to the knowledge that shaped previous design decisions. While often overlooked, the acquisition and maintenance of contextual knowledge about process models and process redesign rules used in redesign efforts can facilitate the streamlining of complicated procedures. Given the cross-functional and radical nature of process redesign, a lot of learning is required of process workers, and as a result they also create new knowledge, which should be captured at the organizational level.

Furthermore, business process redesign is a multifaceted phenomenon involving organizational structure, management systems, human resource architecture, and

many other aspects of organizational life. The IPO framework presented in this paper should prove helpful in gaining a high-level perspective and guiding the analysis and redesign of business processes. Any framework, however, only maps tasks and activity requirements. Management is thereby responsible for budgeting along mapped activities, directing (redirecting) process workers, and exhibiting visible support. In other words, a framework is just a blueprint; management must lead change. Moreover, people must be rewarded, not reprimanded, for taking calculated risks. Information technology plays a supportive role in this process. Hence, the vision must account for the environment, methodology, people, and IT.

We have drawn attention to the *process* of analysis and design of processes in support of business process change initiative. While the framework presented provides some high-level guidelines, it is a simplification of the complexities of real processes in an attempt to highlight possibilities for redesign. Future research could enrich this framework by developing more rigorous methods for recording processes and identifying dependencies in organizations. This could serve as a starting point for a more formalized methodology for documenting requirements analysis for business process redesign and transformation. Furthermore, given that organizations are increasingly becoming interconnected, future research should explore dependencies not only within organizations but also across organizations, as well as the coordination mechanisms necessitated by these dependencies.

References

1. Grover, V., Kettinger, W.J.: Business Process Change: A Reflective View of Theory, Practice, and Implications. In: Zmud, R.W. (ed.) Framing the Domains of IT Management, pp. 147–172. Pinnaflex, Cincinnati (2000)
2. Curtis, B., Kellner, M.I., Over, J.: Process Modeling. Communications of the ACM 35(9), 75–90 (1992)
3. Mela, N., Pidd, M.: Business processes: Four perspectives. In: Grover, V., Markus, M.L. (eds.) Business Process Transformation, pp. 41–66. M.E. Sharpe, Armonk (2008)
4. Kleiner, A.: Revisiting Reengineering. Strategy & Business 20, 27–31 (2000)
5. Wang, P.: Whatever Happened to Business Process Reengineering? The Rise, Fall, and Possible Revival of Business Process Reengineering from the Organizing Vision Perspective. In: Grover, V., Markus, M.L. (eds.) Business Process Transformation, pp. 23–40. M.E. Sharpe, Armonk (2008)
6. Grover, V.: From Business Process Reengineering to Business Process Change Management: A Longitudinal Study of Trends and Practices. IEEE Transactions on Engineering Management 46(1), 36–46 (1999)
7. Teng, J.T.C., Grover, V., Fielder, K.D.: Developing Strategic Perspectives on Business Process Reengineering: From Process Reconfiguration to Organizational Change. OMEGA 24(3), 271–294 (1996)
8. Melao, N., Pidd, M.: A Conceptual Framework for Understanding Business Processes and Business Process Modeling. Information Systems Journal 10, 105–129 (2000)
9. Grover, V., Teng, J.T.C., Fiedler, K.: Technological and Organizational Enablers of Business Process Reengineering. In: Grover, V., Kettinger, W.J. (eds.) Business Process Change: Reengineering Concepts, Methods and Technologies, pp. 16–33. Idea Group, Hershey (1995)

10. Davenport, T.H., Prusak, L., Wilson, H.: What's the Big Idea? Creating and Capitalizing on the Best New Management Thinking. Harvard Business School Press, Boston (2003)
11. El Sawy, O.A.: Redesigning Enterprise Process for E-Business. McGraw-Hill, New York (2001)
12. Davenport, T.H.: The Fad That Forgot People. Fast Company 1, 70–74 (1995)
13. Alter, A.E.: Business Process Improvement Survey: Creating, Smarter, Faster, and Cheaper Processes is It's Main Mission. CIO Magazine (October 2006)
14. Grover, V., Kettinger, W.J.: Special Section: The Impacts of Business Process Change on Organizational Performance. Journal of Management Information Systems 14(1), 9–12 (1997)
15. Armistead, C., Rowland, P. (eds.): Managing Business Processes: BPR and Beyond. John Wiley & Sons, Chichester (1996)
16. Kock, N., McQueen, R.: Product Flow, Breadth and Complexity of Business Processes. Business Process Management Journal 2(2), 8–22 (1996)
17. Dietz, J.: DEMO: Towards A Discipline of Organization Engineering. European Journal of Operation Research 128, 351–363 (2001)
18. Garvin, D.: The Process of Organization and Management. Sloan Management Review 39(4), 33–50 (1998)
19. Earl, M., Khan, B.: How New is Business Process Redesign? European Management Journal 12(1), 20–30 (1994)
20. Tushman, M., Nadler, D.: Information Processing as an Integrating Concept in Organizational Design. Academy of Management Review 3, 613–624 (1978)
21. Huber, G.P., McDaniel, R.R.: The Decision Making Paradigm of Organization Design. Management Science 32(5), 576–589 (1986)
22. Moore, T.C., Whinston, A.B.: A Model of Decision Making with Sequential Information Acquisition. Decision Support Systems 2(4), 289–308 (1986)
23. Orman, L.V.: A Model Management Approach to Business Reengineering. In: Proceedings of the American Conference on Information Systems. Association for Information Systems, Pittsburg (1995)
24. Grover, V., Malhotra, M.K.: Business Process Reengineering: A Tutorial on the Concept, Evolution, Method, Technology and Application. Journal of Operations Management 15, 193–213 (1997)
25. Kettinger, W.J., Grover, V.: Special section: Toward a Theory of Business Process Change Management. Journal of Management Information Systems 12(1), 9–30 (1995)
26. Sarker, S., Lee, A.S.: Using Positivist Case Research Methodology to Test Three Competing Practitioner Theories-in-Use of Business Process Redesign. Journal of the AIS 7, 1–72 (2002)
27. Carr, D.K., Johnson, H.J.: Best Practices in Reengineering: What Works and What Doesn't in the Reengineering Process. McGraw-Hill, New York (1995)
28. Hammer, M., Champy, J.: Reengineering the Corporation. Free Press, New York (1993)
29. Ould, M.A.: Business Processes: Modeling and Analysis for Re-Engineering and Improvement. John Wiley, Chichester (1995)
30. Oliver, R.W.: What is Strategy, Anyway? Journal of Business Strategy 2, 7–10 (2001)
31. Basu, A., Blanning, R.W.: A Formal Approach to Workflow Analysis. Information Systems Research 11(1), 17–36 (2000)
32. Markus, M.L.: Power, Politics, and MIS Implementation. Communications of the ACM 26(6), 430–444 (1983)
33. Lapointe, L., Rivard, S.: A Multilevel Model of Resistance to Information Technology Implementation. MIS Quarterly 29(3), 461–491 (2005)

34. Joshi, K.: A Model of Users Perspective on Change: The Case of Information Systems Technology Implementation. MIS Quarterly 15(2), 229–240 (1991)
35. Davenport, T.H.: Process Innovation: Reengineering Work through Information Technology. Harvard Business School Press, Boston (2003)
36. Pasmore, W.A., Fagans, M.R.: Participation, Individual Development, and Organizational Change: A Review and Synthesis. Journal of Management 18(2), 375–388 (1992)
37. Silva, L., Backhouse, J.: The Circuits-of-Power Framework for Studying Power in Institutionalization of Information Systems. Journal of AIS 4(6), 294–336 (2003)
38. Stoddard, D.B., Jarvenpaa, S.L.: Business Process Redesign: Tactics for Managing Radical Change. Journal of Management Information Systems 12(1), 81–107 (1995)
39. Gersick, C.J.G.: Revolutionary Change Theories: A Multi-Level Exploration of the Punctuated Equilibrium Paradigm. Academy of Management Review 16(1), 10–36 (1991)
40. Nutt, P.C.: Context, Tactics, and the Examination of Alternatives during Strategic Decision Making. European Journal of Operations Research 124(1), 159–186 (2000)
41. Hammer, M., Stanton, S.: The Reengineering Revolution. HarperCollins, New York (1995)
42. Sambamurthy, V., Bharadwaj, A., Grover, V.: Shaping Agility through Digital Options: Reconceptualizing the Role of IT in Contemporary Firms. MIS Quarterly 27(2), 237–263 (2003)
43. Greasley, A.: Using Process Mapping and Business Process Simulation to Support a Process-Based Approach to Change in a Public Sector Organization. Technovation 26(1), 94–103 (2006)
44. Sarker, S., Lee, A.S.: Does the Use of Computer-Base BPC Tools Contribute to Redesign Effectiveness? Insights from a Hermeneutic Study. IEEE Transactions on Engineering Management 53(1), 130–145 (2006)
45. van der Aalst, W.M.P.: Business Alignment: Using Process Mining as a Tool for Delta Analysis and Conformance Testing. Requirements Engineering 10, 198–211 (2005)
46. Nissen, M.E.: Designing Qualitative Simulation Systems for Business. Simulation & Gaming 27(4), 462–483 (1996)
47. Teng, J.T.C., Grover, V., Fielder, K.D.: Business Process Reengineering: Charting a Strategic Path for the Information Age. California Management Review 36(3), 9–31 (1994)
48. Davenport, T.H., Short, J.E.: The New Industrial Engineering: Business Change or Mythic Proportions? MIS Quarterly 18(2), 121–127 (1990)
49. Basu, A., Blanning, R.W.: Synthesis and Decomposition of Processes in Organizations. Information Systems Research 14(4), 337–355 (2003)
50. Crowston, K.: Taxonomy of Organizational Dependencies and Coordination Mechanisms. In: Malone, T.W., Crowston, K., Herman, G. (eds.) The Process Handbook, pp. 85–108. MIT Press, Cambridge (2003)
51. Crowston, K.: The Bug Fixing Process in Proprietary and Free/Libre Open Source Software: A Coordination Theory Analysis. In: Grover, V., Markus, M.L. (eds.) Business Process Transformation, pp. 69–99. M.E. Sharpe, Armonk (2008)
52. Willcocks, L., Smith, G.: IT-Enabled Business Process Reengineering: Organizational and Human Resource Dimensions. Journal of Strategic Information Systems 4(3), 279–301 (1995)
53. Majchrzak, A., Wang, Q.: Breaking the Functional Mind-Set in Process Organizations. Harvard Business Review 74(5), 93–99 (1996)
54. Majchrzak, A.: Breaking the Functional Mind-Set: The Role of Information Technology. In: Grover, V., Markus, M.L. (eds.) Business Process Transformation, pp. 125–139. M.E. Sharpe, Armonk (2008)

55. Delen, D., Dalal, N.: Successful Business Process Transformation at J.D. Edwards. In: Grover, V., Markus, M.L. (eds.) Business Process Transformation, pp. 237–250. M.E. Sharpe, Armonk (2008)
56. Majchrzak, A., Malhotra, A., Stamps, J., Lipnack, J.: Can Absence Make a Team Grow Stronger? Harvard Business Review 82(5), 131–137 (2004)
57. Malhotra, A., Majchrzak, A., Carman, R., Lott, V.: Radical Innovation without Collaboration: A Case Study at Boeing-Rocketdyne. MIS Quarterly 25(2), 229–249 (2001)
58. Willis, C.: How Winners Do It. Forbes 162(4), 88–91 (1998)
59. Mohsen, M.D.H.: Reengineering the Process in the Design of Object-Orientation Using Concepts of Industrial Engineering. International Journal of Computer Applications in Technology 15(6), 167 (2003)
60. Massey, A.P., Montoya-Weiss, M.M., O'Driscoll, T.M.: Transforming the new product development process: Leveraging and managing knowledge. In: Grover, V., Markus, M.L. (eds.) Business Process Transformation, pp. 185–206. M.E. Sharpe, Armonk (2008)
61. Nissen, M.E.: Redesigning Reengineering through Measurement-Driven Inference. MIS Quarterly 22(4), 509–510 (1998)
62. Brown, J.S., Duguid, P.: Knowledge and Organization: A Social Practice Perspective. Organization Science 12(2), 198–213 (2001)
63. Brown, J.S., Duguid, P.: The Social Life of Information. Harvard Business School Press, Boston (2000)
64. El Sawy, O.A.: Redesigning IT-Enabled Customer Support Processes for Dynamic Environments. In: Grover, V., Markus, M.L. (eds.) Business Process Transformation, pp. 157–183. M.E. Sharpe, Armonk (2008)
65. Treacy, M., Wiersema, F.: The Discipline of Market Leaders. Addison-Wesley, Boston (1995)

The Intertwining of Enterprise Strategy and Requirements

Pericles Loucopoulos[1] and Joy Garfield[2]

[1] The Business School, Loughborough University,
Loughborough, LE11 3TU, United Kingdom
`p.loucopoulos@lboro.ac.uk`
[2] Manchester Business School, University of Manchester,
Manchester, M13 9PL, United Kingdom
`joy.garfield@postgrad.manchester.ac.uk`

Abstract. Requirements Engineering techniques need to focus not only on the target technical system, as has traditionally been the case, but also on the interplay between business and system functionality. Whether a business wishes to exploit advances in technology to achieve new strategic objectives or to organise work in innovative ways, the process of Requirements Engineering could and should present opportunities for modelling and evaluating the potential impact that technology can bring about to the enterprise. This chapter discusses a *co-designing* process that offers opportunities of change to both the business and its underlying technical systems, in a synergistic manner. In these design situations some of the most challenging projects involve multiple stakeholders from different participating organisations, subcontractors, divisions etc who may have a diversity of expertise, come from different organisational cultures and often have competing goals. Stakeholders are faced with many different alternative future 'worlds' each one demanding a possibly different development strategy. There are acute questions about the potential structure of the new business system and how key variables in this structure could impact on the dynamics of the system. This chapter presents a framework which enables the evaluation of requirements through (a) system dynamics modelling, (b) ontology modelling, (c) scenario modelling and (d) rationale modelling. System dynamics modelling is used to define the behaviour of an enterprise system in terms of four perspectives. Ontology modelling is used to formally define invariant components of the physical and social world within the enterprise domain. Scenario modelling is used to identify critical variables and by quantitatively analyzing the effects of these variables through simulation to better understand the dynamic behaviour of the possible future structures. Rationale modelling is used to assist collaborative discussions when considering either ontology models or scenarios for change, developing maps, which chart the assumptions and reasoning behind key decisions during the requirements process.

Keywords: Requirements, business strategy, co-development, ontology, scenarios, rationale, modelling.

1 Introduction

Requirements Engineering (RE) as a field of study and practice has traditionally focused on the specification of technical requirements (i.e., defining the functional and

non-functional properties of target systems [1-6]). Over the past 20 years there has been considerable progress in conceptual frameworks, and tools for capturing and modelling of requirements with emphasis being on function enrichment. However, there is an increasing realisation and consensus amongst information systems researchers and practitioners that the development of systems is not solely a technical activity and that organisational factors very often have a profound effect on both the delivered system and the design process. This is particularly pronounced in today's organisations where two phenomena in particular seem to play an increasingly important role in sustainability and development: (a) *growth in information processing* and (b) *enterprise transformation*. Both of these accentuate the intertwining of enterprise and systems, the needs of an enterprise for systems that can both provide relevant and valuable information to carry out their work better and exploit system functionalities for carrying out their work differently.

According to Gantz, Chute et al. (2008) the *growth in information processing* is rising exponentially. In 2008, the volume of information created was estimated at 410 exabytes, already surpassing the capacity to store it and that by 2011 the gap between information generation and information storing will increase dramatically to almost 1000 exabytes. Dealing with such an information explosion is both a technical and organisational problem. Organisations are responsible for the security, privacy, compliance and reliability of 85% of all information being created [7]. During RE such organisational issues need to be considered at the outset if an organisation is to deal with economic and legal issues that may arise from the use of systems managing their information resource. Even environmental issues come into play when one considers that in 2000 power consumption per server rack was 1KW, whereas data centres nowadays operate closer to 20KW per server rack.

It is a challenge for organisations to change and innovate in global market environments that are quickly becoming more unpredictable, with organisations that have become more virtual and mobile, with technologies that are becoming revolutionary and integrative, and with people that are more independent, knowledgeable, assertive and mobile [8]. *Enterprise transformation* offers the opportunity to organise work in ways that have never before been possible [9]. It concerns fundamental change in established relationships between an enterprise and its constituency (market or society) in the way that products or services are provided [10]. Information systems specifically and information technology generally may be deployed in order to organise internal processes to deliver these new sets of relationships. It seems therefore prudent that during RE one needs to consider what the likely impact of support systems will be on the enterprise against multiple strategic criteria.

These two issues provide the motivational backdrop for the arguments, positions and approach presented in this chapter. The central tenet of this chapter is that the role of requirements is changing in that one needs to consider not only functionality and quality of systems but also the impact that specific choices of system requirements will have on enterprise designs.

As an example, consider the case of developing a new automated meter reading (AMR) system by an electricity distribution company, an application which has received much attention in recent years due to deregulation directives and greater awareness of governments and consumers about energy saving factors. There are obvious functional and non-functional issues for the system regarding, amongst others, the meter interface, the communication system and the central office system. In

developing such a system one will have to consider alternative implementations of individual components as well as in the way the system components may be integrated. Traditionally, a requirements engineer would deploy appropriate approaches to model a variety of perspectives such as for example, the business processes e.g. [11], the goals for change e.g. [12], scenarios of use e.g. [13] and perhaps using some requirements specification management tool to cope with plethora of requirements e.g. [14].

There is a key question however, which may have a profound effect on strategic decisions and that is "*what will the effects of the AMR be on other parts of the enterprise?*". Normally, such a question would not be directly addressed by RE techniques. Even approaches that deliberately seek to understand the goals of an enterprise make the assumption that these goals are known, at least as far as their relation of the enterprise is to the desired system and that design decisions about the system will have no influence on enterprise strategy. However, different choices on system functionality may impact in different ways on critical enterprise facets such as customer relations, financial models, internal processes and human resources. These are key strategic issues that will need to be answered by management at some point in the re-designing of the electricity enterprise. Often these are addressed informally and outside the domain of system development.

The argument put forward in this chapter is that RE, acting as a conduit between business-oriented and system-oriented concerns, is ideally placed to examine *systemically* the feedback mechanisms involved in complex systems thus, yielding benefits to both activities of the designing of the enterprise and that of the system. Understanding early on in the development process the impact of different requirements choices on the enterprise itself is much more likely to actively engage stakeholders, to highlight the strategic options open to them and thus, ultimately deliver useful and sustainable systems that are aligned to enterprise strategy and offer opportunities for influencing this strategy. This process, otherwise known as co-development, aims to ensure alignment between business processes and support technical systems [15], a concept also discussed by Rolland in this volume. A number of other approaches have been put forward for co-development, e.g. [15], [16], [17] and [18]. For example Bleistein et al (2006) [19] proposed an approach which integrates problem frames; goal-oriented modelling techniques and business process modelling in order to ensure that IT systems requirements are aligned with, provide support for, and enable business strategy. Although these methods guide development there tends to be a lack of additional support to accommodate challenges that arise during co-development activities.

The role of design thinking in such a co-development activity is critical. It is often tempting to assume that the route from requirements to implementation can be somehow managed in a predictable way. This assumes a *rational* stance but, this is questionable for most systems because: (a) goals change due to deliberation and negotiation between multiple stakeholders, (b) change is hardly ever linear and its effects may be discovered in social systems in unsuspected ways and potentially a long way away from its source [20], (c) in practice requirements evolve, for as Schön observed "… as one wrestles with the problem one's requirements change" [21], (d) models are only as good as the assumptions on which they are based [22], (e) these assumptions can only be validated by stakeholders if the effects of their choices are clearly shown in the stakeholders' own vocabulary [23]. These issues highlight the importance of using a

designing stance when considering requirements. The term 'designing stance' is used in this chapter to mean that the process should involve reflection, exploration, negotiation, compromise and revision. It seems that these are the activities in which top class designers engage when considering complex projects in uncertain situations [24].

In adopting a designing stance to requirements, the following is a set of objectives espoused by the approach discussed in this chapter:

- to encourage the use of multiple models as a way of inventing and proposing alternatives,
- to exploit invariant structures (e.g. standards, legislation etc) which can act as an anchor to alternative models,
- to engage stakeholders in developing scenarios of alternative futures under different conditions, and
- to submit the reasoning used in model building and in the development of scenarios to critiquing by stakeholders.
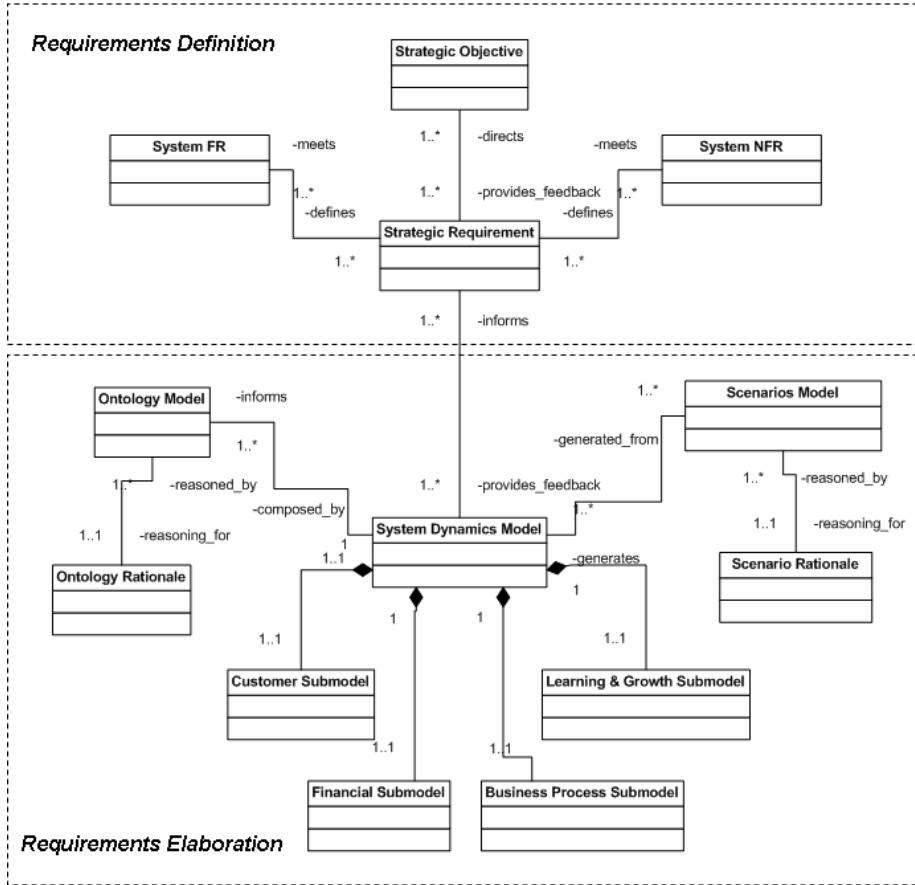
Through these objectives, we aim to provide a framework within which stakeholders can consider the multitude of issues that can impact on different enterprise strategies. Within this framework, we utilise a set of techniques and support tools all of which are well established thus making the framework a practical proposition.

2   A Framework for Modelling and Evaluating Feedback between Requirements and Strategy

The conceptual framework presented in Fig. 1 is aimed at meeting the challenges and objectives presented in section 1.

The definition of requirements involves the identification of (a) business objectives, (b) strategic requirements, (c) functional requirements and (d) non-functional requirements. There is a plethora of methods, techniques and tools that deal extensively with such definitions and hence the focus of this chapter is mainly on requirements elaboration.

Central to the elaboration process is the development of a model that describes the dynamics of the enterprise and its interaction to the proposed system, shown in Fig. 1 is the system dynamics model. The model is intended to describe the feedback between various system components. It is intended to be constructed in such a way so as to be used for testing key system parameters under different conditions and observing the behaviour of the entire system under these conditions. This implies that the model needs to be constructed in such a way so as to permit multiple interpretations of it. To meet these requirements the approach adopts System Dynamics [25-27] as its underlying theoretical baseline. Modelling of behaviour in the system dynamics model can greatly assist in understanding how the system changes over time together with the location and reasons for potential problems with the system. This means that areas for improvement can be identified, new ideas tested and most importantly get an understanding of how a system works without taking any significant risks. Such modelling also assists in reducing biases, uncertainties and conflicts amongst stakeholders together with forming a foundation for the development of scenarios.

Fig. 1. A conceptual framework

Given that any business strategy is likely to be influenced by different perspectives, the approach encourages the development of four model *viewpoints* namely those of customer, financial, business processes and learning and growth, as suggested by the Balanced Scorecard [28]. The Balanced Scorecard provides a framework for translating an organisation's vision and mission into performance indicators. Financial measures of past performance are complemented with measures of the drivers of future performance by including customer, internal business processes and learning and growth perspectives. By forcing senior managers to consider all important operational measures together, the Balanced Scorecard lets them see whether improvement in one area may have been achieved at the expense of another. The use of System Dynamics, which focuses exclusively on feedback structures, provides an excellent vehicle for evaluating such interrelations.

One of the aspects of a system dynamics model is its qualitative dimension, not dissimilar to many other conceptual modelling paradigms in RE. Testing for their validity is mainly based on walkthroughs involving stakeholders, model developers

and potentially facilitators. Experience has shown that stakeholders experience difficulties in comprehending qualitative models and even more significantly, have difficulties in extrapolating from the model to the potential behaviour of the system according to different design options available to them [23]. The absence of parameters, inputs, initial conditions and generally of factors that are needed for testing these qualitative models, greatly diminish their value as tools to understanding phenomena of the world. For without testing of the models it is impossible to comprehend their implications by merely observing, walking through, and debating about their contents. Nor is it always feasible to test them through observations from experimentation in the real world. However, the system dynamics model is complimented by a quantitative dimension, which allows stakeholders to subject the model to simulation. Testing qualitative models through simulation has been approached with some caution by some authors (c.f. [29]) but on the whole the rigorous testing offered by simulation has proved to be of indispensable value [30-32]. Furthermore, Lang shows that qualitative and quantitative properties are not mutually exclusive and both facets are required to support business scenario analysis [33]. Simulation imposes rigorous testing that removes ambiguity, exposes alternatives to stakeholders and effectively removes any affectation towards the models driven by personal biases or political factors.

In order to assist the development of a system dynamics model, the approach makes use of *ontology modelling*. Increasing product complexity, market place globalisation and changes in product life cycles have underlined the need for increased re-use of components, information and knowledge across projects in order to deliver efficient and cost effective product solutions [34]. However, if knowledge is not formally structured to inform decisions during RE, it is easy to overlook important details and leave modelling more prone to conflicts, misunderstandings and incompleteness. The inclusion of ontology in the conceptual framework provides a strategic context for requirements; structures a complex application and clarifies semantics and standards.

The ontology model is structured according to the four perspectives of the Balanced Scorecard. It consists of concepts in the form of superclass-subclass hierarchies of invariant components related to the domain. Classes are comprised of object and/or data type properties and individuals otherwise known as instances. Assertions (e.g. restrictions/constraints) and rules assist in determining relationships between concepts. The *system dynamics model* is informed by the *ontology model* and through its relationship to strategic requirements, it can provide feedback on concepts such as legislative, financial, resource etc that would be of value to the analysis of these requirements and by extension to *business objectives*. The ontology model is captured using Protégé [35].

The *scenarios model* is used to enable stakeholders to visualize the effects of different possible futures on the business and societal environment and determine which requirement option would be the most strategically viable. The system dynamics model is used as the structure upon which alternative scenarios can be generated. Scenarios enable 'what if' questions to be asked in terms of business needs and its aspirations with regards to strategic objectives. Critical variables are defined and behaviour tested according to different values of these variables. Scenarios support and enhance the solution-first strategy in which a provisional design solution is used as a method for identifying requirements. The term 'solution-first strategy' has been

defined as "…a strategy that involves generating a provisional design solution as a method of identifying requirements" [36].

Although scenarios serve as a means for discussing alternative solutions, grounding discussions and negotiations on real examples, and supporting trade-offs among design alternatives [37], a record of reasoning and decisions regarding scenarios is often not present. This results in a lack of traceability. The lack of tools for tracing scenarios can be a hindrance to the process of evaluating alternative futures by stakeholders when they are faced with a large number of scenarios to analyse and evaluate [37]. In order to ameliorate this situation, the framework presented in this chapter deploys a *scenario rationale* component. Coupled to this, there is the *ontology rationale* component whose aim is to record the modelling assumptions when structuring domain and application knowledge.

The rationale adopted takes the form of collaborative visualised argumentation, based on the principles of Rittel and Webber [38]. Within argumentation, rationale consists of the problems and issues that arise in the course of a design, along with pros and cons for each alternative [39]. Rationale is captured using Compendium [40].

3   A Case Study Example

3.1   Overview

Electricity liberalisation is used as a case study to demonstrate the concepts discussed in section 2. It comprises a complex strategic change situation, in which requirements need to be adequately understood and tested to determine which strategic decisions to take in order to be competitive. This particular case study example focuses on the Distribution Business Unit. Distribution is defined as the transport of electricity on medium-voltage and low-voltage distribution systems with a view to its delivery to customers (which include installations). The overall goal of electricity liberalisation ([41] and [42]) was to enter the competition market whilst responding promptly and competently to customer needs and changing market conditions. With privatisation, utilities are now facing up to increasing competition and are under severe pressure to differentiate services by price, quality, and time of use to different types of customer in order to raise profits through heightened operational efficiency [43]. Overall, a number of changes have taken place under liberalisation, such as lower electricity prices, reduction in costs and margins, improvement in labour productivity but also an increase in unemployment [44]. However, more recent statistics show that due to the current economic downturn and rising fuel costs, the cost of electricity continues to rise, with a predicted 40% rise in the UK before the end of 2008 [45]. Levinson and Odlyzko (2007) note that as fuel prices rise and there is intense public opposition to building more power plants and transmission systems as well as concerns about pollution, climate change and fuel depletion, attention is paid to methods that either reduce electricity consumption, or at least shift it away from periods of high loads [46]. In conjunction with this, several clauses in recent European Union Directives refer to the Distribution Business Unit, putting pressure on action with reference to metering of electricity, expressed in Fig. 2 and Fig. 3.

1. Member States shall ensure that, in so far as it is technically possible, financially reasonable and proportionate in relation to the potential energy savings, final customers for electricity, natural gas, district heating and/or cooling and domestic hot water are provided with competitively priced individual meters that accurately reflect the final customer's actual energy consumption and that provide information on actual time of use. When an existing meter is replaced, such competitively priced individual meters shall always be provided, unless this is technically impossible or not cost-effective in relation to the estimated potential savings in the long term. When a new connection is made in a new building or a building undergoes major renovations, as set out in Directive 2002/91/EC, such competitively priced individual meters shall always be provided.

2. Member States shall ensure that, where appropriate, billing performed by energy distributors, distribution system operators and retail energy sales companies is based on actual energy consumption, and is presented in clear and understandable terms. Appropriate information shall be made available with the bill to provide final customers with a comprehensive account of current energy costs. Billing on the basis of actual consumption shall be performed frequently enough to enable customers to regulate their own energy consumption.

3. Member States shall ensure that, where appropriate, the following information is made available to final customers in clear and understandable terms by energy distributors, distribution system operators or retail energy sales companies in or with their bills, contracts, transactions, and/or receipts at distribution stations: (a) current actual prices and actual consumption of energy; (b) comparisons of the final customer's current energy consumption with consumption for the same period in the previous year, preferably in graphic form; (c) wherever possible and useful, comparisons with an average normalised or benchmarked user of energy in the same user category; (d) contact information for consumers' organisations, energy agencies or similar bodies, including website addresses, from which information may be obtained on available energy efficiency improvement measures, comparative end-user profiles and/or objective technical specifications for energy-using equipment.

Fig. 2. *Article 13* - Directive 2006/32/EC [47] relevant statements

2.(d) Encouragement of the adoption of real-time demand management technologies such as advanced metering systems

2.(e) Encouragement of energy conservation measures

Fig. 3. *Article 5* - Directive 2005/89/EC [48] relevant statements

3.2 Business Objectives and Strategic Requirements

In conjunction with these directive statements, five business strategic objectives have been identified and shown in Table 1.

Table 1. Business strategic objectives

Business Strategic Objective No.	Business Strategic Objective Description
SO1	To increase the number of customers
SO2	To increase customer satisfaction
SO3	To increase profits
SO4	To decrease the time to calculate the bill
SO5	To decrease CO_2 emissions

Table 2 shows three high level strategic requirements [49] that indicate stakeholder needs regarding metering in order to achieve efficiency within the Distribution Business Unit and successfully fulfil the business strategic objectives in Table 1.

Table 2. Strategic requirements

Strategic Requirement No.	Strategic Requirement Description
SR1	Improve metering procedures
SR2	Minimise period to calculate customer charges
SR3	Develop computerised mechanisms for energy metering

These strategic requirements could be fulfilled by the possible introduction of new meters. Metering is important, as electricity consumption data is required to calculate customer bills. However it needs to be determined whether the introduction of new meters to fulfill strategic requirements SR1, SR2 and SR3 would be strategically advantageous with reference to the business and societal environment.

As a way of fulfilling requirements SR1, SR2 and SR3, automation of electricity metering is considered in comparison to traditionally read meters. The existing meter reading system uses traditional electro-mechanical accumulation electricity meters, which require a meter reader to travel to each customer's premises every three months in order to manually read the electricity meter, record the data output and submit the readings to the data processing department. However, traditional meter readings are not easily accessible for consumers, the information is displayed in KWh, often shown as a cumulative total, with no ability for the consumer to access historical, or even instantaneous information [50]. Moreover, the maximum accurate reads in the UK can only be four per year even if meters were read accurately every quarter [50]. The current meter reading process is also prone to delay. Consequently customers are paying for electricity consumed many months previously. Large numbers of staff are required to manually read meters, with some premises having difficult to access e.g. rural and agricultural locations. Furthermore traditional meter reading methods have resulted in increased maintenance costs due to outdated equipment. In conjunction with increased rises in fuel and electricity costs there is an increase in the likelihood that electricity meters will be tampered with enabling theft of electricity, producing further business losses.

The proposal to introduce new metering technologies would entail the introduction of automatic meter readings (AMR), the technology used for automatically collecting data from electricity metering devices and transferring it to a central database for analysis and billing. Smart metering has been successfully introduced in the US, Italy, Sweden and Australia. The rationale for automation varied, from being able to read meters in hard to read areas; reducing the cost of meter reads and billing; billing requirements and load shifting [50]. AMR technologies include handheld, mobile and fixed network technologies. These are based on different platforms such as telephony (wired and wireless), radio frequency, power line or advanced transmission. Benefits depend upon the underlying transportation and communication technology, while costs depend on the cost structure of the revenue collection technology and on the burden it imposes on users [46].

From the *customer perspective* installation of an AMR would require access to customer premises to install the new metering device potentially causing disruptions and annoyances. Customer benefits would be realised later, with no access required to properties for meter readings. The AMR would enable the customer to have a clearer picture of electricity usage in the form of management of bills and conservation of electricity consumption. Consumers would be able to monitor real-time energy consumption in cash terms rather than watts of electricity [51] providing a more realistic energy monitoring tool. Management and conservation would also be reflected in the amount of electricity consumed, producing cost savings for the consumer but potential losses in revenue for the business. Indeed many observers think that once the consumer can see the changes in their energy use instantaneously they are much more likely to act to reduce that consumption [50]. Improved electricity conservation would in turn benefit the natural environment. There would also be no need for customers to submit their own meter readings. It could also potentially reduce customer complaints and damage claims resulting from regular visits to customer sites [52]. If linked to an automatic metering management network, the automatic meter can provide the quick reporting of tampered meters so that appropriate action can be taken to both investigate and correct the situation.

From a *financial perspective* benefits could be realised in the form of overall operational cost reductions. However the energy sector is now dragging its feet because the direct financial benefit is limited [53]. There would be an overall decrease in staff salaries for fixed network AMR, but increase in redundancy payouts. However, initial investments would be required for installation and setup. It is anticipated that there would be a greater initial setup cost for fixed AMR in comparison to handheld AMR. However there have been shifts within this paradigm. Electricity Today (2007) highlights that the Tantalus wireless system in the US, which uses public spectrum over a private network for transmission of meter data, provides the performance of two-way telemetry but with a significantly lower cost structure. Currently in the UK, there are no standards relating to automated meters and therefore if a customer changes supplier the meter cannot be read by a rival and the Distribution Business Unit would have to recover its investment in automation.

From an *internal business process perspective*, automation would be less prone to human error particularly in a fixed network. However there would need to be a number of logistical considerations with reference to the installation of the infrastructure. For example the meters can take readings as often as twice a minute, so power companies could have to deal with large amounts of data from millions of customers [54].

Suppliers would therefore have to overhaul their IT systems to cope with the huge increases in data generated by the meters, and this information would have to be cross-referenced with existing systems and stored for up to three years to meet data regulations [51]. Automation would also enable the remote connection and disconnection of electricity. Billing complaint calls could be handled more quickly due to availability of more frequent meter readings following the installation of AMR [52]. However, there is a growing pressure from consumer groups for AMR to be regulated. For example EnergyWatch states that government action will be needed to establish appropriate interoperability standards for the meters themselves and the Energy Retail Association wants a deadline for rollout for the meters in the next decade [53].

From *a learning and growth perspective* fixed network automation does not require meter readers to collect data in the field as it automatically transfers electricity consumption from the automated meter to a central computer system. There would however be a potential increase in staff with the change in systems for installation purposes. Handheld AMR would still require a meter reader to 'walk-by' customer's premises, carrying a handheld computer to collect meter readings. Such staff may also need a small amount of training in the use of new handheld meter reading devices and new office systems. There would be fewer employee injuries, especially in areas with fenced yards, dogs and landscaping [52], reducing the amount paid in compensation. This would be more pronounced in the case of a fixed network AMR.

Table 3. Functional requirements

Functional Requirement No.	Functional Requirement Description
FR1	AMR meter to be permanently installed at each customer's premises
FR2	Meter reading device to remotely read electricity consumption each month or on-demand
FR3	Meter to permanently display real-time prices and electricity consumption to customers
FR4	Communication system to enable the transfer to meter data to the central billing office, e.g. installation of antennas, towers, collectors, repeaters or other permanently installed infrastructure to collect meter readings from AMR and transfer the data to a central computer
FR5	Central office equipment: software to store meter and collection data for each customer
FR6	Central office equipment: an interface to allow for daily transactions with the billing system
FR7	Automation technology to allow for remote reconnection and disconnection of customer electricity
FR8	Handheld computers with a receiver/transceiver to be used by each meter reader to collect meter readings from an AMR capable meter every three months
FR9	Handheld computer data to be automatically transferred to the central billing office software (FR5) for processing

3.3 Automation of Meters: Functional and Non-functional Requirements

To realise fixed network automation a number of functional requirements would need to be considered to support such automation, shown in Table 3.

To realise handheld AMR, FR2 would be replaced with FR8 and FR4 with FR9. To realise fixed network or handheld AMR non-functional requirements (NFR) would be required as shown in Table 4.

Table 4. Non-functional requirements

Non-Functional Requirement No.	Non-Functional Requirement Description
NFR1	Automated meter data to be transferred by secure means
NFR2	Automated meters to be tamper proof
NFR3	Handheld devices to be robust and rugged with ease of handling
NFR4	Automated technologies and infrastructure to be regularly maintained
NFR5	Customer consumption need to be stored securely

Options to be considered for automation are summarised as follows:

1. Continue with *as_is* situation, in which meters are read by traditional means.
2. Introduce handheld automation.
3. Introduce fixed network automation.
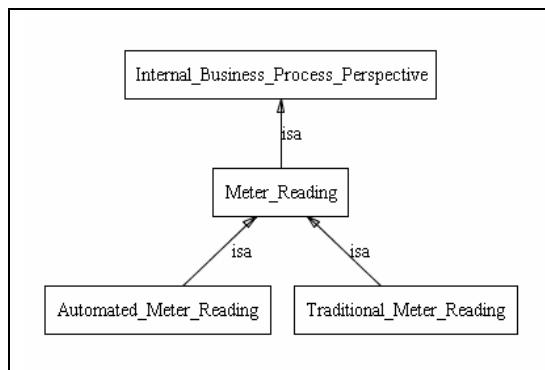
The degree to which automation (options 2 and 3) is introduced would also need to be determined, as a complete change from traditional meter reading to automation may not be suitable to the overall success of the enterprise.

3.4 Automation of Meters: Ontology, Scenarios and Rationale

An example of the ontology classes for the traditional and automated meter reading process are shown in Fig. 4. These conceptual component classes are subclasses of the meter reading class and the meter reading class forms a subclass of the internal business process perspective class. This assists with the organisation of concepts in the ontology, which is used to provide a structure for the System Dynamics model. For example the ontology is comprised of the four perspectives of the Balanced Scorecard and the relationships defined in the assertions section provide the links between perspectives. This structure is then translated to the System Dynamics model.

Class relations, object and data type properties and assertions were stated for each class. An example of three data type property assertions for the traditional meter reading class are as follows:

∃ hasTravelTime has 0.6
∃ hasReadingTime has 4.0
∃ hasUploadReadingTime has 0.2
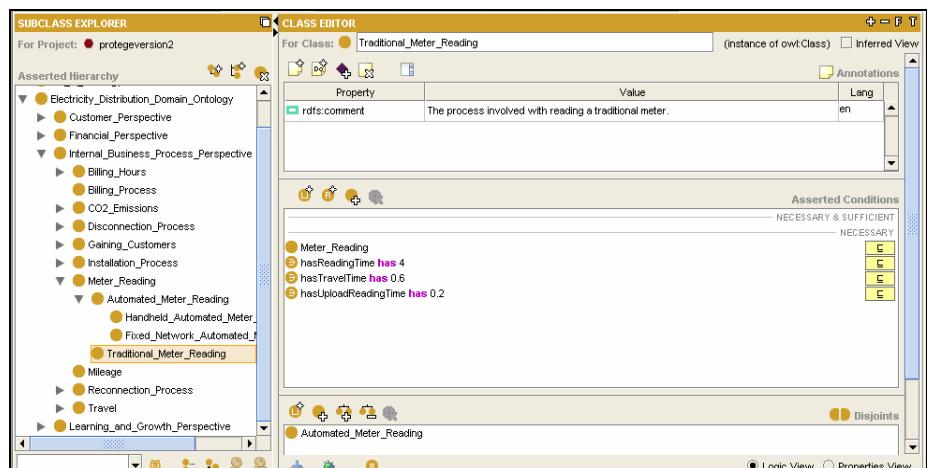
Fig. 4. Ontology classes for meter reading process

These definitions represent a best practice of the time involved to travel, read and upload a customer's meter reading using traditional means, expressed in minutes. The figures for traditional meter reading contrast to handheld automated meter readings shown below:

Э hasTravelTime has 0.3
Э hasReadingTime has 0
Э hasUploadReadingTime has 0.05

It is estimated that the travel time would be halved and the meter reading time, zero, as meter readers would 'walk-by' customer's premises using a handheld device to read the meter. The meter readings would be automatically uploaded rather than manually entered into the computer system.

Fig. 5 shows a screenshot of the ontology in Protégé for the traditional meter reading class. The data type property assertion values are shown in the centre of the figure and the ontology class structure on the left.
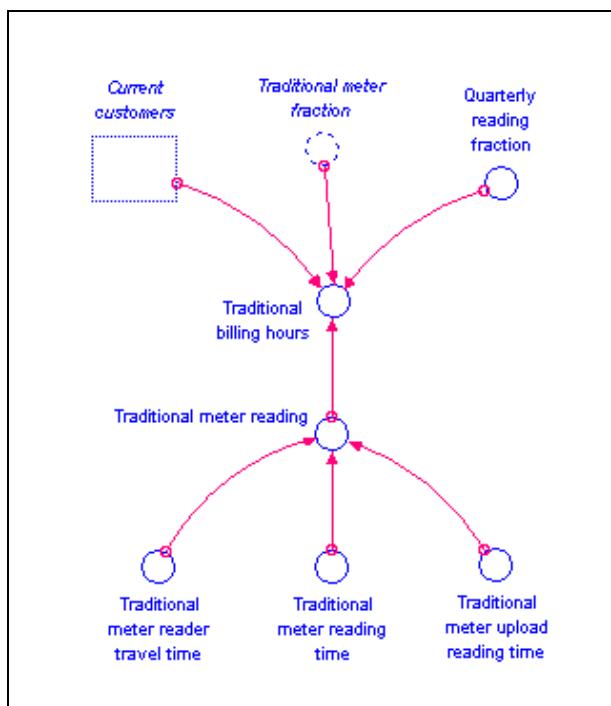
Fig. 5. Screenshot of Protégé ontology - traditional meter reading class
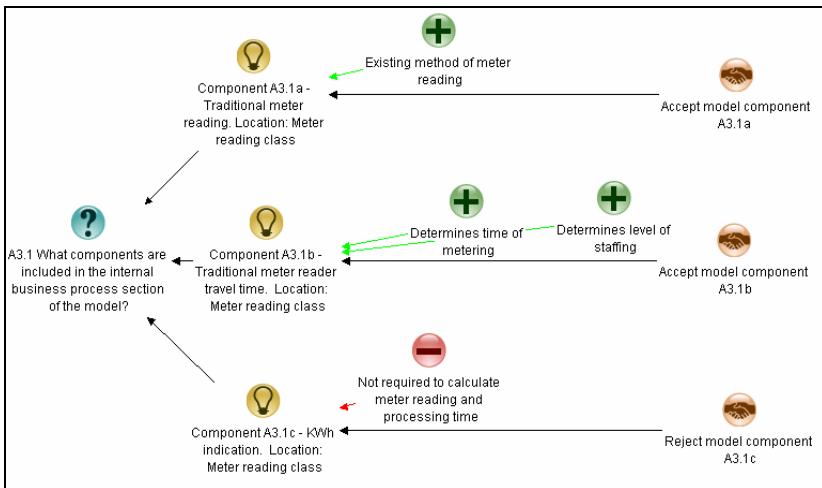
The meter reading classes and properties from the ontology (Fig. 5) are used to compile the model of requirements (Fig. 6). In the ontology for System Dynamics modelling figures of best practice are used to quantify the model components, enabling the System Dynamics model to reflect greater realism and standards. The use of the ontology also allows semantics to be defined (e.g. though classes, property names and descriptions) together with relationships for concepts in the System Dynamics model. This assists in reducing concept misunderstandings. Fig. 6 shows a very small fraction of the system dynamics model for traditional meter reading, which comprises of travel, reading and upload times from the ontology. The billing hours are dependent on the number of traditional meter customers and the number of times that the meter is read each year.

Fig. 7 shows an illustration in Compendium of the modelling rationale for traditional meter reading.

Traditional meter reading and traditional meter reader travel time are both accepted as components in the system dynamics model. It is not necessary to include 'indication of kilowatts', as it is not required for the calculation of meter reading time. Each model component is reasoned about and documented in a similar way during model development. This provides a means for multiple stakeholders to deliberate and negotiate.

Fig. 1. System Dynamics model for traditional meter reading

Fig. 7. Compendium diagram of modelling rationale for traditional meter reading
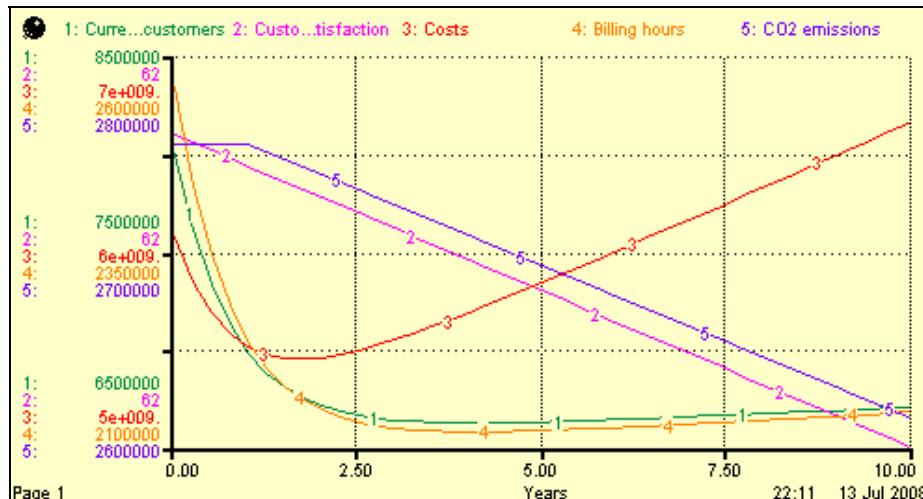
Installation of the infrastructure and meters for automation would take place during the first five years. The implementation of new technologies is reflected in the results over the preceding five years. It is assumed that the Distribution Business Unit has the following starting figures: 8 million customers, 1500 meter readers and 250 billing processing staff. The level of current customers together with those lost to competitors and disconnection is simulated in the customer perspective of System Dynamics model. The learning and growth perspective keeps a record of staffing levels including members of staff that have been lost through redundancy. The CO_2 emissions take vehicle and office energy emissions into consideration. For example vehicle emissions are calculated from the average meter reader mileage against the CO_2 emissions per mile travelled. Investment costs for automation were as follows: handheld automation - €40 per meter and radio-fixed network automation - €80 per meter [55]. These figures include: hardware, software, installation, integration with billing, training and vendor deployment support. Other costing figures included in the financial perspective of the System Dynamics model are as follows: salaries, redundancy payout, meter reader injury compensation, reconnection, disconnection, electricity and electricity theft, maintenance, vehicles, resources and payment method commission.

The following five scenarios were simulated and the behaviour of the system over a 10-year period observed:

Scenario B1.1 – 100% traditionally read meters
Scenario B1.2 – Introduce handheld automation at 50%
Scenario B1.3 – Introduce handheld automation at 100%
Scenario B1.4 – Introduce fixed network automation at 50%
Scenario B1.5 – Introduce fixed network automation at 100%
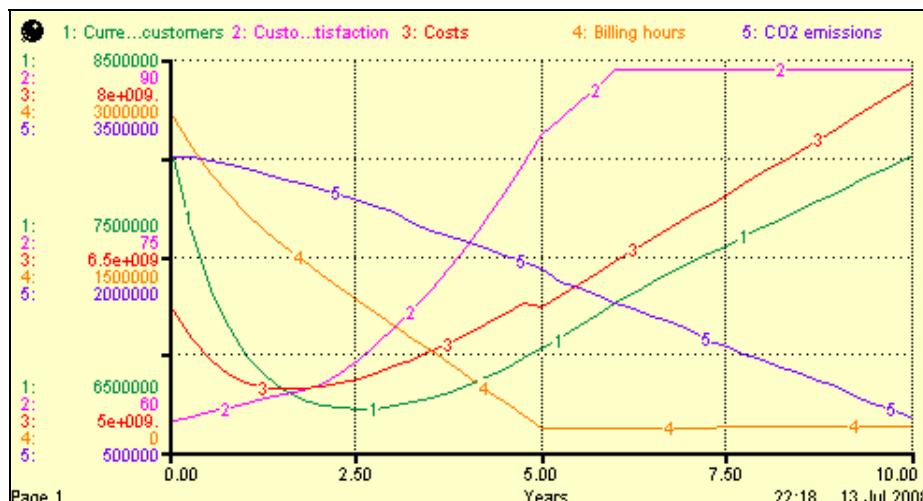
Fig. 8 shows behaviour over time for scenario B1.1 – 100% traditional meters. A decrease in customer satisfaction is shown, which has a consequential effect on reducing customer numbers. Costs continue to increase over a ten year period with ever

increasing maintenance costs required for ageing meters, meter reader salaries and compensation, etc. Billing hours reflect the number of customers and therefore decrease, although when compared to scenarios B1.2 to B1.5, billing hours are the highest. The decrease in CO_2 emissions reflects the natural improvement to vehicle emissions by vehicle manufacturers. Although this figure is also the highest when compared to the other scenarios.

Fig. 8. Behaviour over time for scenario B1.1 – 100% traditional meters

Fig. 9 shows behaviour over time for scenario B1.3 – 100% handheld automation.

Fig. 9. Behaviour over time for scenario B1.3 – 100% handheld automation

Customer numbers see an initial decrease but then begin to increase following the early stages of the introduction of handheld automation. Customer satisfaction increases at a slower pace during the introductory years of automation, followed by a steeper rise and then a 'plateau' when all systems have been fully installed and integrated. Costs reflect installation and setup of the new system and also the rise in customers attracted by a more efficient service. Billing hours reduce significantly during the first five years. The decrease in CO_2 emissions reflects the natural improvement of CO_2 emissions and decreased vehicle usage due to reduced staffing levels, resulting from increased speed of meter reading.

The results for each scenario are shown in the summary table (Table 5). 100% fixed automation shows the largest increase in customer satisfaction and number of customers. With reference to the financial aspect there is an increase in cost for automation in comparison to traditional metering due to the investment costs associated with setting up the new systems.

Table 5. Comparison of statistical results for scenarios

	Scenario B1.1 - 100% traditional meters	Scenario B1.2 - 50% handheld automation	Scenario B1.3 - 100% handheld automation	Scenario B1.4 - 50% fixed automation	Scenario B1.5 - 100% fixed automation
Current customers	6.7 million	7.1 million	8.009 million	7.2 million	8.5 million
Customer satisfaction	62%	76%	90%	77%	91%
Costs (€)	6.5 billion	7 billion	7.6 billion	7.2 billion	8 billion
Profit (€)	1.658 billion	1.659 billion	1.8 billion	1.5 billion	1.4 billion
Billing hours	2.1 million	1.2 million	160,716	1.1 million	53
CO_2 emissions (kgs)	2.6 million	937,300	759,234	237,529	59,463

The reasoning for acceptance or rejection of each scenario is shown in Fig. 10. The scenario rationale shows that scenario B1.3 – 100% handheld automation is accepted as it has the most pro nodes and therefore fulfils the most strategic objectives.

These results are also displayed in Table 6. Therefore implementation of 100% handheld automation would allow strategic requirements SR1, SR2 and SR3 to be accepted.
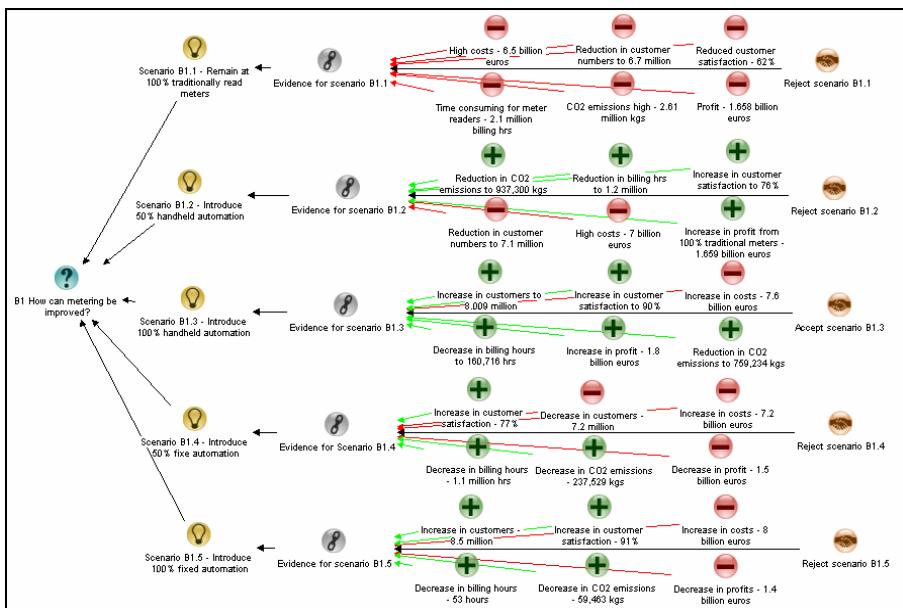
Fig. 10. Scenario rationale for meter reading

Table 6. Alignment of scenarios and business strategic objectives

	Scenario B1.1 - 100% traditional meters	**Scenario B1.2** - 50% handheld automation	**Scenario B1.3 -** 100% handheld automation	**Scenario B1.4 -** 50% fixed automation	**Scenario B1.5 -** 100% fixed automation
SO1 - Increase customer numbers	✗	✗	✓	✗	✓
SO2 – Increase customer satisfaction	✗	✓	✓	✓	✓
SO3 – Increase profits	✗	✓	✓	✗	✗
SO4 – Decrease time to calculate bill	✗	✓	✓	✓	✓
SO5 – Decrease CO_2 emissions	✗	✓	✓	✓	✓

The conceptual framework provides a number of benefits during co-development of requirements. Prior to modelling the ontology provides a strategic context for requirements and enables the structuring of a complex Universe of Discourse, clarifying

semantics and standards of components relevant to the requirements. During modelling ontology provides a knowledge base to inform modelling decisions, potentially reducing misunderstandings and conflicts. Rationale facilities discussion about model components and keeps a record of reasoning regarding model construction, taking multiple stakeholders needs into consideration. Ontology and rationale working together provide knowledge and reasoning to support the modelling of requirements. Following modelling, simulation of requirements alternatives allows the testing of strategic viability. Scenario rationale enhances this by documenting decision-making regarding different alternatives.

4 Conclusion

Requirements Engineering is considered by many as the most critical of all development activities for socio-technical systems. The sensitive area of early requirements is only recently beginning to be addressed in a methodological sense. Considerable effort is required to bridge the semantic islands that are often formed between different communities of client stakeholders, designers, regulators, etc. Indeed the entire system development process seems to be disadvantaged by lack of techniques to assist with effective communication [56, 57]. An in-depth study on industrial practice [58] provides evidence that communication is crucial to the entire design process.

In early requirements, when there is a great deal of vagueness and uncertainty about system goals that are often set against a background of social, organizational and political turbulence, the need for a systematic and systemic way of dealing with all co-development aspects seems to be of paramount importance.

Whilst qualitative-based conceptual modelling approaches seem to be an improvement on purely linguistic-based approaches, they fail to bridge the communication gap between client stakeholders and analysts. The issue of analyst-client relationship has been highlighted by many authors [59, 60]. This type of modelling paradigm that has evolved from work on Databases, Software Engineering or Object-oriented Design, with its analyst orientation, does little to enhance communication.

This chapter has presented a framework within which a set of techniques provides capabilities for improving the understanding of the interrelations between strategic goals and system functionality.

References

1. IEEE Computer: Special Issue on Requirements Engineering. IEEE Computer 18(4) (1985)
2. Davis, A., Hsia, P., Kung, D.: Status Report on Requirements Engineering. IEEE Software 10(6), 75–79 (1993)
3. IEEE-Std.'830', IEEE Guide to Software Requirements Specifications, The Institute of Electrical and Electronics Engineers, New York ANSI/IEEE Std 830-1984 (1984)
4. Loucopoulos, P., Karakostas, V.: System Requirements Engineering, 1st edn. McGraw Hill, London (1995)
5. TSE: Special Issue on Requirements Engineering. IEEE Transactions on Software Engineering (1977)

6. Nuseibeh, B., Easterbrook, S.: Requirements Engineering: A Roadmap. In: Finkelstein, A. (ed.) The Future of Software Engineering. ACM Press, New York (2000)
7. Gantz, J.F., Chute, C., Manfreditz, A., Reinsel, D., Schlichting, W., Toncheva, A.: The Diverse and Exploding Digital Universe. IDC (2008)
8. Leibold, M., Probst, G., Gibbert, M.: Strategic Management in the Knowledge Economy: New Approaches and Business Applications. Publicis Corporate Publishing and Wiley-VCH-Verlag GmbH & Co KGaA, Erlangen (2002)
9. Malone, T.W., Laubacher, R., Morton, M.S.S.: Inventing the Organizations of the 21st Century. MIT Press, Cambridge (2003)
10. Rouse, W.B.: A Theory of Enterprise Transformation. Systems Engineering 8 (2005)
11. Melao, N., Pidd, M.: A conceptual framework for understanding business processes and business process modelling. Information Systems Journal 10, 105–129 (2000)
12. Yu, E., Mylopoulos, J.: Why Goal-Oriented Requirements Engineering. In: Fourth International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ 1998), Pisa, Italy (1998)
13. Alexander, I., Maiden, N.A.M.: Scenarios, Stories, Use Cases Through the System Development Life-Cycle. John Wiley and Sons Ltd., London (2004)
14. Telelogic: Telelogic DOORS (2007)
15. Loucopoulos, P.: System Co-Development Through Requirements Specification. In: Vasilecas, O., Wojtkowski, W., Zupančič, J., Caplinskas, A., Wojtkowski, W.G., Wrycza, S. (eds.) Information Systems Development: Advances in Theory, Practice and Education, pp. 1–13. Springer, US (2005)
16. Nuseibeh, B.: Weaving Together Requirements and Architectures. IEEE Computer 34, 115–117 (2001)
17. Pohl, K., Sikora, E.: The Co-Development of System Requirements and Functional Architecture. In: Krogstie, J., Opdahl, A.L., Brinkkemper, S. (eds.) Conceptual Modelling in Information Systems Engineering, pp. 229–246. Springer, Heidelberg (2007)
18. Bleistein, S.J., Aurum, A., Cox, K., Ray, P.K.: Strategy-Oriented Alignment in Requirements Engineering: Linking Business Strategy to Requirements of e-Business Systems using the SOARE Approach. Journal of Research and Practice in Information Technology 36, 259–276 (2004)
19. Bleistein, S.J., Cox, K., Verner, J., Phalp, K.T.: Requirements Engineering for e-Business Advantage. Requirements Engineering Journal 11, 4–16 (2006)
20. Forrester, J.W.: Nonlinearity in High-Order Models of Social Systems. European Journal of Operational Research 30, 104–109 (1987)
21. Schön, D.A.: The Reflective Practitioner: How Professionals Think in Action, 2nd edn. Basic Books, New York (1983)
22. Sterman, J.D.: All Models are Wrong: Reflections on Becoming a Systems Scientist. System Dynamics Review 18, 501–531 (2002)
23. Loucopoulos, P., Zografos, K., Prekas, N.: Requirements Elicitation for the Design of Venue Operations for the Athens 2004 Olympic Games. In: Proceedings of the 11th IEEE International Conference on Requirements Engineering, Monterey Bay, CA, pp. 223–232 (2003)
24. Gehry, F.O.: Reflections on Designing and Architectural Practice. In: Boland, R.J., Collopy, F. (eds.) Managing as Designing. Stanford University Press, Stanford (2004)
25. Forrester, J.W.: Designing the Future. Universidad de Sevilla 15, Sevilla, Spain (1998)
26. Sterman, J.: Business Dynamics: Systems Thinking and Modeling for a Complex World. Irwin/McGraw-Hill, Boston (2000)

27. Morecroft, J.: Strategic Modelling and Business Dynamics. John Wiley & Sons Ltd., Chichester (2007)
28. Kaplan, R.S., Norton, D.P.: The Balanced Scorecard - Measures that Drive Performance. Harvard Business Review 70, 71–79 (1992)
29. Lane, D.C.: With a Little Help From Our Friends: How System Dynamics and Soft OR Can Learn From Each Other. System Dynamics Review 10, 101–134 (1994)
30. Homer, J.B.: Why We Iterate: Scientific Modeling in Theory and Practice. System Dynamic Review 12, 1–19 (1996)
31. Barlas, Y., Kanar, K.: Structure-Oriented Behavior Tests In Model Validation. In: 18th International Conference of the System Dynamics Society, Bergen, Norway, pp. 33–34 (2000)
32. Eddins, W.R., Crosslin, R.L., Sutherland, D.E.: Using Modelling and Simulation in the Analysis and Design of Information Systems. In: Sol, H.G., van Hee, K.M. (eds.) Dynamic Modelling of Information Systems, pp. 89–119. Elsevier, Amsterdam (1991)
33. Lang, K.: Simulation of Qualitative Models To Support Business Scenario Analysis. In: 18th International Conference of the System Dynamics Society, Bergen, Norway, p. 122 (2000)
34. Ratchev, S., Urwin, E., Muller, D., Pawar, K.S., Moulek, I.: Knowledge based Requirements Engineering for One-of-a-Kind Complex Systems. Knowledge Based Systems 16, 1–5 (2003)
35. Protégé: The Protégé Project (2008), http://protege.stanford.edu
36. Carroll, J.M.: Scenarios and Design Cognition. In: IEEE Joint International Conference on Requirements Engineering (RE 2002), Essen, Germany, pp. 3–5 (2002)
37. Weidenhaupt, K., Pohl, K., Jarke, M., Haumer, P.: Scenario Usage in System Development: A Report on Current Practice. IEEE Software 15, 34–45 (1998)
38. Rittel, H.W.J., Webber, M.: Dilemmas in the General Theory of Planning. Policy Science 4, 155–169 (1973)
39. Shipman, F.M., McCall, R.J.: Integrating Different Perspectives on Design Rationale: Supporting the Emergence of Design Rationale from Design Communication. AIEDAM 11, 141–154 (1997)
40. Compendium-Institute (2008), http://compendium.open.ac.uk/institute
41. European Union: Directive 96/92/EC of the European Parliament and of the Council of 19 December 1996 concerning common rules for the internal market in electricity. Official Journal of the European Union L027, 20–29 (1997)
42. European Union: Directive 2003/54/EC of the European Parliament and of the Council of 26 June 2003 concerning common rules for the internal market in electricity and repealing Directive 96/92/EC. Official Journal of the European Union L176, 37–56 (2003)
43. Guy, S., Marvin, S.: Pathways to 'Smarter' Utility Meters: the Socio-Technical Shaping of New Metering Technologies. Department of Town and Country Planning, University of Newcastle upon Tyne Working Paper No. 56 (1995)
44. Ernst &Young: Research Project for the Department of Trade and Industry on The Case for Liberalisation, London (2006)
45. Pagnamenta, R., Kennedy, S.: Consumers Face up to 40% Rise in Energy Bills as Gas Price Soars. The Times, London (2008)
46. Levinson, D., Odlyzko, A.: Too Expensive to Meter: The Influence of Transaction Costs in Transportation and Communication. In: Royal Society Discussion Meeting on Networks: Modelling and Control, London (2007)

47. European Union: Directive 2006/32/EC of the European Parliament and of the Council of 5 April 2006 on energy end-use efficiency and energy services and repealing Council Directive 93/76/EEC. Official Journal of the European Union L114, 64–85 (2006)
48. European Union: Directive 2005/89/EC of the European Parliament and of the Council of 18 January 2006 concerning measures to safeguard security of electricity supply and infrastructure investment. Official Journal of the European Union L033, 22–27 (2006)
49. ELEKTRA-Consortium: ELEKTRA Electrical Enterprise Knowledge for Transforming Applications, Demetra: System Design Specification for PPC (1998)
50. Porter, H.: Smart Metering - The Real Energy Benefits. In: International Energy Efficiency in Domestic Appliances and Lighting Conference 2006 (EEDAL 2006), London (2006)
51. Brown, J.: Utilities Query Smart Meter Plan. Computing (2006a)
52. Electricity-Today: Evolving Technologies in AMR. Electricity Today, 24–26 (2007)
53. Friedlos, D.: Smart Meters need Backing. Computing (2007)
54. Brown, J.: Can Smart Meters Add Up? Computing (2006b)
55. Plexus-Research: Deciding on 'Smart' Meters: The Technology Implications of Section 1252 of the Energy Policy Act of 2005 (2006)
56. Coughlan, J., Macredie, R.D.: Effective Communication in Requirements Elicitation: A Comparison of Methodologies. Requirements Engineering 7, 47–60 (2002)
57. Urquhart, C.: Analysts and Clients in Organisational Contexts: A Conversational Perspective. Strategic Information Systems 10, 243–262 (2001)
58. Curtis, B., Krasner, H., Iscoe, N.: A Field Study of the Software Design Process for Large Systems. In: Jarke, M. (ed.) Meta Models for Requirements Engineering [Nature Team] (1988)
59. Kennedy, S.: Why Users Hate your Attitude. Informatics, 29–32 (1994)
60. Bashein, B., Markus, M.L.: A Credibility Equation for IT Specialists. Sloan Management Review 38, 35–44 (1997)

Managing Legal Texts in Requirements Engineering

Paul N. Otto and Annie I. Antón

Department of Computer Science,
North Carolina State University
{pnotto,aianton}@ncsu.edu

Abstract. Laws and regulations are playing an increasingly important role in requirements engineering and systems development. Monitoring systems for requirements and policy compliance has been recognized in the requirements engineering community as a key area for research. Similarly, legal compliance is critical in systems development, especially given that non-compliance can result in both financial and criminal penalties. Working with legal texts can be very challenging, however, because they contain numerous ambiguities, cross-references, domain-specific definitions, and acronyms, and are frequently amended via new statutes, regulations, and case law. Requirements engineers and compliance auditors must be able to identify relevant legal texts, extract requirements and other key concepts, and monitor compliance. This chapter surveys research efforts over the past 50 years in handling legal texts for systems development. This survey can aid requirements engineers and auditors to better specify, test, and monitor systems for compliance.

Keywords: legal requirements, legal compliance.

1   Introduction

Requirements for software systems are increasingly originating in laws and regulations. For example, numerous laws and regulations have emerged in the past fifteen years regarding data privacy. In the United States, privacy requirements have been elaborated by laws and regulations governing particular industries, such as the Health Insurance Portability and Accountability Act (HIPAA), which governs patient health records, and the Gramm-Leach-Bliley Act, which governs financial institutions. In Europe, the European Union passed the EU Directive on Data Protection, which details privacy requirements for all organizations operating in Europe. Several other countries, such as Canada with its passage of the Personal Information Protection and Electronic Documents Act, have enacted comprehensive privacy requirements cutting across all private-sector industries.

The need for system developers to monitor systems for both requirements and policy compliance has been identified as a challenging and important problem

in the requirements engineering community [43]. According to recent surveys of senior information security professionals, legal compliance has been the primary driver of information security policy for the past three years [18]. Requirements engineers and system developers currently face two major problems in assessing legal compliance: (1) determining the applicable regulations, and (2) creating the requirements and policies necessary to achieve compliance with those regulations [25]. Methodologies for monitoring compliance with requirements and policies currently are not available to developers [43]. And yet, stakeholders need to better understand the regulations that govern the systems for which they are responsible and require precise answers to specific queries about what is allowed and what is not allowed [4][37]. The penalties for non-compliance can be severe; for example, HIPAA specifies up to $250,000 and 10 years in prison for criminal violations.

For requirements engineers, access to specific laws and regulations has become easier with the push towards online access for legal texts occurring in some countries. However, an organization must still identify the regulations relevant to its specific system before it can even begin to assess its compliance with the law. Once the relevant laws and regulations are identified, extracting requirements from legal texts is still a difficult and error-prone process [48]. In addition, an organization must still engage in traditional software engineering activities (e.g., analysis, modeling, development) as well as traditional security activities (e.g., policy enforcement and auditing) in order to properly implement legal compliance processes [15].

This chapter surveys research efforts over the past 50 years in modeling and using legal texts for system development. The survey identifies the strengths and weaknesses of each approach, and based on analysis of the literature to date as well as our prior experiences in analyzing policy and regulations [14][20][39], we propose a broad set of requirements for tool support that would aid requirements engineers and compliance auditors alike. It is our hope that these requirements will prompt serious consideration by the requirements engineering community, as it is within this community that we believe significant progress can be made to address the challenges related to legal compliance in software systems. The treatment of legal texts requires consideration of the business, organizational, and community context surrounding the development process in order for requirements engineers to appropriately address legal requirements.

The remainder of this chapter is organized as follows. Section 2 discusses the nature of legal texts, noting the various characteristics that make such texts difficult to work with. Section 3 discusses the three key layers of law relevant to requirements engineers in working with legal texts. Section 4 analyzes various efforts from the past 50 years in modeling regulations, extracting key concepts, and using legal texts in system development. Based on our extensive review of prior work, Section 5 proposes a set of broad requirements for comprehensive systems to assist requirements engineers and auditors with regulatory compliance tasks. Finally, Section 6 discusses the analysis and outlines future work needed to realize such systems.

2   The Nature of Legal Texts

There are certain characteristics of laws and regulations that make them both useful and difficult to apply to design methodologies. Legal texts tend to be very structured and hierarchical documents. However, legislatures and agencies at the federal, state, and local level can all specify new laws and regulations, and these legal texts may complement, overlap, or even contradict one another due to differing objectives and changes over time [26]. In addition, amendments and revisions to the same provision of a legal text can lead to internal contradictions [4]. As a result, some areas of law undergo constant changes, whereas other areas are relatively stable [6].

Another important characteristic of laws and regulations is the frequent references to other sections within a given legal text and even to other legal texts. Much of the prior work in computer science examining laws and regulations has noted the difficulty of handling these numerous cross-references within legal texts (e.g., [7][14][25]). These cross-references force requirements engineers to spend additional time reading and understanding legal texts before they can even begin to extract key concepts or apply the legal texts to system design. May et al. employ a methodology to derive formal models from regulations that they applied to the HIPAA Privacy Rule [35]. In their study, discussed in Section 4.6, they assume that external and ambiguous references are satisfied by default [35]. This contradicts our own study of the HIPAA Privacy Rule [14], discussed in Section 4.3, in which we discovered that cross-references introduce important constraints from other sections that restrict which rules apply in different situations and contexts.

If references to other sections of a particular legal text or other external legal texts are unaccounted for, requirements engineers are prone to make interpretations and inferences that are inconsistent with the law. Such assumptions will inevitably lead to overlooking important exceptions or priorities and ultimately lead to non-compliance. Traceability within the context of legal systems takes on a far greater significance than we already afford it in the requirements engineering community because legal traceability is supercharged, so to speak, with priorities and exceptions that govern special cases (e.g., which information can be accessed, when such access is allowed). Thus, the ability to manage cross-references and maintain traceability from the originating law, regulation, or policy to the relevant requirements must be addressed in any system for supporting requirements engineers and/or compliance auditors.

Laws and regulations typically specify a large number of relevant definitions and acronyms, further complicating the job of requirements engineers and system designers [25]. Along with cross-references, such extensive definitions necessitate a significant amount of domain knowledge before the legal texts are comprehensible and usable. When spread across multiple texts that may have overlapping, inconsistent, or contradictory terms, the domain-specific lexicon significantly raises the barrier to entry for developers hoping to build legal compliance into their software systems. A more fundamental problem in dealing with laws and regulations is the fact that legal texts are laden, often by design, with

ambiguities. For example, § 164.306(a)(2) in the HIPAA regulations requires organizations to "protect against any reasonably anticipated threats or hazards to the security or integrity" of protected health information; the section does not define what constitutes reasonable anticipation. Researchers frequently note the difficulty in identifying and resolving such ambiguities in legal texts (e.g., [1][14][30][45]).

Researchers have examined the problem of ambiguity in natural language as it affects requirements engineering. Kamsties characterizes ambiguity as more problematic than other forms of defects that appear in requirements [23]. Kamsties et al. discuss various types of linguistic and requirements engineering-specific, context-dependent ambiguities [24]. They also note that using formal methods to remove ambiguities from natural language representations simply results in an unambiguously wrong specification [24]. Moreover, such formal specifications are inaccessible to the majority of stakeholders and their formality makes it more difficult to discover ambiguities than in a natural language representation [24]. Berry et al. present a guide to disambiguating text for requirements engineers and lawyers alike [8].

There are several different classifications of ambiguities. A simple dichotomy of ambiguities consists of those that are intentional—to allow the law to be generalized—and those that are unintentional [1]; the above example from the HIPAA regulations likely represents an intentional ambiguity. Additional categorizations include: high-level classifications (e.g., implication-coimplication, disjunctive-conjunctive, ambiguity of reference) [1]; categorization within specific domains, such as software engineering, linguistics, and law [8]; a taxonomy of ambiguities occurring in requirements [23]; a distinction between nocuous and innocuous ambiguities [16]; and specific types of ambiguities uncovered during empirical analysis (e.g., conjunctions, under-specifications) [14]. Just as courts and administrative agencies must struggle to interpret the law when ambiguities are present, so must users, be they requirements engineers or developers, make crucial interpretation decisions during requirements gathering and software design.

3   Three Key Layers of Law

In considering the law's impact on requirements engineering, three broad classifications of source documents emerge for consideration during the development process: statutory text, supplemental information, and case or administrative interpretations of the text.

3.1   Statutory Language

The specific text of a law or regulation often specifies requirements directly. For example, HIPAA contains low-level system requirements governing the implementation of specific security concerns, such as how passwords are handled. Often such requirements may not be evident through a cursory reading of the

legal text, and thus various techniques may be necessary in order to extract relevant requirements for software system design.

Due to the nature of legal texts, as discussed in Section 2, it is not possible for requirements engineers to consider only the statutory language to adequately address legal requirements. As with requirements more generally, the law is bound up in context. The complexity surrounding statutory language means that requirements engineers cannot rely merely on the wording of the legal text, but instead must consult at least one of the other categories of source documents.

3.2   Supplemental Information

Supplemental information can provide rich insight into a legal text's purpose and meaning. Laws and regulations often are accompanied by guiding documents on how to interpret and use the legal texts. Such supplemental information may include reference handbooks or other published guides to interpreting the text [25]. The passage of most laws and regulations is accompanied by a detailed legislative or administrative history that can illuminate the purpose and intent of specific provisions of a given legal text. For example, U.S. laws generally undergo committee hearings and reports before passage by the legislature, whereas U.S. regulations often involve agency comment and review periods before the regulation becomes enforceable. These drafting and review documents thus can provide guidance to how a legal text is to be read and understood.

Ambiguity associated with regulations has compelled government agencies to provide detailed reference materials and instructive handbooks to improve understanding and compliance efforts [29]. For example, the U.S. Department of Health and Human Services publishes a summary of the HIPAA Privacy Rule and guidance documents for implementing the HIPAA Security Rule. Additional supplemental guides are created by organizations separate from the government agencies that actually promulgate regulations [30]. This large, diverse set of documentation can be crucial for software developers who are attempting to identify regulatory compliance requirements early in the design process. However, requirements engineers must be careful when using these supplemental documents, as they do not have the same legal standing and may even contain misinterpretations of the original regulatory text.

3.3   Administrative and Case Law

The third layer of the law arises through case and administrative rulings interpreting legal texts.. The amount and influence of interpretative rulings on any given legal text varies widely. Some areas of law, such as tax law, are well-settled and have a large body of case law; as such it is possible to classify most cases as 'routine' [22]. Other areas, such as information security and data privacy law, are still emerging fields and are therefore subject to greater fluctuation in the requirements stemming from laws and regulations; as such, very little case or administrative law exists to guide requirements engineers in interpreting legal texts governing security or privacy concerns.

Administrative and case law presents difficult problems for requirements engineers in both understanding context and managing complexity. For example, requirements engineers may need to consult with their organization's legal counsel in order to establish interpretations of a given ruling, due to complexities such as the authority of a given ruling (e.g., binding, persuasive) and whether all or part of the ruling is still good law. Furthermore, rulings may establish new requirements directly as part of an interpretation of a given legal text. Therefore, use of administrative and case law requires changes in the development process to accommodate legal and organizational contexts that may traditionally have been considered separately from engineering tasks.

4   Survey of Work with Legal Texts

This section examines various approaches for modeling legal texts, extracting key concepts from the language of legal texts, and creating compliance-checking systems. This survey followed the principles of systematic review as detailed by Kitchenham [27]. The survey sought to identify, "what efforts have been made to model legal texts for use in requirements engineering and system development?" Briefly, the survey methodology entailed: (1) identifying potentially relevant research through use of the ACM and IEEE databases; (2) following citations to uncover additional papers for review, which resulted in the discovery of over 150 relevant publications; and (3) careful analysis and review, resulting in the selection of 39 papers in the following discussion, organized into ten categories.

4.1   Symbolic Logic

One of the earliest attempts to model legal texts involved the use of symbolic logic, also known as mathematical logic. The approach attempted to balance the benefits of natural language with the rigor of symbolic logic [1], serving as a precursor for later efforts to provide both human- and machine-readable interpretations. Allen's technique employed six key logical connectives: implication, conjunction, coimplication, exclusive disjunction, inclusive disjunction, and negation [1]. By identifying the logical connectives, one could largely eliminate the unintended ambiguities present in legal texts by using a more mathematical representation. This effort, while noteworthy in its systematic legal representation, did not leverage the processing and data manipulation capabilities of computers; it sought to answer specific queries and make legalistic determinations, rather than shape requirements gathering or systems development.

4.2   Knowledge Representation

Numerous approaches to representing legal texts as computer programs began in the late 1970s, largely based on logic programming techniques. These knowledge representation efforts were based on the premise that a model of legal texts

should closely parallel the language of those texts [10]. As such, most of these approaches used Prolog—a logic programming language targeted for knowledge representation and expert systems—to represent the legal rules extracted from laws and regulations. Specific efforts included: TAXMAN, modeling the U.S. Internal Revenue Code [36]; representing the British Nationality Act as a logic program [45]; modeling the Income Tax Act of Canada [46]; representing the U.K. welfare law as a logic program [7]; ESPLEX, a logic system for representing legal rules [10]; capturing the Indian Central Civil Service Pension Rules in logic [44]; and modeling the Dutch Opium Act [41]. Each of these knowledge representation techniques could aid requirements engineers in understanding legal texts and answering specific queries during requirements elicitation.

Knowledge representations of legal texts afford certain advantages to system developers and policy-makers alike. Logical representations enable users to identify unintended ambiguities in a legal text [45]. This allows requirements engineers to pinpoint specific ambiguities and resolve those issues before system development commences. It also allows policy-makers to address these ambiguities in future amendments to laws and regulations. Developers can use expert systems to make specific queries when issues arise regarding compliance or design decisions. Such targeted queries enable developers to resolve known compliance issues with the relevant legal texts.

Several characteristics of these systems limit the generalizability or applicability of this research to legal texts. The knowledge representation approach has focused primarily on either well-settled areas of law or regulations with minimal accompanying case law. Most of the projects were conducted as case studies and, to the best of our knowledge, no final product or working system ever resulted from the research. The goal often was to answer specific queries or handle what-if scenarios; none of these early efforts used the model to influence system development or check for compliance. These knowledge representation approaches had no degree of automation: for each new legal text, a user would be required to manually extract the legal rules and encode them in logical clauses. Finally, the research efforts referenced above, with the exception of [41], make no mention of providing traceability between the representation in logic and the original legal text. As previously discussed, this lack of traceability creates compliance vulnerabilities as law evolves via case law, amendments to the original legal text, or additional laws and regulations coming into force. These drawbacks combine to make knowledge representation techniques problematic and very limiting for software developers who need to extract requirements and system design elements directly from legal texts.

A more recent variation on logic programming efforts employs event calculus to track changes in legal texts over time [32]. The approach uniquely captures the frequent changes associated with legal texts, enabling users to model and understand how the law changed across revisions [32]. Martinek and Cybulka create a knowledge base maintaining information for when changes are made to regulatory texts [32]; this provides a limited measure of traceability for developers evaluating changes over time. The approach provides a unique look at the

dynamic nature of legal texts, but does not address the same aforementioned shortcomings facing other logic programming implementations.

4.3   Deontic Logic

Another logic-based approach to modeling legal texts involves the use of deontic logic to capture the rights and obligations present in the law. The impetus for this approach is that "the law is like a programming language controlling a society ...[where] observations must be made, calculations performed, records kept and messages transmitted" [47]. Extracting specific rights and obligations from legal rules permits the creation of a knowledge base, as was possible with the logic programming efforts discussed in Section 4.2, to model the key elements of legal texts and answer directed user queries. The major deontic logic efforts include: LEGOL, a formal LEGally Orientated Language for capturing obligations [47]; ON-LINE, an ONtology-based Legal INformation Environment for capturing and analyzing legal texts as legal knowledge [49]; work establishing the legal importance of monitoring permissions as well as obligations [11]; and systems for automated extraction of normative references from legal texts [9][40].

Deontic logic approaches have not yet met users' needs for working with legal texts and ensuring compliance. By extracting rights and obligations, deontic logic systems disambiguate legal texts and make them more palatable for system designers. Early work established the utility of such an approach, but a user was still required to manually encode the law into the deontic operators for rights and obligations [47][49]. The ON-LINE system was able to deal with only small sections of legislation at a time and usability of the ontology-based approach proved problematic during usability testing [49]. More recent efforts include automated extraction of normative references—such as specific rights and obligations—detailed in a legal text, and addressed the problem of the law's evolution by tracking changes over time [9][40]. This provides for some degree of traceability, as the system maintains information on each extracted element—including its type, number, date, section and subpart headers—and the normative references [40]. However, these more recent projects were not completed, and there are few examples to illustrate the effectiveness of such an approach. While these research efforts established deontic logic as a worthwhile approach to extract key information from legal texts, they did not result in usable tools for developers to influence system design or monitor compliance.

A more recent deontic logic implementation involves the explicit extraction and balancing of rights and obligations from regulations [14]. The research focuses on providing requirements monitoring and compliance support for system developers and maintainers [12]. Semantic parameterization entails identifying the ambiguities within a legal text and balancing the extracted rights and obligations [12]. This decomposition of regulations enables the user to identify both explicit and implied rights and obligations [12]; capturing these implied rights and obligations is not addressed by the other deontic logic approaches. The process, however, requires manual extraction of the rights, obligations, delegations, and constraints. Unlike most other approaches, Breaux and Antón maintain traceability across all

artifacts from the HIPAA section and paragraph number to the corresponding software requirements and access control rules. This approach has been tested on only a part of the HIPAA Privacy Rule; as such, its scalability and applicability to other domains is not yet validated.

4.4   Defeasible Logic

Defeasible logic provides an alternative logic-based approach to modeling legal texts. Defeasible logic is a form of non-monotonic skeptical reasoning, wherein there are strict rules, defeasible rules, and defeaters. Strict rules always hold, while defeasible rules hold true unless an exception, or defeater, exists for the rule. Given the existence of overlapping and conflicting legal texts at different levels of government, defeasible logic appears to be a natural fit for modeling such texts [3]. The practical use of defeasible logic in routine legal practice is emphasized as a key advantage for system developers and users of legal texts [22]; defeasible logic can aid in both decision support and legal reasoning [4].

Proponents of defeasible reasoning also have noted that deontic logic will not capture all eight fundamental legal conceptions [21]: right, no-right, privilege, duty, power, disability, immunity, and liability [19]. Hohfeld presented these fundamental legal conceptions as the basic elements needed to understand any legal relation, noting specifically that 'rights' and 'duties' (obligations) were insufficient to address the complexities in many areas of law [21].

Antoniou et al.'s approach has yielded an operational implementation of a defeasible logic system [3], but there remain several disadvantages to such an approach for modeling legal texts and monitoring compliance. For example, numerous features need to be added to any 'pure' defeasible logic implementation (e.g., representing hierarchies, arithmetic and temporal operators, and capturing underlying legal knowledge) to model all of the law's nuances [3]. The computational complexity of a defeasible logic system is in dispute: early research touted low complexity as a major advantage [4], whereas more recent research indicated that approximating a model was necessary due to concerns about complexity [22]. Again, these efforts in defeasible logic make no mention of maintaining traceability and provide no examples of directly modeling legal texts. Given the lack of follow-up on defeasible logic approaches, the viability of such systems remains uncertain. It appears that no system is available to leverage defeasible reasoning in requirements engineering and compliance monitoring.

4.5   First-Order Temporal Logic

Barth et al. proposed using first-order temporal logic to extract key concepts—context, roles, type of information—rather than precisely modeling a legal text [5]. The approach, which is based on a conceptualization of privacy using the contextual integrity framework [38], captures only the privacy-related elements of regulations such as parts of HIPAA [5]. The use of formal logic is reminiscent of other logic-based approaches, but the narrower focus on privacy limits the applicability of this approach to other regulations. Preliminary results show that

the contextual integrity framework captures most privacy elements from the regulations tested to date; however, Barth et al. do not disclose what percentage of privacy elements originally present in the text were extracted using their framework [5].

Barth et al. establish the framework's viability in assessing compliance between privacy policies and the privacy provisions of regulations. However, a major limitation of this approach for the requirements engineer is that Barth et al. make no mention of maintaining traceability between the extracted concepts and the original regulatory text. Although this approach may be capable of aiding developers in evaluating system requirements and design vis-à-vis privacy regulations, its narrow framework does not appear to extend to other legal texts. Unlike many of the earlier research projects discussed in this section, this framework may soon be available to other researchers for validation and extension.

4.6   Access Control

Another approach to modeling legal texts employs access control techniques to capture the privacy-related elements of regulations. May et al. propose an "auditable privacy system" that includes conceptualizations for transfer, actions, creation, rights establishment, notification, and logging [35]. Leveraging the similarity between legal privacy texts and APIs in specifying rules on accessing protected information, they derive privacy-focused mandatory access control rules directly from regulations [35]. This translation into access control rules captures regulatory conditions and obligations as allow/deny operations. Conditions and obligations that cannot be represented as access control rules are cast instead as external environmental flags [35].

The auditable privacy system implementation fulfills some key requirements engineering tasks, but its narrow focus keeps it from adequately supporting the complex needs of requirements engineers working with legal texts. May et al. use a modeling language to represent legal texts and privacy policies, thus enabling model checking and verification operations. Such formalism supports queries on the regulatory model, so that developers and policy-makers alike can analyze a given text and evaluate compliance and design issues [35]. However, their regulatory model abstracts away many key aspects and characteristics of legal texts; for example, May et al. assume that external and ambiguous references are satisfied by default. In addition, the model omits many low-level system requirements (e.g., password procedures) specified by HIPAA [35]. The narrow privacy focus, coupled with the inadequate support for key elements of legal texts, makes this approach unsatisfactory for requirements engineers who need to extract requirements from legal texts and monitor compliance.

4.7   Markup-Based Representations

Given the hierarchical nature of legal texts, some researchers have attempted to capture legal texts with semi-structured markup languages, such as Standard Generalized Markup Language (SGML) and Extensible Markup Language

(XML). Such markup-based representations can mimic the structure of legal texts and also maintain annotations and other metadata regarding each section, part, or even sentence of the original legal text [25]. A markup-based representation also enables the system to easily capture and display information on definitions, acronyms, and cross-references within legal texts, thereby addressing several of the key requirements for using legal texts during system development. A semi-structured representation can be combined with well-established information retrieval techniques and first-order predicate logic to aid users in both locating and analyzing relevant sections of a legal text [30]. In addition, some newer legal texts already are being represented in XML; augmenting these existing representations is a relatively easy task [37]. Research efforts in this area include: SGML modeling of decisions of the Supreme Court of Canada [42]; REGNET, an XML framework for representing regulations [25][26][29][30][31]; and an overview of several XML models for representing legal texts [37].

Markup-based representations hold promise for providing requirements engineers with the necessary framework for leveraging regulations in system development. The work in SGML was an isolated effort now superseded by research utilizing XML, a simplified derivative of SGML that is easier to process. The REGNET project, based on an XML framework, has generated over 25 published papers describing the system and its use in tasks such as: representing regulations [25], providing similarity analysis between different regulations [29], and helping policy-makers in drafting new regulations [31]. The REGNET project includes a parser to automatically transform regulations into XML and uses other tools to semi-automatically generate conceptual tags for the markup [25]. REGNET provides a foundation for verifying compliance with a specific regulation, but has been tested only in limited domains and the prototype system is not yet available to other researchers. In addition, in its current form REGNET does not provide a precise model of the regulations [25].

Finally, the research evaluating several different markup-based approaches does not provide details on the underlying representations; instead it focuses on techniques for ranking the different XML models being reviewed [37]. Thus, while markup-based approaches benefit from mimicking the hierarchical, semi-structured nature of regulations, previous research approaches do not offer developers any available tools to shape requirements engineering and design efforts around regulatory compliance. The REGNET prototype system shows the most promise in assisting with compliance efforts, but comparing and drafting regulations, rather than extracting system requirements, has become the main focus of this work.

4.8   Goal Modeling

The SecureTropos approach, based on the i* framework, involves extracting and representing the goals, soft goals, tasks, resources, and social relationships for defining obligations [33]. It then uses these concepts to model the relationships for actors, dependencies, trust, delegation, and goal refinement [33]. SecureTropos has been used to assess a university's compliance with the Italian Data

Protection Act [33]. Whereas the focus of the research is on applying requirements engineering principles to security requirements, the broader context examines how an organization can assess its compliance with standards from a particular legal text.

The SecureTropos approach requires a manual extraction of the concepts. As with several previously-discussed approaches, traceability is not addressed, and we have yet to find any examples of the mapping between the extracted concepts and their presence in the original regulation. SecureTropos may enable developers to better design systems to be compliant with the fundamental concepts of a specific security regulation, but its scalability and applicability to a broader range of legal texts is as yet unproven. Finally, SecureTropos does not currently provide users with the ability to answer specific legal queries or identify changes in the law over time.

4.9   Reusable Requirements Catalog

Toval et al. created a reusable catalog of legal requirements that were derived from specific legal texts regarding security and personal data protection [48]. The Personal Data Protection (PDP) Catalog enables requirements engineers to incorporate legal requirements into the development lifecycle and build compliance into new systems [48]. By providing reusable legal requirements, analysts can more easily uncover ambiguities and inconsistencies, and the quality of the catalog improves with each usage [48].

This initial foray into applying requirements engineering methodologies to legal requirements provides some interesting insight, but does not satisfy the comprehensive set of requirements engineering needs that we address in this chapter. For example, Toval et al. highlight traceability as particularly important in requirements engineering, yet they provide no evidence of maintaining traceability between the derived requirements and the source in the legal text, much less the traceability required for all the cross-references to other texts. Although their process appears to be a manual effort, Toval et al. fail to mention the length of the regulations they processed or how much time they spent extracting requirements. Thus, it is difficult to properly evaluate the efficiency and efficacy of their approach. In addition, a legal requirements catalog requires updates each time the law changes. Finally, the PDP Catalog does not address the problem of overlapping or conflicting legal texts; the ability to manage and resolve these conflicts is an essential part of the requirements engineering process for systems governed by laws and regulations.

4.10   Identifying Requirements through Case Law

Breaux et al. explored the possibility of identifying requirements through legal violations discussed in administrative and case law [13]. The research focused on discovering critical requirements, or requirements derived from vulnerabilities in existing software systems as documented within court records [13]. By eliciting requirements that may have been missed during initial system development

and design, requirements engineers can improve compliance and better react to changing interpretations of laws and regulations. The exploratory case study discusses three key tasks facing requirements engineers in handling court records: (1) how to identify relevant case law; (2) how to select relevant court documents for analysis; and (3) how to best represent extracted information, through sequence, misuse case and KAOS diagrams [13]. This research also describes the criminal law process in the United States, which provides useful guidance for requirements engineers attempting to work with criminal case law.

This work is still in its infancy, however, as it was applied to only eight carefully-selected court cases, seven of which involve criminal acts with underlying HIPAA violations. The scalability of such a technique has not been established, nor does this approach provide sufficient guidance for identifying case law relevant to a particular software system. Furthermore, the approach was tested only against criminal case law and did not address administrative rulings or other types of case law. Finally, the court cases selected in this case study did not involve direct violations of the laws of interest to requirements engineers, but rather criminal conduct that implicated those laws (e.g., identity theft with an underlying HIPAA violation, rather than an individual being prosecuted for a HIPAA violation directly). However, the case study shows promise for how requirements engineers might incorporate administrative and case law into the requirements elicitation and analysis phases of software system development.

5   Supporting Requirements Engineering in Legal Contexts

Given our experiences to date [2][14][20] and our thorough survey of efforts to support the analysis of legal texts discussed herein, we identify several key elements for any system to support the analysis of legal texts for requirements specification, system design, and compliance monitoring.

5.1   Identification of Relevant Laws and Regulations

Our discussion in Section 1 focused on the need to identify relevant legal texts, extract the requirements for a given system, and answer specific legal queries to test for compliance. Identifying relevant legal texts may not appear to be a problem facing requirements engineers, but our experience to date shows that it is a key consideration during requirements elicitation. Oftentimes, analysts discover additional relevant laws or regulations only when they are midway through a careful analysis of a particular legal text. Much as the reader of this chapter may see a citation and check the list of references to locate and read that source, requirements engineers similarly identify external laws or regulations that constrain the very legal text they are examining at any given point in time. This is not a trivial activity. The referenced legal text may have a completely different set of definitions and terminology, requiring further interpretation and careful analysis. Making use of the supplemental documents to identify similar and related legal texts will also aid in addressing the identification problem.

5.2   Classification of Legal Texts with Metadata

Some classification of legal texts is necessary for developers and auditors to sort through the large corpus of legal texts and identify those with relevance to the project or system at hand. To this end, the idea of tagging legal texts with metadata, as proposed by [25] and others, can lead to a categorization of such texts over time. For example, a regulatory section such as § 164.310 in the HIPAA regulations can be annotated as generally describing security, or specifically detailing physical safeguards; in another categorization, it could be tagged as containing low-level system requirements. With each new legal text tagged, the corpus becomes more accessible and easily navigated. This use of metadata can directly combat the complexity of legal texts and facilitate incorporating legal requirements into the development process.

5.3   Prioritization of Legal Texts and Exceptions

A system for handling legal texts should address the nature of such texts in its underlying approach. One key requirement is to handle the hierarchical nature of laws and regulations. Oftentimes exceptions take precedence over the normative legal requirement. To properly assist requirements engineering efforts within this context, a support system should understand and manage the relationships between overlapping or contradictory legal texts. This will enable analysts and auditors to make determinations about which legal texts override others, depending on jurisdiction. This becomes particularly important when considering the effects of globalization. For example, various nations' laws and regulations on data privacy may differ or contradict one another; thus, users need mechanisms for resolving those situations. In addition, it is important to accommodate supplemental information as well as case or administrative law. This information can again be captured as metadata; sections further explained or disambiguated by supplemental texts or interpretative rulings can be annotated with the more detailed information.

5.4   Management of Evolving Laws and Regulations

It is critical for requirements engineers and compliance auditors to be able to manage the evolution of legal texts over time. Given the frequent revisions to legal texts as previously discussed, requirements engineers need to be able to capture these changes and maintain an up-to-date view of the relevant texts requiring analysis at any given time. It may be necessary to compare changes, and understand the impact of their scope, at distinct time periods to understand how requirements have evolved and how compliance efforts are impacted by modifications to legal texts. Thus the system must not maintain traceability not only between legal texts and requirements, but must also track the point in time at which that link was established. For legal analysis and the future development of case or administrative law, such metadata may be critical for verifying compliance. Analysts may be forced to update requirements or concepts as laws and regulations change, and therefore

they will require methods for tracking the status of development efforts vis-à-vis the changes in legal texts over time.

5.5   Traceability between References and Requirements

As previously discussed, traceability support for both external and internal references is critical to ensure requirements engineers are able to accurately capture the full meaning of any given legal text. In Section 2 we discussed the prevalence of cross-references within laws and regulations; external references also occur frequently in legal texts. Thus, it is imperative to maintain traceability between any section with a reference and the legal text being referenced. This is especially important given that external references often establish legally binding priorities among requirements and allowable information accesses, uses, disclosures, and removals. Navigating across these references, as well as from specific legal statements to the derived requirements, will improve analysts' understanding of the text and is essential for gathering all requirements and concepts expressed by a particular text.

5.6   Data Dictionary and Glossary to Ensure Consistency

The use of consistent definitions and terminology is important in the design of any software system, and of paramount importance in the context of legal compliance. A data dictionary for all domain-specific definitions and acronyms is needed to support requirements engineers, policy-makers, and auditors in establishing a unified glossary for the system specification, design documents, and compliance audit artifacts. In dealing with legal texts, requirements engineers will frequently encounter unfamiliar and complex terms, making a thorough glossary even more important [17]. Given that multiple legal texts may share similar words with different interpretations, users must be able to view any word's definition given the context of a specific text. These definitions should then be referenced in the creation of a system-wide glossary; once again traceability between the original legal terms and the system glossary must be maintained.

5.7   Semi-automated Navigation and Searching

Analysts need to be able to access legal texts in a machine- and human-readable state. Previous requirements engineering research emphasized the relevance of such access in highly-regulated domains such as health care [17]. Some tasks, such as extracting concepts and adding metadata, need to be supported by semi-automated processes; use of semi-automated annotation tools is an active research topic (e.g., Semio Tagger [25] and CERNO [28]). In addition, users must be able to view the original legal text at any time, and traceability needs to be maintained between any machine-readable or logic-based format and the original natural language representation. Analysts must be able to easily search and navigate legal texts with varying levels of granularity. Given the complexity

of the law, users may need to search for specific terms, for more general concepts, or even scan entire sections of legal texts to clarify their understanding or support requirements engineering efforts.

5.8   Annotation of Legal Statements

As discussed in Section 2, legal texts are laden with ambiguities. Some ambiguities in the law may be intentional, but analysts still need to establish an interpretation of the law in these cases, as well as maintain traceability with the section being interpreted. Analysts must be able to attach auxiliary annotations to ambiguous sections to flag them for further analysis in collaboration with the proper stakeholders, such as an organization's legal counsel. Ideally, analysts should be able to track interpretations across legal texts such that users will be able to view all assumptions upfront and differentiate the interpretations according to the context and conditions associated with any given situation. The ability to link legal texts and requirements with supplemental documentation will aid analysts by providing them with additional support for disambiguating texts for requirements extraction.

5.9   Queries Comparing Legal Concepts and Compliance

As supported by a wide range of approaches [1][7][10][35][36][44][45][46], it should be possible to perform directed queries on the model of the legal text. These queries enable analysts to support disambiguation and auditing efforts. Specific legal queries can allow analysts and auditors to identify all applicable laws and regulations, discover all uses of a particular term or concept, and compare different texts. Auditors may also wish to query the system to determine whether a particular legal text has been addressed in a system's design, or whether any requirements correspond to a given section.

6   Discussion and Future Work

This chapter has largely focused on work within the computer science and artificial intelligence domains. It is likely that there has been work with legal texts in other engineering domains that can be applied to the tasks facing requirements engineers in devising software systems for using laws and regulations. It would be useful to examine how system developers are currently handling legal texts. Empirical studies of specific organizations would likely reveal additional requirements in dealing with laws and regulations. One such study could focus on a particular domain and examine how requirements engineers and system developers identify and handle relevant legal texts. Another study could focus on a particular legal text to pinpoint what elements of the text are used and how the text is managed in terms of the project.

Other concepts studied in requirements engineering are likely to be relevant for systems managing laws and regulations. Future work should consider how

requirements engineering research on viewpoints and frameworks can be applied to legal compliance systems. Research into natural language processing may also provide insight into parsing legal texts.

Research is underway examining how to mine legal texts to create hierarchies of stakeholders, data objects, and events. There have also been preliminary efforts to conduct an empirical study of a requirements specification to check for legal compliance [34]. The case study begins with the previously-derived requirements and works backward to establish traceability with the legal text [34]. It is likely such research will uncover additional issues in monitoring compliance by working backwards from requirements specifications to the legal text and thus will discover additional requirements for any future legal compliance system.

7   Conclusion

This chapter discusses the role of legal texts in requirements engineering and attempts to bring attention to this important domain within the requirements engineering community. The characteristics of laws and regulations make them both necessary and challenging to use during system development. This survey examines the past 50 years of work in modeling laws and regulations, extracting key concepts from legal texts, and monitoring compliance. In addition, we discuss what is required to effectively support analysts that must deal with legal texts in specifying system requirements and determining legal compliance.

References

1. Allen, L.E.: Symbolic Logic: A Razor-Edged Tool for Drafting and Interpreting Legal Documents. Yale Law Journal 66(6), 833–879 (1957)
2. Antón, A.I., Earp, J.B., Potts, C., Alspaugh, T.A.: The Role of Policy and Stakeholder Privacy Values in Requirements Engineering. In: 5th IEEE International Symposium on Requirements Engineering, pp. 138–145 (2001)
3. Antoniou, G., Billington, D., Governatori, G., Maher, M.J.: On the Modelling and Analysis of Regulations. In: 10th Australasian Conference on Information Systems, pp. 20–29 (1999)
4. Antoniou, G., Billington, D., Maher, M.J.: On the Analysis of Regulations using Defeasible Rules. In: 32nd Hawaii International Conference on System Sciences, pp. 1–7 (1999)
5. Barth, A., Datta, A., Mitchell, J.C., Nissenbaum, H.: Privacy and Contextual Integrity: Framework and Applications. In: 2006 IEEE Symposium on Security and Privacy (2006)
6. Bench-Capon, T.J.M.: Support for Policy Makers: Formulating Legislation with the Aid of Logical Models. In: 1st International Conference on Artificial Intelligence and Law, pp. 181–189 (1987)
7. Bench-Capon, T.J.M., Robinson, G.O., Routen, T.W., Sergot, M.J.: Logic Programming for Large Scale Applications in Law: A Formalisation of Supplementary Benefit Legislation. In: 1st International Conference on Artificial Intelligence and Law, pp. 190–198 (1987)

8. Berry, D.M., Kamsties, E., Krieger, M.M.: From Contract Drafting to Software Specification: Linguistic Sources of Ambiguity. Technical Report, University of Waterloo (2003)
9. Biagioli, C., Francesconi, E., Passerini, A., Montemagni, S., Soria, C.: Automatic Semantics Extraction in Law Documents. In: 10th International Conference on Artificial Intelligence and Law, pp. 133–140 (2005)
10. Biagioli, C., Mariani, P., Tiscornia, D.: ESPLEX: A Rule and Conceptual Based Model for Representing Statutes. In: 1st International Conference on Artificial Intelligence and Law, pp. 240–251 (1987)
11. Boella, G., van der Torre, L.: Permissions and Obligations in Hierarchical Normative Systems. In: 9th International Conference on Artificial Intelligence and Law, pp. 109–118 (2003)
12. Breaux, T.D., Antón, A.I.: An Algorithm to Generate Compliance Monitors from Regulations. Technical Report, North Carolina State University (2006)
13. Breaux, T.D., Lewis, J.D., Otto, P.N., Antón, A.I.: Identifying Vulnerabilities and Critical Requirements Using Criminal Court Proceedings. Technical Report, North Carolina State University (2008)
14. Beaux, T.D., Vail, M.W., Antón, A.I.: Towards Regulatory Compliance: Extracting Rights and Obligations to Align Requirements with Regulations. In: 13th IEEE International Conference on Requirements Engineering, pp. 46–55 (2006)
15. Casassa Mont, M., Thyne, R., Bramhall, P.: Privacy Enforcement with HP Select Access for Regulatory Compliance. Technical Report, HP Labs (2005)
16. Chantree, F., Nuseibeh, B., de Roeck, A., Willis, A.: Identifying Nocuous Ambiguities in Natural Language Requirements. In: 13th IEEE International Conference on Requirements Engineering, pp. 56–65 (2006)
17. Cysneiros, L.M.: Requirements Engineering in the Health Care Domain. In: IEEE Joint International Conference on Requirements Engineering, pp. 350–356 (2002)
18. Ernst & Young: 10th Annual Global Information Security Survey (2007)
19. Governatori, G., Rotolo, A., Sartor, G.: Temporalised Normative Positions in Defeasible Logic. In: 10th International Conference on AI and Law, pp. 25–34 (2005)
20. He, Q., Otto, P.N., Antón, A.I., Jones, L.: Ensuring Compliance Between Policies, Requirements and Software Design: A Case Study. In: 4th IEEE International Workshop on Information Assurance (2006)
21. Hohfeld, W.N.: Some Fundamental Legal Conceptions as Applied in Judicial Reasoning. Yale Law Journal 23(1), 16–59 (1913)
22. Johnston, B., Governatori, G.: Induction of Defeasible Logic Theories in the Legal Domain. In: 9th International Conference on AI and Law, pp. 204–213 (2003)
23. Kamsties, E.: Surfacing Ambiguity in Natural Language Requirements. Ph.D. Dissertation, University of Kaiserslautern (2001)
24. Kamsties, E., Berry, D.M., Paech, B.: Detecting Ambiguities in Requirements Documents Using Inspections. In: 1st Workshop on Inspection in Software Engineering, pp. 68–80 (2001)
25. Kerrigan, S., Law, K.H.: Logic-Based Regulation Compliance-Assistance. In: 9th International Conference on Artificial Intelligence and Law, pp. 126–135 (2003)
26. Kerrigan, S., et al.: Information Infrastructure for Regulation Management and Compliance Checking. In: National Conference on Digital Government Research, pp. 167–170 (2001)
27. Kitchenham, B.: Procedures for Performing Systematic Reviews. Technical Report, Keele Universitiy (2004)
28. Kiyavitskaya, N., Zeni, N., Mich, L., Cordy, J.R., Mylopoulos, J.: Annotating Accommodation Advertisements using CERNO. In: 14th ENTER Conference (2007)

29. Lau, G.T., Law, K.H., Wiederhold, G.: Similarity Analysis on Government Regulations. In: 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 111–117 (2003)
30. Lau, G.T., Kerrigan, S., Law, K.H., Wiederhold, G.: An E-Government Information Architecture for Regulation Analysis and Compliance Assistance. In: 6th International Conference on Electronic Commerce, pp. 461–470 (2004)
31. Lau, G.T., Law, K.H., Wiederhold, G.: Legal Information Retrieval and Application to E-Rulemaking. In: 10th International Conference on Artificial Intelligence and Law, pp. 146–154 (2005)
32. Martinek, J., Cybulka, J.: Dynamics of Legal Provisions and its Representation. In: 10th International Conference on Artificial Intelligence and Law, pp. 20–24 (2005)
33. Massacci, F., Prest, M., Zannone, N.: Using a Security Requirements Engineering Methodology in Practice: The Compliance with the Italian Data Protection Legislation. Technical Report, University of Trento (2004)
34. Massey, A.K., Otto, P.N., Antón, A.I.: Analyzing HIPAA Compliance: A Case Study in Aligning Requirements with Regulations in the iTrust System. Technical Report, North Carolina State University (2008)
35. May, M.J., Gunter, C.A., Lee, I.: Privacy APIs: Access Control Techniques to Analyze and Verify Legal Privacy Policies. In: 19th Computer Security Foundations Workshop (2006)
36. McCarty, L.T.: The TAXMAN Project: Towards a Cognitive Theory of Legal Argument. In: Niblett, B. (ed.) Computer Science and Law, pp. 23–43. Cambridge Press, New York (1980)
37. Moens, M.-F.: Combining Structured and Unstructured Information in a Retrieval Model for Accessing Legislation. In: 10th International Conference on Artificial Intelligence and Law, pp. 141–145 (2005)
38. Nissenbaum, H.: Privacy as Contextual Integrity. Washington Law Review 79(1), 119–157 (2004)
39. Otto, P.N., Antón, A.I., Baumer, D.L.: The ChoicePoint Dilemma: How Data Brokers Should Handle the Privacy of Personal Information. IEEE Security & Privacy 5(5), 15–23 (2007)
40. Palmirani, M., Brighi, R., Massini, M.: Automated Extraction of Normative References in Legal Texts. In: 9th International Conference on Artificial Intelligence and Law, pp. 105–106 (2003)
41. Peek, N.: Representing Law in Partial Information Structures. Artificial Intelligence and Law 5(4), 263–290 (1997)
42. Poulin, D., Huard, G., Lavoie, A.: The Other Formalization of Law: SGML Modelling and Tagging. In: 6th International Conference on Artificial Intelligence and Law, pp. 82–88 (1997)
43. Robinson, W.N.: Implementing Rule-Based Monitors within a Framework for Continuous Requirements Monitoring. In: 38th Hawaii International Conference on System Sciences (2005)
44. Sergot, M.J., Kamble, A.S., Bajaj, K.K.: Indian Central Civil Service Pension Rules: A Case Study in Logic Programming Applied to Regulations. In: 3rd International Conference on Artificial Intelligence and Law, pp. 118–127 (1991)
45. Sergot, M.J., Sadri, F., Kowalski, R.A., Kriwaczek, F., Hammond, P., Cory, H.T.: The British Nationality Act as a Logic Program. Communications of the ACM 29(5), 370–386 (1986)
46. Sherman, D.M.: A Prolog Model of the Income Tax Act of Canada. In: 1st International Conference on Artificial Intelligence and Law, pp. 127–136 (1987)

47. Stamper, R.: LEGOL: Modelling Legal Rules by Computer. In: Niblett, B. (ed.) Computer Science and Law, pp. 45–71. Cambridge Press, New York (1980)
48. Toval, A., Olmos, A., Piattini, M.: Legal Requirements Reuse: A Critical Success Factor for Requirements Quality and Personal Data Protection. In: IEEE Joint International Conference on Requirements Engineering, pp. 95–103 (2002)
49. Valente, A., Breuker, J.: ON-LINE: An Architecture for Modelling Legal Information. In: 5th International Conference on Artificial Intelligence and Law, pp. 307–315 (1995)

Requirements' Role in Mobilizing and Enabling Design Conversation

Mark Bergman

Naval Postgraduate School
Monterey, CA 93943

Abstract. Requirements play a critical role in a design conversation of systems and products. Product and system design exists at the crossroads of problems, solutions and requirements. Requirements contextualize problems and solutions, pointing the way to feasible outcomes. These are captured with models and detailed specifications. Still, stakeholders need to be able to understand one-another using shared design representations in order to mobilize bias and transform knowledge towards legitimized, desired results. Many modern modeling languages, including UML, as well as detailed, logic-based specifications are beyond the comprehension of key stakeholders. Hence, they inhibit, rather than promote design conversation. Improved design boundary objects (DBO), especially design requirements boundary objects (DRBO), need to be created and refined to improve the communications between principals. Four key features of design boundary objects that improve and promote design conversation are discussed in detail. A systems analysis and design case study is presented which demonstrates these features in action. It describes how a small team of analysts worked with key stakeholders to mobilize and guide a complex system design discussion towards an unexpected, yet desired outcome within a short time frame.

Keywords: Requirements Analysis, Boundary Objects, Design Theory, Design Conversation, Problems-Requirements-Solutions Triangulation.

1 The Role of Requirements in Design

Requirements are a bridge between what customer stakeholders want and what suppliers can design and build. More simply, it is they represent the link between systems analysis and design – i.e. problems and solutions. Requirements quantify and frame problems, while reducing the number of possible feasible solutions. They also define who are and are not legitimate stakeholders within a design process.

To understand the role of requirements in systems design, a few definitions need to be set. A *problem* is a gap between what the current situation and a desired improvement in the situation – i.e. the gap between an existing and a desired organizational state. In general, stakeholders are people who can demonstrably change the course of design process as well as determine and measure success criteria – i.e. goals. Principal stakeholders have resource powers can define a problem and mobilize resources to affect a solution [1].

Table 1. Requirements as Bridges between Problems and Solutions

		Problem → Solution	Solution→Problem
Customers	*Technical*	• Determination of specific desired capabilities and expected operational constraints that together address their problem • Determination of success/failure criteria • Determination of level of acceptable risk and ROI • Determination of existing available resources that can be dedicated to problem analysis	• Declaration of existing legacy systems; business rule, policies, procedures; costs, resources, timelines; and expertise that will not change regardless of the solution – i.e. high level requirements • Determination of physical operations environment • Definition of the expected user, support, and management communities • Qualification of suppliers as possible technology providers • Determination of whether or not a supplier can provide an acceptable solution within given capability, cost and time constraints
	Political	• Determination of whose problems receive organizational attention • Definition of involved stakeholders and their levels of organizational power – i.e. authority • Framework for creating a stable, enforceable agreement – contract • Determination of which suppliers need to be involved in creating a contract	• Definition of parts or the whole authority structure(s) that will govern the new system solution • Dedication of champions of the new technology • Determination of a program of organizational development and change • Support of existing contracts and service agreements • Support of ongoing relationships with suppliers
Suppliers	*Technical*	• Determination of which problems the supplier can adequately – technically and economically – address • Availability of expertise as well as their assistance in initial problem determination • Determination of the customer's ability to pay for products and services rendered • Support of existing use of products and services by the customers – existing relationships; part of legacy infrastructure	• Specific existing capabilities and constraints via their product lines as well as their costs and availabilities • Possible, feasible extensions to existing capabilities for more time and cost • Available and delivered profession service expertise in analysis, design, development, and deployment
	Political	• Determination of who will work with the customers • Determination of legal liability as will be specified in a contract • Determination of which customers need to be involved in creating a contract	• Existing and ongoing contracts and agreements • Necessary new contracts and agreements • Support of ongoing relationships with customers

Altogether, legitimate stakeholders – *stakeholders* for short – can directly determine the success or failure of the design product. All others are not stakeholders, regardless of their roles or positions of authority.

A *solution* solves a given problem to the stakeholders' satisfaction, i.e. it meets or exceeds their success criteria. The success or failure of a solution is determined primarily by principal stakeholders; then by the other key stakeholders; followed by the rest of stakeholders in general. For example, say there exists a business owner wants an ERP system to integrate his or her sales, marketing, finance, and operations divisions. This owner must define and measure what the success criteria of the final information system. These criteria are grounded in the measuring the improvement created by the desired new outcome – where the problem is solved and the solution is successfully deployed – versus the existing situation. Any acceptable ERP solution must meet or exceed these criteria. In addition, other stakeholders - key users, IS support groups, other division managers and the like – can accept or reject a solution. Beyond this, the general user community can accept or reject the deployed solution. Therefore, the success criteria of all those that can directly determine a solution's success or failure must be accounted for during design.

Requirements must represent, at minimum, these design criteria. They connect and contextualize problems and solutions [2]. They do so in both directions: 1) problem to solution and 2) solution to problem. Problems are owned, defined, and mobilized by customer stakeholders – lead by one or a group of key principal stakeholders. Solutions are designed, developed, built, deployed, and mobilized by supplier stakeholders – again, lead by one or a small group of key principals. Based on these insights, Table 1 presents an initial (non-exhaustive) list of meta-requirements – categories of requirements – based on an analysis of bridging problems and solutions.

From examining these definitions, the role of requirements in design becomes clear. They must address the technical and political realities of connecting problems and solutions – as per Table 1. Requirements represent mandatory technical specifications, frameworks, and legacy infrastructure as well as the political alignments and agreements between stakeholders. A common form of these agreements is legally binding contracts. In the past, requirements have been defined as mandatory technical specifications [3-5]. Yet, from the examination of bridging customer stakeholders' problems and supplier stakeholders' solutions, this technical role is necessary, but not sufficient. Instead, the overall role of requirements in design is to provide a stable technical and political framework that enables and encapsulates the bridging of customers' problems and suppliers' solutions.

2   Utilizing Problems-Requirements-Solutions (PRS) to Enact Design

For any one problem, there are a number of possible, feasible solutions. Conversely, any one solution can address multiple problems. Therefore, there is a many-to-many relationship between problems and solutions. Requirements act like an "associative

entity" between problems and solutions[1]. They provide a conduit from solutions to problems. Requirements inform stakeholders of the choices of possible solutions as framed by the selected problems. Likewise, they reduce the number of possible problems by associating them only with feasible combinations of workable or buildable technologies along with achievable, sustainable political agreements.

Therefore, a combination of requirements filtered problems & solutions is a candidate model for design. Interestingly, there are fewer restrictions on problems and solutions than there are on requirements. They have some flexibility to change to better fit the needs of the customers and abilities of the suppliers. Yet, requirements are strict in their definition and application. Arguably, this makes the problems and solutions highly dependent upon requirements. Still, requirements must be rediscovered, created, or modified due to changes in problems or solutions during discovery and conflict resolution in the system or product design lifecycle.

Together, problems-requirements-solutions (PRS) triangulation is a model for discovering feasible and satisfying designs. Adjustments to any one part of the PRS model require corresponding adjustments in the other two parts. Yet, the stickiness of requirements reduces the degrees of freedom of corresponding problems and solutions. As each part of this design model is reduced, solidified, and becomes more specific, the same occurs to the other two parts. For example, as problems become more specifically defined, increasingly focused requirements and solutions can be determined or created.

Examining Table 1, PRS triangulation can be based on a set of measures or metrics that indicate the strength and quality the technical and political balance for any particular design. Utilizing metrics feedback, further adjustments to any part of the PRS models can be made to discover improved design balances[2]. Hence, each enactment of design leads to one of three possible PRS states:

1) Infeasible
2) Feasible, but sub-satisficing
3) Satisficing

Infeasible are combinations of problems and solutions that cannot rectify technical or political requirements. All other PRS combinations are feasible. Of these possibilities, theoretically, only one combination has the optimum balance of technical and political requirements. Still, as Simon pointed out, this combination is likely too complex to determine [8]. The best one can do is satisfice in this situation. Of the feasible designs, a few are satisficing, i.e. those with roughly the same trade-offs, yet provide similar value. In other words, these are the top PRS combinations – i.e. design candidates. They are based on achieving the best balance of feasible technologies, political

---

[1] An associative entity is created to normalize many-to-many relationships between two database entities [6, 7]. It enables 1-to-many relationships by focusing on the unique instance that the combination of entities creates. Examples of this are invoices and sales orders.

[2] A danger here is "analysis paralysis." It is possible to keep trying out different combinations while never setting on any one choice. This can be made worse if there is any creeping requirements or featurism. That can explode the possible number of design combinations. Therefore limits must be place on how long and how far the search can commence in order to identify satisficing designs.

arrangements, and satisfaction of all stakeholder requirements within a given amount of time and resources.

Since there are many possible PRS combinations for each design, there is no single set of "correct" requirements. Instead, there exist separate, yet likely overlapping sets of requirements that correspond to competing design variants. As previously mentioned, these requirements can be adjusted in response to the nuances that are discovered or created while refining problems and solutions. This argument breaks with the overly positivistic view of discovering or creating a single true set of requirements that tends to dominate their definition and use in the design process[3]. Instead, requirements can be utilized along with problems and solutions to find competing, satisficing designs. Simply, requirements are part of a design conversation, not a single fixed attribute of design.

3   Requirements as a Hindrance to Design Conversation

As per the IEEE standard 610.12, requirements must be written down in a document [3, 13]. They are usually described in a logic-based form or a design model – commonly, an UML object-oriented (OO) model [14, 15]. Logical format written text or object-oriented complex models is generally not understandable by most stakeholders. Put simply, the vast majority of stakeholders are not logicians. A detailed requirements specification may provide a means of discussion and mobilization to action between analysts, designers, and engineers. Yet, it does not promote an understandable, shared representation between those these technical experts and customers, managers, users, and other stakeholders who do not use logic as a language of work[4].

Moreover, a logical text or modeling language is resistant to adaptations, even between analysts, designers, and engineering. Requirements specifications are not meant to change. Yet, the demands of designs and business needs force changes in requirements as new knowledge about a design's purpose, functionality, and constraints becomes known during early system or product lifecycle processes. A rigid, difficult to understand language inhibits transformation of design knowledge across stakeholder groups.

How can requirements still meet the IEEE definition and enable communication between stakeholders during design? One answer can be found in treating requirements documents as design boundary objects (DBOs). As per Bergman, Lyytinen, and Mark (2007), a design boundary object must meet four essential features to be useful by its target stakeholders.

---

[3] Positivism is the philosophical belief that there exist one objective truth and the goal of science is to uncover it [9]. Arguably, there is a widely espoused belief a in the existence of the one true set of objective requirements that are the results of appropriately performed systems analysis, [10-12]. The requirements absolutely frame and ground any possible design. This essay challenges this belief and replaces it with one that is more flexible based on the nuances of differing combinations of problems and solutions.

[4] Of course, the exception to this situation interactions between technical experts and highly technically literate customers and other business stakeholders.

Table 2. Design Boundary Object Features[5]

Feature	Definition	Domain	Description
Promote Shared Representation	Encapsulates understandings based on a common syntax and semantics, which are shared across social worlds	*Technical*	• Form shared functional representations, i.e. data and technical models, prototypes, architectures, specifications, etc. [17, 18] • Transform knowledge at the boundary between social worlds [19, 20]
		Political	• Form shared political representations, i.e. agreements, contracts, "sign-offs," memorandums of understanding, etc. • Perspective sharing, i.e. making sense of other social world's perspective. [20]
Transform Design Knowledge	Manipulate and converse representations that will propel movement between design routines as to facilitate finding a feasible functional solution and stabilize the political ecology	*Technical*	• Move knowledge from ambiguous to specific; objective/goal to a problem; instable to stable; idea to solution [13, 21] • Realign operational structure to stabilize functional ecology • Enable design traceability [22]
		Political	• Hand-off of power and control from provider to recipient world(s) [19] • Realign power to stabilize political ecology [23, 24] • Enable agreements traceability
Mobilize for Design Action	Source and wield resources and power to propel progress along a design path.	*Technical*	• Participate in SAD routines as to invite functional expertise, review solutions, etc. [21] • Reduce problem ambiguity for solution discovery in a design path • Conscribe expertise relevant for problem identification and solution [21]
		Political	• Participate in decision making, mobilization of resources and allocation of design tasks [25] • Mobilize bias towards preferred resolution [25] or quick disintegration • Enable each group of stakeholders at similar organization levels ability to apply their authority power while respecting the other levels [26]
Legitimization of Design Knowledge	Grant a legitimate status to a boundary object through validation of its content as to align with the stakeholders' intent.	*Technical*	• Certify, verify and validate the truthfulness and correctness of design knowledge [21]
		Political	• Demonstrate acceptability of goal(s), problem(s) and solution(s) in the given institutional order as to authorize the movement of design knowledge across social worlds [1]

---

[5] This table is taken from [16].

These four essential features are defined in Table 2. The most form and format of requirements – logical text based specifications or object-oriented (class or activity) models – tends to fail these features.

As mentioned earlier, logic based text and complex OO models are difficult to understand and resistant to change. These can be for of shared representation between technically expert stakeholders. Yet, the vast majority of stakeholders are not technical experts, and their needs need to be met as well. These formats inhibit understanding and hence are in fact the opposite of a shared representation – a localized, specific representation. The advanced technical representation can and often is very useful for the specific subset of technical stakeholders who understand and utilize the power of logic and models. But, there needs to be a shared representation of requirements for the rest of the stakeholders if they are to promote, not inhibit a design discussion.

If stakeholders cannot understand the representation, it is difficult to make informed design decisions. This directly impacts the ability to transform design knowledge and mobilize for design action. Indeed, as requirements become less understandable by those stakeholders whose needs they represent, it makes it less possible to rely upon during design activities and changes. Instead, it is likely that those stakeholders with decision power, but are not technically savvy will eventually make their own decisions about what design will become without the help and insights of detailed requirements. This may well lead to unfortunate outcomes, for instance building or purchasing systems that meet their needs, but not the needs of other system stakeholders [27, 28].

Interestingly, logic or OO model-based requirements do tend to legitimize design knowledge – as long as there is a legitimizing body in the product or system design group that has the ability to understand and verify these documents. The effort it takes to transform fuzzy needs and constraints into specific, concise, non-conflicting requirements is nontrivial and supplies a tremendous amount of insight to design [3, 15, 29, 30].

Still, those who do not understand or use these detailed forms of requirements cannot legitimize them themselves. They must trust that their needs and wants will eventually be satisfied by the resulting system. But, this breaks the fundamental feedback necessary to perform design. Therefore, it is difficult for non-technical – and even some technical – stakeholders to legitimize logic or OO model-based requirements documents. Altogether, highly technical requirements documents cannot legitimately represent nontechnical customers' and similar stakeholders' desires since these people cannot directly verify and validate them.

4   Where Do We Go from Here? – An Agenda for Improving Requirements Role in Promoting Design Conversation

To counter this weakness in requirements, future requirements documents need to be developed and deployed that, at minimum, conform to the four rules of an operational design boundary object. Therefore, an agenda for research over the next 10 years should focus on these questions:

- What common models, stories and other formats are necessary to enable requirements to serve as design boundary objects?
- How well do they represent problems-requirements-solutions design candidate combinations?
- How easy are they understood, manipulated, and updated, esp. by principal and key stakeholders?

- How easy can two or more design candidates be compared and contrasted to enable design selection?
- What are the methodologies to produce and test these new DRBOs?

It may well be likely that no one document or design requirements boundary object (DRBO) can satisfactorily answer these questions. Due to the adaptive, destructive, intuitive, and organic nature of design [31-33], it is like that a host of DRBOs are necessary to capture and represent requirements. Indeed, detailed logic or OO model-based requirements documents may still retain their place in detailed technical design work. Some, arguably few, problems require complete design and building of systems which are highly dangerous or financially risky. The higher the risk and larger the cost of failure, the more need for detailed, rigorous, heavyweight requirements analysis.

Yet, the majority of issues organizations face is not overly difficult. They cover the broad spectrum of annoyances and dysfunctions that occur as organizations grow, change and mature [34-36]. These can range from the need for better sales systems, point-of-sales services, project work product management, training courses and conference management, meeting and classroom scheduling, timesheet submissions, travel reporting, checking in and out of company property, inventory and parts management, and thousands of other similar day-to-day problems. Design for these problems is likely more lightweight. They do not need the detailed requirements analysis which is necessary to address more risky problems.

If the design space is broken up along level of risk, then design processes and corresponding problems, requirements, and solutions boundary objects can be redefined as per these different spaces. One could start out with low-risk, high-level problems-requirements-solutions and discover if there are more serious elements of risk uncovered during design activities. If so, then more detailed and technically rigorous analysis and design methods can be applied. Otherwise, a lightweight design methods and corresponding PRS triangulation processes can be deployed successfully for the majority of industry, academic, and governmental needs.

5   System Analysis and Design Cases Study – Design Requirements Boundary Objects in Action

Design can be seen as a detailed conversation amongst powerful stakeholders. The goal of this conversation is to identify problems that mobilize stakeholder action and discover feasible solutions that address these problems. The conversation arguably follows the following path. At first, it focuses on understanding and capturing the key pain points that need resolution or pleasure points that enable organizational growth. These issues are prioritized against other's issues. Early feasibility analysis can be applied. It is here that the initial, high-level requirements are established, i.e. those pertaining to

operations, finance and economics, schedule, technical realities, and political and legal issues. The design boundary objects need to capture this key information, while making it shareable and actionable. High-level solutions can be considered against the defined problems (pain or pleasure points) and corresponding requirements. The rest of the conversation is a matter to digging down to enough detail to find viable PRS combinations and the feasibilities metrics data to fairly compare them until either a solution is found and selected for development or the effort ends due to a lack of time, money, or further commitment from key stakeholders.

Here is a small case study to further illustrate the concept of design as a conversation. It demonstrates how DBOs, including DRBOs can enable and propel a nontrivial design conversation towards a successful conclusion. It begins at the Monterey Bay Aquarium (MBA) in Monterey, CA.

The MBA had key problems with their digital image storage and retrieval system. A team of graduate student researchers investigated and preformed a systems analysis and design. A scope analysis form was filled out during the initial interviews and observations. This form captures brief, yet detailed descriptions of: the mission and purpose of the organization, identifying the key stakeholders and their roles in the existing system, motivating problems, and describing the existing, legacy system. This form is the first design boundary object created and it initiated the design conversation. The scope analysis was examined and utilized to promote follow-up conversations. These follow-up conversations allowed key stakeholders to become further identified and define their salient problems. These problems were contextualized by MBA's mission and current operational constraints. Upon reflection and discussion, a deep structure, core problem was identified. It was summarized in a one page *Request for Information Services* document:

The Monterey Bay Aquarium needs improvement in managing the wealth of digital images catalogued at their facility. There is a strong interest to share their growing digital image library with the public to promote their overall mission of "Inspiring Conservation of the Oceans."

In addition, a brief, high-level description of an expected solution was provided:

Develop or acquire intuitive, high-capacity data storage and retrieval system. The system must incorporate a multi-user, web interface for digital image management.

This design boundary object provided the scope and boundaries of the systems analysis and design project. It created the initial problem-requirement-solution basis.

Yet, this PRS was too broad and needed refinement. The next steps the team took were to perform a more detailed problem analysis as well as an initial feasibility analysis. The problem analysis was a standard problem decomposition of the key problem into more manageable sub-problems [8]. These secondary issues clarify and contextualize the core problem. This information was captured in another DBO – the problem analysis form. These problems were also connected to constraining issues in dealing with the existing, mainly manual digital media management system. All of the problems were assigned priority and urgency rankings to further capture and represent the key stakeholders needs, positions, and intentions.

In addition, an initial feasibility analysis was performed. It focused on determining:

- *Operational Feasibility* – What are the changes to work place activities, processes, procedures, business rules, and organizational system and how well will these changes be received or opposed
- *Technical Feasibility* – What are the likely technologies that are part of a solution, how well do they meet the required new capabilities, can they operate under given constraints, what are the changes in expertise necessary to deploy the new system
- *Economic Feasibility* – What is the budget and financial metrics of success
- *Schedule Feasibility* – How long will it take to develop or purchase the selected solution and can this be accomplished in the allotted organization time window.

This DRBO determined the high-level capabilities and constraints, i.e. requirements, of any successful solution to the core problem. These requirements are stated in a manner that can be shared across stakeholder groups – senior management, engineering, finance, project management, and so forth. The feasibility analysis document was designed to enable each stakeholder group to refine the information within their area of responsibility. For instance, finance and marketing groups could supply economic details. The project manager supplied schedule details, and so forth. The results of each involved stakeholder group were able to be brought back together in this collaborative DRBO. As the systems analysis and design process continued, this document promoted shared representation of feasibility criteria; shared key information across stakeholder groups; mobilized action; and represented agreements that clearly defined the criteria for successful solutions. In turn, the feasibility analysis enabled and promoted the discovery of further requirements, problems, which helped scope down the space of possible solutions.

Armed with this information, the team then performed a use case analysis and captured the stakeholder social network. Use case analysis is described in a variety of texts [37-39], and hence, will not be discussed in detail. The purpose of applying use case analysis was to collect and derive the capabilities and constraints of the new system. The new system represents a conceptual model of a general system that, if it were already deployed and in place, would feasibly address the aforementioned problems. Each use case describes a specific operational capability of the new system. This is described with as little discussion of technology as possible in order to allow for and encourage different technical design approaches. In this case for the MBA, the use case analysis produced a simple system model and descriptive narratives. That sufficed to represent the current state of design to the key stakeholders. It also enabled initial information system architectural models.

In addition, a social network was developed, which represented the re-aligned relationships of the stakeholders in the new system. This indicated who the key individual stakeholders were, groups they represented, and their work activity based relationship with the others. It also captured possible political conflicts that could impair the acceptance of a new system. It was combined with the use cases to create a high-level operational capabilities model of a new system. This DBO enabled the beginning of organizational power conflict resolution even before a new system was highly

defined. In turn, political impasses were examined and rectified during use cases analysis and conceptual system architectural design.

The result of this work was the production of a technology-independent solution system. This was a common view of a system model was able to be reviewed, commented on, and augmented by the key stakeholders. It became the basis for the conceptual architecture. At this point, the team performed data analysis and process analysis of the new system. These filled out functional feasibility capabilities and constraints, again without specifying specific technologies. The results of this work refined the operational, economic, and schedule feasibility details.

After a round of feasibility refinement, the team discovered or developed possible solution candidates that fit the criteria. This assisted in grounding conceptual outcomes of the systems analysis and design with viable technologies. The purpose of this activity was to verify the existence of at least one technically feasible new system, which can be built or bought.

There was not enough detail in the use case system model for the key stakeholders to make an informed choice about solution design direction. More detailed feasibility analysis data was necessary to perform an informed, detailed decision analysis. Data and process modeling was completed to derive enough requirements details to enable this design decision analysis. These details were fed back into the feasibility analysis, which finally became mature enough to consider build versus buy trade-offs of possible solution systems.

A comparison was made between:

- Do Nothing (maintain status quo)
- Build a new digital imaging repository
- Buy a digital imagining repository (and perform a competitor analysis)

These choices were placed in a table against the feasibility criteria. This feasibility comparison table represented the decision analysis. It contained the key capabilities and constraints that any viable solution system must adhere. Yet, a detailed requirements analysis had not yet occurred. As it turned out, it was unnecessary. The team found a selection of vendors with viable SaaS (Software as a Service) systems that met the aquarium's feasibility needs.

The key information about these service based systems and the company contacts were supplied to the aquarium's information systems managing director. He indicated delight in finding existing service system solutions that were even better then he and his staff indicated that they were willing to build or buy. Indeed, the cost of the service systems was lower than anticipated, as competition in the market had improved offerings over the last few years. Support of the system could be predominately handled by the SaaS company, relieving the aquarium of overhead which could be put to other use. The rest of systems analysis and design was picked up by the aquarium staff and they are well on their way to implementing their new solution system.

This whole activity was performed in less than two months, mainly by a novice team of graduate students. The results of this, and dozens of similar analyses, is that when problems, requirements, and solutions are part of a design conversation and are represented by design boundary objects, finding desired outcomes can happen quite quickly. Moreover, applying a fully detailed requirements analysis and systems design was not necessary, and would have been counterproductive. The trick is to obtain

the critical feasibility criteria about PRS combinations in a manner that accelerates the design conversation. Once enough insight is gained to make an informed decision, then the process can be propelled quickly to a terminus. Each step indicates whether there is a need for more detailed requirements analysis and design activity or if there is enough to finish and move on. Or, paraphrasing Jim Collins from *Good to Great*, great is the enemy of good, and good is good enough [40].

In the case of the Monterey Bay Aquarium, a light to mid-level amount of analysis was necessary to uncover a viable, feasible solution choice. Each organization's situation is different. Still, it is argued that the majority of the problems do not require, deep, heavyweight and systems design and requirements analysis to address their issues. A lightweight approach – which can be increasingly heavier and detailed – can be utilized to promote highly productive design conversations.

6   Conclusion

Design is more than facts and figures. The organic aspects of design invoke the need to inquire, reflect, build, change, destroy, refine, and eventually obtain the goals of those receiving the design product. Requirements inform, bound, and propel design. Yet, requirements are part of the design conversation. They are not immutable ideals that must be rigorously adhered.

A design conversation is a play between problems, requirements, and solutions. It must be bounded or no tangible, feasible results can be produced. Requirements reduce the problem and solution spaces in which design occurs. But, that is not enough. Requirements need to be a shared representation amongst the involved stakeholders. They must transform knowledge, mobilize action and most importantly, legitimize key design knowledge that is the basis for development. Requirements are design boundary objects. They are fundamental to enabling and eventually completing design conversations.

Not all design requires deep, detailed requirements. Often, the problems organizations face are lightweight. They need only be described with a few, well grounded requirements. These can be enough to find satisficing solutions. Yet, if the problem under examination requires more detail, depth, and insight to be understood and safely addressed, rigorous requirements analysis can be readily deployed. Hence, design, and specifically requirements analysis methodology can be adjusted as per the level of risk associated with the core problem.

To improve the design conversation, design requirements boundary objects need to be refined, updated, and invented anew. More research needs to be performed on determining how to improve the core attributes of DRBOs that would improve their design conversation performance against the design boundary object's four factors (Table 2). More simply, the speed and effectiveness of design knowledge enhancing artifacts need to be improved. This would be a vast improvement over the current general determination the success or failure of system design *ex post* – i.e. acceptance or rejection after deployment. Correspondingly, any improvements in early designs produce a geometric or exponential savings in resources and effort [41, 42]. In turn, this will free up resources to solve evermore problems, produce new systems, and further the art of design.

Acknowledgements

References

1. Bergman, M., King, J.L., Lyytinen, K.: Large Scale Requirements Analysis Revisited: The need for Understanding the Political Ecology of Requirements Engineering. Requirements Engineering Journal 7, 152–171 (2002)
2. Bergman, M., King, J.L., Lyytinen, K.: Large Scale Requirements Analysis as Heterogeneous Engineering. In: Floyd, C., Klischewski, R. (eds.) Social Thinking - Software Practice, pp. 357–386. MIT Press, Cambridge (2002)
3. Loucopoulos, P., Karakostas, V.: System Requirements Engineering. McGraw-Hill Book Co., London (1995)
4. Wieringa, R.: Requirements engineering: frameworks for understanding. Wiley, New York (1996)
5. Kotonya, G., Sommerville, I.: Requirements engineering: processes and techniques. J. Wiley, Chichester (1998)
6. Teorey, T.J., Lightstone, S.S., Nadeau, T.: Database Modeling and Design: Logical Design. Morgan Kaufmann, San Francisco (2005)
7. Elmasri, R., Navathe, S.B.: Fundamentals of Database Systems. Addison Wesley, Reading (2006)
8. Simon, H.A.: The Sciences of the Artificial. MIT Press, Cambridge (1996)
9. Okasha, S.: Philosophy of Science. Oxford University Press, USA (2002)
10. Grady, J.O.: Systems Requirements Analysis. Academic Press, London (2006)
11. Thayer, R.H., Dorfman, M.: Software Requirements Engineering. Wiley-IEEE Computer Society, Piscataway (2000)
12. Robertson, S., Robertson, J.: Mastering the Requirements Process. Addison-Wesley Professional, Reading (2006)
13. Pohl, K.: Process-centered requirements engineering. Wiley, New York (1996)
14. Booch, G.: Object-oriented analysis and design with applications. Addison-Wesley, Upper Saddle River (2007)
15. Podeswa, H.: UML for the IT business analyst: a practical guide to object-oriented requirements gathering. Thomson Course Technology PTR, Boston (2005)
16. Bergman, M., Lyytinen, K., Mark, G.: Boundary Objects in Design: An Ecological View of Design Artifacts. Journal of the AIS 8 (2007)
17. Sutcliffe, A., Gault, B.: The ISRE Method for Analyzing System Requirements with Virtual Prototypes. Systems Engineering 7, 123–143 (2004)
18. Whitten, J.L., Bentley, L.D.: Systems Analysis & Design Methods. McGraw-Hill/Irwin, New York (2006)
19. Carlile, P.R.: A Pragmatic view of Knowledge and Boundaries: Boundary Objects in New Product Development. Organizational Science 13, 442–455 (2002)

20. Karsten, H., Lyytinen, K., Hurskainen, M., Koskelainen, T.: Crossing boundaries and conscripting participation: representing and integrating knowledge in a paper machinery project. European Journal of Information Systems 10, 89–98 (2001)
21. Henderson, K.: Flexible sketches and inflexible data-bases: Visual communication, conscription devises and boundary objects in design engineering. Science Technology and Human Values 16, 448–473 (1991)
22. Ramesh, B., Jarke, M.: Toward reference models for requirements traceability. IEEE Transactions on Software Engineering 27, 58–93 (2001)
23. Markus, M.L.: Power, Politics and MIS implementation. Communications of the ACM 26, 430–444 (1983)
24. Fairholm, G.W.: Organizational power politics: tactics in organizational leadership. Praeger, Westport, Conn. (1993)
25. Hirscheim, R.A., Klein, H., Lyytinen, K.: Information Systems Development and Data Modeling, Conceptual and Philosophical Foundations. Cambridge University Press, Cambridge (1995)
26. Bergman, M., Mark, G.: Exploring the Relationship between Project Selection and Requirements Analysis: An Empirical Study of the New Millennium Program. In: IEEE (ed.) RE 2002, pp. 247–254. IEEE Press, Essen (2002)
27. Grudin, J.: Why CSCW applications fail: problems in the design and evaluation of organizational interfaces. In: CSCW 1988, pp. 85–93. ACM Press, New York (1988)
28. Grudin, J.: Groupware and Social Dynamics: Eight Challenges for Developers. Communications of the ACM 37, 92–105 (1994)
29. Graham, I.: Requirements engineering and rapid development: an object-oriented approach. Addison Wesley, Harlow (1998)
30. Lamsweerde, A.v., Letier, E.: Handling Obstacles in Goal-Oriented Requirements Engineering. IEEE Transactions on Software Engineering 26, 978–1005 (2000)
31. Markus, M.L., Majchrzak, A., Gasser, L.: A Design Theory for Systems that Support Emergent Knowledge Processes. MIS Quarterly 26, 179–212 (2002)
32. Gregor, S.: The Nature of Theory in Information Systems. Management of Information Systems Quarterly 30, 611–642 (2006)
33. Carroll, J.M.: Making use: scenario-based design of human-computer interactions. MIT Press, Cambridge (2000)
34. Scott, W.R., Davis, G.F.: Organizations and Organizing: Rational, Natural and Open Systems Perspectives. Prentice Hall, Upper Saddle River (2006)
35. Sorenson, O.: Interdependence and Adaptability: Organizational Learning and the Long-Term Effect of Integration. Management Science 49, 446–463 (2003)
36. Truex, D.P., Baskerville, R., Klein, H.: Growing systems in emergent organizations. Communications of the ACM 42, 117–123 (1999)
37. Cockburn, A.: Writing effective use cases. Addison-Wesley, Boston (2001)
38. Denney, R.: Succeeding with use cases: working smart to deliver quality. Addison-Wesley, Upper Saddle River (2005)
39. Chalin, P., Sinnig, D., Torkzadeh, K.: Capturing business transaction requirements in use case models. In: Proceedings of the 2008 ACM symposium on Applied computing. ACM, Fortaleza (2008)
40. Collins, J.C.: Good to great: why some companies make the leap–and others don't. Harper Business, New York (2001)
41. Sommerville, I.: Software Engineering. Addison Wesley, Reading (2006)
42. Boehm, B.W.: Software engineering economics. Prentice-Hall, Englewood Cliffs (1981)

Design Requirements for Communication-Intensive Interactive Applications

Davide Bolchini[1], Franca Garzotto[2], and Paolo Paolini[2,3]

[1] Indiana University-Purdue University Indianapolis (IUPUI), School of Informatics,
535 W Michigan St., Indianapolis, IN, U.S.A.
`dbolchin@iupui.edu`
[2] Politecnico di Milano, Department of Information and Electronics and School of Industrial
Design, P.za Leonardo da Vinci, 32 – 20133 Milano, Italy
`garzotto@elet.polimi.it`
[3] University of Lugano, Faculty of Communication Sciences,
Via G. Buffi 13 – 6900 Lugano, Switzerland
`paolo.paolini@polimi.it`

Abstract. Online interactive applications call for new requirements paradigms to capture the growing complexity of computer-mediated communication. Crafting successful interactive applications (such as websites and multimedia) involves modeling the requirements for the user experience, including those leading to content design, usable information architecture and interaction, in profound coordination with the communication goals of all stakeholders involved, ranging from persuasion to social engagement, to call for action. To face this grand challenge, we propose a methodology for modeling communication requirements and provide a set of operational conceptual tools to be used in complex projects with multiple stakeholders. Through examples from real-life projects and lessons-learned from direct experience, we draw on the concepts of brand, value, communication goals, information and persuasion requirements to systematically guide analysts to master the multifaceted connections of these elements as drivers to inform successful communication designs.

Keywords: Infosuasive applications, communication goals, persuasion, brand, stakeholders, users, key values, requirements taxonomy, hypermedia design, web design.

1 Introducing "Infosuasive" Reasoning

The web has already made its transition from information to communication medium. It has become one of the main vehicles for commercial and non commercial "entities" to reach the global society and to establish or promote their "brand" in the global economy [1],[2]. Through the web, companies, educational or cultural institutions, charities, governmental bodies, politicians, artists, and many other subjects, offer services, inform, communicate and engage their stakeholders, build and maintain a relationship with them, and attempt to influence their attitudes and behavior. The goal of many modern web applications is at the same time informative – i.e., to support knowledge needs, operational – i.e., to support transactions such as buy, sell or make

reservations, social – to connect people, and persuasive – i.e., to change user's opinions, attitudes and behaviors [3].

This trend has progressively increased the complexity of the requirements and design space and introduced a number of novel issues that imply a rethinking of our current development approaches.

In particular, we focus on systems whose main goals are informative and persuasive, calling them "infosuasive" applications. We address the very early design stage of infosuasive applications, when concepts are initially formulated vaguely, are somewhat unfocused and inaccurate, and must be progressively organized and refined to become more precise design solutions. This stage of the development process must give voice to a variety of actors, being them application stakeholders or members of the design team. This is due to the multi-facet nature (informative and persuasive) of the goals of the applications, and to the fact that the contexts in which the system must be "placed" is potentially more complex to frame. Whereas in the past, web applications were conceived for a known business and social context, such a "clear-cut" context is oftentimes lacking today [4]. In the global society, many, mutually influencing issues have to be understood and decided on during the very early design process, which include cultural, social, psychological and ethical dimensions (beside strategic, marketing, and technological aspects that were normally taken into account in the past). As a consequence, this activity requires more and more an interdisciplinary approach involving competences in web engineering, interface design, marketing, branding, ethnography, communication science (and perhaps others).

The main contribution of this chapter is to propose a conceptual framework for infosuasive requirements, that supports the members of the design team (marketers, brand designers, communication designers, graphic designers, information architects, technology experts) and all the relevant stakeholders (clients, strategic decision makers, financing partners) to share their thoughts, to integrate their different viewpoints, to organize the variety of issues that need to be analyzed, to find a direction in the numerous design options, and to finally represent the results of this activity in an effective way.

Our approach is value-driven since it is centered around the concept of value, regarded as a means to achieve given communication goals on specific communication targets. We place the analysis of these aspects in the wider context of web requirements engineering, highlighting their relationships with business and techno-organizational analysis and user needs analysis. We then pinpoint how values and communication goals impact on various design dimensions of infosuasive web application - contents, information architecture, interaction, operations, and lay-out. Our work is inspired to goal-based and value-based requirements engineering, brand design methods, and value-centered design "frameworks" (as proposed by the HCI community).

A complementary driver to model communication requirements comes from the notion of brand, which tightly interacts with the consideration of values during development.

In fact, as an organization or institution has to (re)position itself on the market, it has not only to define or rethink the overall business strategy, but also (re)shape the communication towards the various stakeholders (current and potential clients, shareholders, other institutions, organizations, sponsors, etc.). In this process, the concept

of brand in its multiple declinations – e.g., brand image and brand experience – plays a crucial role. If, etymologically, brand just means "identification mark or sign" (given to a product or service and reified by a name associated to a visual sign, symbol or logo), its conventional meaning is much richer than this. For a company or an institution, the brand is "who we are, what we believe, why you should trust us" [5]. It is the vehicle for communicating "a promise that the company or institution can keep to all its stakeholders: its customers, trades, stockholders, employees" [6].

The persistent communication of this promise brings to the creation of a brand image. It is a set of symbolic constructs within the minds of people that consists of all the information, expectations, values, emotions, or attitudes that "consumers" generally, both locally and globally, associate with an "entity" – being it a product, service, company, institution or, at a broader level, a country or a culture.

Carefully integrating values and brand-related considerations in the requirements and design process is at the core of the proposed framework, which aims at providing conceptual tools to conduct requirements elicitation, analysis and design for communication-intensive interactive applications in complex organizational contexts.

2   Background: Branding and Value-Driven Requirements

The term "value" is broad, and it is outside the purpose of this chapter to discuss the moral, philosophical, psychological or economical foundations of this concepts (the reader is referred to [7] for an overview.). Our less ambitious goal in this section is to review some design approaches in HCI, e-commerce, requirements engineering, web engineering, e-branding, in which the notion of value has been explored.

Values sensitive design (VSD) [8], [7] emerged in the mid '90s in the HCI community as an approach to the design of information and computer systems that accounts for human values in a principled and comprehensive manner early and throughout the whole design process. Value sensitive design particularly emphasizes values with moral import, including privacy, trust, respect for intellectual property rights, freedom from bias, moral responsibility, honesty, democracy. Many works in VSD exemplify how different aspects of web design can account for such values. Rather than a "methodology" in engineering sense, VSD is intended as a "framework for understanding" [7] how specific values play out in the overall design process, and how these values can be undermined or promoted by the technology, thus shaping (but not rigidly determining) individual and social behavior.

Value centered design (VCD) [9] shifts the focus from "value as human belief" (as promoted by VSD) to "value as worth", that is, whatever some people somewhere find worthwhile, individually or collectively, irrespective of ethics, wisdom, style, taste, etiquette or the approval of others. Values are regarded as a motivator for investing time, money, energy, or commitment in the development or use of a web product or service by all (direct or indirect) stakeholders. This approach is still in its infancy, and the proposed VDC "process" is still quite general. It basically suggests to iteratively identify and evaluate the benefits (either economical, or emotional, or affective) gained by the end user (either as an individual or as a collectivity) by effect of the experience with the system under design. As stated by its inventors, "much needs

to be done to move from a set of arguments and a plausible development framework to proven approaches"[9].

The VSD concept of "technology as value promoter" has been integrated in a broader design approach known as persuasive design [10], which focuses on how to design technological artifacts that change attitudes and behaviours. The reference book on this subject [10] provides a number of general persuasion guidelines; it also exemplifies how they are used in existing web applications, e.g., to foster reliance, credibility and trust, or to instill some human values in web users (such as environmental attention), or to induce specific habit changes in users' life.

To facilitate persuasion, emotion is acknowledged as playing a major role. Emotional design [11] investigates how emotions during a product's experience can create value for the user (e.g., pleasure, fun, calmness, trust), which in turn results into a value for the product developer. Although the vision of emotional design covers any, physical or virtual, design object, web artifacts are one of the main domains for this approach, and [11] reports many impressive examples of the seductive, persuasive power of creating emotions through proper interface solutions.

The notion of value for persuasion also plays a key role in brand design, also named e-branding [12] when applied to the web. The notion of "brand" is perhaps as broad and general as the concept of value. The web has strengthened and expanded the economic relevance of branding concepts not only for e-institutions (i.e., those existing only on the web, e.g., Amazon) but for any commercial and non commercial entity that wants to establish or promote their "brand image" in the global society. The web has also fostered a rapid evolution of brand design basic principles, creating new opportunities of brand expression and offering new means to shape the "promise of values" that are relevant to user's expectations and desires [13].

Most of the current research in the above areas is stimulating but tends either to be overly abstract, or to solve the different issues through an anecdotic style of investigation. It lacks a conceptual or procedural framework that might guide a design team and make an approach easily re-usable. Above all, they do not provide any systematic guidance to reflect the different high-level value issues onto the dimensions of the design space.

Value based design (VBD) [14], a recent approach emerged in requirements engineering and in web engineering, is more pragmatic and systematic. It looks at the notion of "value as worth" from a strictly business perspective, i.e., in terms of the economic benefit that is induced by a system and makes the company or institution more competitive and profitable. Recently, VBD was applied to the design of e-commerce systems.

The e3value model [15], [16], [17], for example, provides a conceptual framework for representing and analyzing business models for e-commerce, in terms of a network of actors and enterprises creating, distributing, and consuming things of economic value through the web. e3value represents the economic interest of various stakeholders from multiple perspectives: the business value viewpoint, the business process viewpoint and the software architecture viewpoint. e3value modeling techniques can be combined with goal-based requirements modeling (using i* [18]) to

support designers in creating, representing, and analysing business models and stake-holders goals for e-services in a more comprehensive way [15]. Tongrungrojana & Lowe [19] integrate high level business value requirements specified in e3value to lower level detailed design using WebML+, a formal extension to an existing web modeling language (WebML [20]). For a similar purpose, the VIP model [21] pro-vides a UML based modeling framework that integrates e3value constructs with WebML.

In a previous publication [22], we discussed the rationale for taking into account branding issues in the web design process, and propose brand values as first order citizens in a web requirements modeling framework – named AWARE+ [22]. AWARE+ extended a previous requirements model - AWARE (Analysis of Web Application Requirements [23]) – that exploited a goal-oriented RE approach specifi-cally for web applications. AWARE balanced the consideration of users' needs and other stakeholders' goals; these are operationalized into application requirements through refinement and decomposition processes, whose output is fed into a subse-quent design activity. An original feature of the method is the use of a hypermedia design taxonomy to categorize requirements and to facilitate the organization of the design activity [24]. Based on the experience gained in a very large web project in e-tourism [25], we built AWARE+ [22], which focused on content-intensive web appli-cations with the purpose of bridging high level communication requirements with web design concepts. We intensively used and tested AWARE+ in number of successive projects involving large design teams. From such experience we built a new, substan-tially revised version of the framework, which is presented in this chapter.

The current version captures our deeper understanding of the communication and persuasion power of the web and how communication and persuasion elements play with other factors in the very early design process of infosuasive web applications. With respect to the previous version, the main novelty of the new release of AWARE+ relies upon the definition of infosuasive web application; the introduc-tion of communication analysis in requirements management; the adoption of "val-ues" as modeling primitives for communication and persuasion purposes; and the definition of clear relationships among values, communication goals and communi-cation targets.

3   Eliciting and Modelling Communication Requirements

In the definition of the requirements space for infosuasive web applications, each of the many actors in the stakeholder picture – clients, strategic decision makers, marketers, persons responsible for business process development, brand designers, communication designers, graphic designers, information architects, technology managers - contributes with a different perspective, grounded in differences in skills, responsibilities, knowledge and expertise, and culture in a broad sense. The requirements analysis is to be carried on from different viewpoints; each of them is initially self-contained, encapsulates partial, high level knowledge about the problem domain andthe relevant stakeholders, and is typically specified in a particular, suitable representation language. The results of the

various viewpoints eventually need then to be combined to inform lower level design decisions. Traditionally, three main viewpoints are considered.

Business analysis addresses the problem from an economical perspective using for example a model like e3value. This activity is directly related to the institution's strategic vision, sets the business goals and constraints of the application, defines the expected value (in commercial terms) for the various stakeholders, and defines the characteristics of the business model.
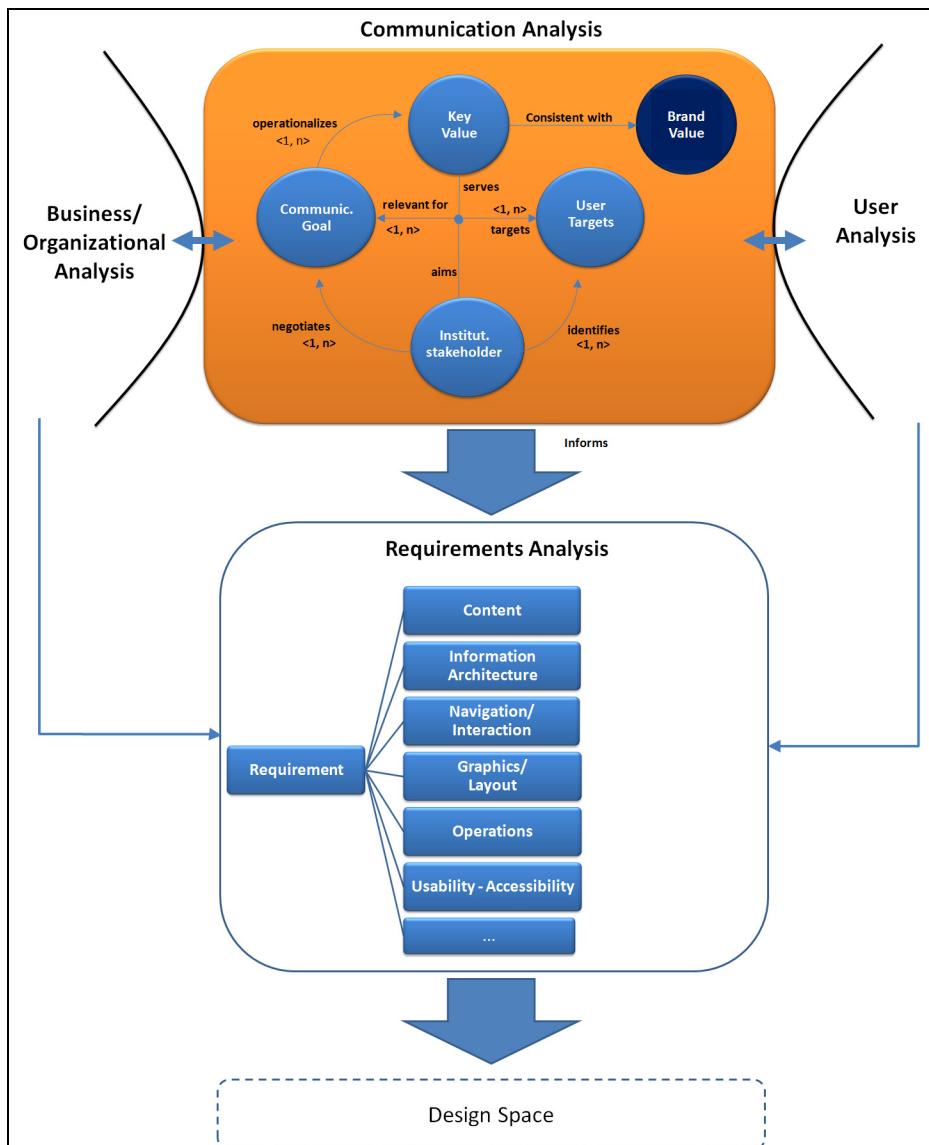
Techno-Organizational analysis [4], [26], [27] investigates the context in which the web application is built and conceived. It explores all the elements that, together, define the "culture", the structure, and the dynamics of an organization: the organizational rules and constraints; the organization "tradition"; the schemas, norms, and routines together with associated activities and resources; the relationships between the organization and its social, political, institutional, and economic environments.

End-user analysis identifies various aspects, including: the information and operational needs of end users; the context in which the application is intended to be used; the motivations for using the application, as well as user values as desired qualities of the interaction (e.g., usability, security, accessibility). These elements can be elicited through user research (ethnographic techniques, questionnaires, interviews, focus groups, or participatory design methods [28], [29]) and can be further elaborated using scenarios, task analysis, goal-based RE approaches [18], [23], [30], or similar conceptual tool.

The persuasive dimension of infosuasive applications introduces the need for a new form of analysis, which we refer as communication analysis. This activity, which is crucial for infosuasive applications, is the focus of our approach (Figure 1). Figure 1 pinpoints the key modeling concepts of communication analysis - communication target, key value, brand value. It highlights the relationships among such elements and the requirement space that defines the functional and non functional characteristics of a web application, in terms of content, information architecture, lay-out, operational and socialization services, and other non functional aspects (e.g., security, usability, accessibility). Whereas figure 1 summarizes the conceptual elements, in practice the communication analysis may adopt agile matrixes and tables to document the output of the different communication analysis tasks. As it will be shown in the various examples, a tabular notation is very agile and readable especially to support elicitation and brainstorming with the stakeholders.

Communication analysis involves the elicitation of specific aspects but also leverages upon the knowledge information that is generated from the other forms of analysis. Business, organizational, and end user analysis export the relevant input for communication analysis (hiding the details on how such knowledge is elicited and modeled). Requirements of different nature are then informed by the knowledge produced by all forms of analysis: the same requirement or design property can be motivated at the same time by a business, informative, operational or organizational goal, and from key values and brand values resulting from communication analysis.

The novelty of our approach is to identify the role of communication analysis in the overall requirements process, and to elaborate the key elements that participate in this activity. Before discussing the above concepts more precisely, we will briefly describe a case study that will be used to exemplify our approach.

Fig. 1. At a glance: AWARE+ for "infosuasive" applications

4   A Real-Life Case Study

In January 2007, the University of Lugano - Switzerland (hereinafter USI – Universita' della Svizzera Italiana) commissioned to the authors the redesign of the communication and technological web infrastructure of the whole university. Particularly, the project involved the complete rethinking and redesign of fifty websites related to the University of Lugano (one official university website and one website for each of

the four faculties, plus a variable number of websites of laboratories, institutes, projects, and university services). The motivation for this project was driven by a number of factors, including the "aged" look&feel of the websites (designed four years earlier), the clumsy information architecture resulting from continuous micro-changes occurred over the years, technological considerations (need of technology update), content quality issues (no proper workflow for quality control was in place), and other contingent organizational reasons.

Over a period of 8 months, the new version of AWARE+ has been successfully applied to the early-stage of requirements analysis and design of the main university website (launched online in December 2007) and of the faculty websites (ongoing design), and will be used as well for the other new websites to be developed within the context of the project. The project directly involved various representative stakeholders (the USI President, the Administrative Director, faculty representatives, faculty members, service representatives, and the communication and media office), a multidisciplinary design team (including requirements analysts, web designers, software engineers, web programmers, graphic designers, web editors and information architects) and user representatives (students).

5   Modelling Communication Requirements: Key Primitives

5.1   Communication Targets

Communication analysis starts from the consideration of the profiles of the communication targets, i.e., the user communities the web communication action is directed to. The definition of target profiles is directly related to the client strategic vision, is under the direct responsibility of top level institutional stakeholders, and is partially informed by the output of business analysis. Still, it is based on communication criteria and not on strictly marketing considerations, and does not necessarily coincide with the definition of market segments [28]. Whereas the overall purpose of the market segmentation and targeting is to identify groups of similar (existing or potential) "customers", the intent of the user segmentation based on communication purposes aims at capturing all relevant external stakeholders who may be of interest for the communication action, even if not necessarily being potential customers.

For the USI website, the primary communication targets have been elicited in the discussion with the top level institutional stakeholders: the President, the General Secretary and the Promotion Office Representatives. The sixteen communication

Table 1. Communication Targets for USI website

COMMUNICATION TARGETS	
Internal students	Bachelor, Master, PhD, Alumni
Prospect Students	Bachelor, Master, PhD
Others	Students' Families, Funding Agencies, Canton, Confederation, Other Universities, Outside researchers, Outside colleagues, Prospect Partners. Local Media, National/International Media

targets are reported in Table 1 and conveniently grouped into "Internal students", "Prospect students" and "Others".

Each target represents a user community the University would like to engage and maintain a successful dialogue with through the website in order to achieve its communication goals, as discussed next.

5.2 Communication Goals

Communication goals define the persuasion purposes, i.e., the effects that must be achieved on the users in terms of adoption or change of ideas, attitudes, behaviors, i.e., what Pierce called a "habit change" [31]. Defining the communication goals is a critical activity, which allows hidden or non explicit objectives to surface in a clear and very distinct fashion. As for communication targets, also the definition of communication goals is directly related to the client strategic vision, and is under the responsibility of the institutional stakeholders. Communication goals are not the business goals specified by the business analysis, but may act as a vehicle to achieve

Table 2. Communication Goals for USI website

COMMUNICATION TARGET	USI COMMUNICATION GOALS (attitudes or behavior to induce on the different targets)
Internal students	
Bachelor	Feel satisfied as members of a community; Act as a "word of mouth" recruiters; Feel encouraged to pursue their studies with specialization curricula (masters).
Master	Feel satisfied as members of a community; Act as a "word of mouth" recruiters; Feel encouraged to remain in touch with USI after the completion of their studies.
PhD	Feel satisfied as members of a community; Act as a "word of mouth" recruiters; Feel encouraged to remain connection points toward USI from their future positions.
Alumni	Feel encouraged to keep in touch Help for recruiting new students, for setting up internships, for corporate and institutional relationship, for "word of mouth" branding
Prospect Students	
Bachelor	Feel that USI as an attractive place to come for studies
Master	Feel that USI as an attractive place to come for studies
PhD	Feel that USI as an attractive place to come for studies
Students' Families	Feel that USI as a safe and constructive place to come for studies
Funding Agencies	Believe that USI provides a key contribution to the growth of the Swiss scientific arena as a place for research excellence.
Canton	Believe that USI as one of the key players for the growth of the "Ticino scientifico".
Confederation	Believe that USI is a key academic bridge to Italy. Believe that USI is an added value in the Swiss univ. panorama because it represents the Italian-speaking part of the Confederation.

one or more business goals, and should be consistent with them. By their own nature, communication goals are directed to specific targets, and must be differentiated according to the different target profiles [32]. One of the key principles of communication and persuasion, which the web channel can support better than traditional media, is "tailoring": a persuasion action is more effective if it is personalized, i.e., tailored to the target profile, in order to build a customized relationship between the receiver and the sender [6], [11].

The USI stakeholders needed to elaborate the overall purpose of the communication towards each different communication target. Institutional stakeholders can express their own communication goals for each communication target; even better, they should possibly reach an agreement, or a compromise, which allows coming up with a few set of shared communication goals. A coherent, agreed and manageable set of communication goals should reflect a clearly directed communication strategy and paves the way for a more focused design effort, thus potentially leading to a more effective persuasive action.

As shown in Table 2, the diversification of the communication goals for the different targets is quite rich. The guiding questions to elicit such goals in our project have included the following: "How does the university want to be perceived by each user community?" (see goals for the Prospect students, Students' Families, the Canton and the Confederation); "What actions or behaviors can users undertake that can be beneficial for the university"? (see for examples the goals for internal bachelor students).

5.3 Key Values

Key values are a way to operationalize communication goals into more pregnant "messages" and "perceptions" that need to get across to each communication target. Values may be very diverse in nature, but they may emerge from a common line of reasoning, stimulated by the question: given the communication goals that have been stated, what can the entity promise to be to each communication target, in terms of valuable qualities and attributes, to facilitate the achievement of the goals?

A key value can be a moral, ethical, social, or cultural belief which an entity is committed to. Environmental sustainability, for example, is a value in this sense. Values of this nature are important for persuasion purposes since people tend to identify themselves more easily with entities that share with them some common fundamental beliefs. As well known to social psychologists and brand designers, when an entity plays to our values about ourselves and the society, we experience it positively [11]; value sharing is a trigger for human connection and enforces the rational or emotional relationship between the message sender and the receiver.

A key value can be a quality of an entity (being it a product, a service, a company, an institution, a person or, at a broader level, a country or a culture) that is worthwhile for people, either at individual or collective level. This quality is not necessarily functional, but can be something that gives rise to positive emotional or affective effects [5], [33]. For example, "excellence in teaching" is a value (intended as functional quality) that can be associated to a university (see USI example below). "Eco-chic" or "exclusive" can be a value (intended as emotional quality) associated to a tourism resort. The definition of key values is a complex process, which requires a deep

Table 3. Key values ("messages") for each communication target

COMMUNICATION TARGET	Key Value 1	Key Value 2	Key Value 3	Key Value 4
Internal students				
Bachelor	Identify with USI as global entity	Friendly and familiar	We are transparent	
Master	Identify with USI as global entity	Awareness of the USI community	We support your career	
PhD	Identify with USI as global entity	Awareness of the USI community	We support your growth	
Alumni	Identify with USI as global entity	We are changing	We need your help: get involved	
Prospect Students				
Bachelor	Quality of education	Humanized, familiar (good relations with teachers)	Effective, innovative	Safe environment
Master	We focus: well specialized curricula	International, multilingual	Top quality teachers	
PhD	We focus on specific high level areas	Well connected to other institutions	Efficient organization	
Funding Agencies	Excellence of research	Well connected	Very young, multidisciplinary	
Canton	Competitive research (at international level but also responding to national or local research mandates)	High quality education	Well connected to local institutions	
Confederation	Strong internationalization (Faculties and Students)	Academic bridge to Italy. USI as Italian-speaking part of the Swiss academia.	Young and modern	Agile and effective governance
Other Universities	Competitive research	Up to international standard	Well connected in Switzerland and international	
Outside researchers	Competitive research	Well connected	Excellent and familiar environment	
Outside colleagues	Competitive research	Well connected	Excellent and familiar environment	
Prospect Partners	Excellent of research	Efficient	Agile and well organized	
Local Media	USI is a center of culture	USI is well connected to local institutions	Innovative	
National/International Media	Very active	Excellence of research	Innovative	

understanding of multiple factors, including the social, psychological, cultural characteristics of the communication targets; their attitudes, desires, trends, beliefs. It also entails an understanding of the characteristics of the physical and organizational context is which their experience will take place (in order to build a promise that can be fulfilled), as well as attitudes, desires, beliefs of institutional stakeholders.

Thus the definition of key values may leverage upon the knowledge resulting from communication analysis as well as all the other analysis activities, as shown in figure 1. On one hand, key values may reflect the organizational values of an institution. They may be conceived in view of the achievement of one or more communication goals for given targets. It is likely that a market repositioning or a change of business goals will reflect into a change in the key values definition. On the other hand, key values should match the desired or expected values of end users, which

may result from user analysis, as suggested by value-centered design approaches. Key values should be therefore ideally elicited involving both end users and institutional stakeholders. They represent the common ground (between users and institutional stakeholders) on which to operate to facilitate the success of the communication.

The process of key values definition is not a simple one, because it involves strategic thinking, realism, and, at the same time, an immediate perception of the communication effectiveness to be achieved. To stay on track during key value elaboration with the institutional stakeholders, these persons need to be constantly reminded that key values are not only abstract principles but they should be functional to the achievement of the stated communication goals. In order to shape the key values for each user segments, the strategic thinking involved here consists in selecting those traits of the entity "personality" which are a value for a target, so that a communication action can leverage on them to achieve its persuasion objectives.

As a practical technique, for the definition of key values in our project we gently forced stakeholders to stick to not more than 3 or 4 for communication target. The risk was to have a long list of values so that the communication could lose focus and thus effectiveness.

This simple constraint on the number of key values implied a considerable effort during the negotiation with the various stakeholders and among the design team. However, it turned out that the same fact of reasoning towards specific niches of targets and communication goals, instead of referring to the "generic" brand of the university, already helped a lot the team in shaping clear, sound, and agreed key values.

For example, in order to induce internal bachelor students to act as "word of mouth" recruiters", we needed to make them feel part of USI as a whole (beyond the boundaries of their specific class or course programme), and to foster the perception that USI is a familiar and friendly environment. Values such as "sense of belonging" and "human sized context" were recognized as functional to specific communication goals and were therefore included among key values.

5.4 Brand Values

Key values should be consistent and aligned with the general (pre-existing, if any) brand values elaborated by the institutional/corporate communication and marketing experts. For example, USI had general brand values such as "international, innovative, interdisciplinary".

The brand values – typically elaborated by brand experts – represent the "core message", the "priority values" that more than any other value define the identity and the personality of an entity. They contribute to the definition of the "brand image" - the set of beliefs, emotions, attitudes, or qualities that people immediately associates to an entity in their mind when they think of that entity. Brand values are those with the highest potential of hitting a conscious or unconscious level, and of remaining as long lasting imprinting that endures after the real or digital experience with the entity itself. Brand values are the elements that will be reified, during design, into the few visual constructs (symbolic or textual such as logo or "motto") through which an entity will be identifiable under different conditions and "you will recognize it as yours" [6].

To understand the relationship between key values and brand values, it is important to note that:

- Key values are more tactical (relevant for the contingent project at issue), brand values are more strategic and long-term. In other words, brand values are more "stable": if a change of business goals may easily induce a modification in key values, a change of brand identity values must reflect a more radical and profound transformation of the whole entity.
- Key values can be addressed and elaborated by web engineers; brand values are typically the results of the work of brand, communication and marketing experts.
- Brand values are less directed towards a specific target, but should be appropriate for almost all of them.
- Brand values may not be defined and elaborated through elicitation: if an entity has already a well identifiable "brand", they are already explicit.

Overall, brand values must be pervasively and persistently communicated across the entire application, while different key values may be addressed in different portions of the application, devoted to specific targets and communication goals.

5.5   Requirements and Transition to Design

How do the elements discussed so far impact onto lower level design decisions? To smoothly support the transition between communication analysis and application design, requirements are identified and classified according to a hypermedia design taxonomy, that defines the design "dimensions" on which the various communication analysis elements may have an impact (see figure 1). These design dimensions reflect a conventional classification of design features as defined in most existing web design models [20], [22], [34].

Content requirements indicate the characteristics of the core information elements to include in the application. This category of requirements may have multiple sources: content needed to support operational tasks (such as "finding course information"); content necessary for informative reasons or institutional constraints (e.g. University regulation); "strategic" content descending from communication analysis and based on specific brand assets. This is the content which should have a communication impact (e.g. convincing about the excellence in research) on the user in the light of the communication messages expressed by key values.

Information architecture requirements define the characteristics of the overall structure of the content (including access criteria, navigation paths topology, hierarchical position of the different content elements, etc.).

Interaction and navigation requirements specify navigation patterns (such as "index" or "guided tour"), interaction paradigms (e.g., "menu based") or communication formats (e.g., "storytelling").

Layout requirements refer to the application presentation, i.e., to the visual and "look & feel" properties of the web interface, including chromatic style, elements allocation on the screen, visual priority and affordance, logo characteristics, etc. Brand values have typically a strong impact on this dimension.

Operational services requirements correspond to conventional functional requirements on the operations performed by the application or made available to the user to achieve his or her operational goals.

Socialization services requirements are a special type of operational requirements that address the social dimension of the web, defining the characteristics of services devoted to "connect people" and perform social task.

According to our model (see again figure 1), requirements are also considered as to their relevance to the *delivery channel* (e.g., stationary PC, PDA, mobile phone, web TV, etc.) by which the user can experience a given message or use a service .

In principle, the output of communication analysis informs all the above types of requirements, since any characteristics of an application may be exploited, in principle, to transmit or enforce a given "message" (brand values and key values). In the rest of the chapter we illustrate the impact of the proposed framework on content and layout/graphics requirements, as examples of the interaction between the communication analysis, the requirements and the design space.

5.6 Informing Content Requirements

Content is usually defined in terms of a set of typed or non typed multimedia "content units". These may be built on the basis of the input of business, techno-organizational or user analysis (see again figure 1). Indications for shaping the content can come from goals and values of different user profiles, from values and goals for different market segments, or from organizational goals and values of different components of the organization (the latter elements determine, for example, which content must be published for legal or organizational constraints). Another potential source for fruitfully brainstorming about content requirements is benchmarking analysis or pattern analysis, which provides a comparative description of the content design solutions of other similar entities (e.g., university websites, in USI case study). This set of content units (see a snapshot of it in the rows of Table 4) represents a preliminary set of coarse grain content pieces, at different level of abstraction, for the application under design.

We then cross content units with key values and communication targets to highlight the persuasion semantics of the different content units (their "message"). The resulting representation requires, in principle, a 3D matrix, but in practice a combination of two 2D matrices is more lightweight and readable, as shown in Tables 4 and 5. Coding key values (e.g. 1, 2, 3 …) may help make reference to them in other documentation contexts and bring them over onto the further steps of the design process.

Crossing content units with key values independently from the targets (Table 4), and analyzing them from different perspectives, highlight a number of content requirements issues that are crucial for guiding the design process:

- *Analysis by rows*. A row indicates which key values should be communicated through a given content unit. For example, the content unit about the university "Campus" can lend itself to be effectively used as a "persuasion moment" to convince the user about the key values "Friendly and familiar, safe environment" (key value 2) and "We are changing / we are evolving / we are active" (key value 6).

Table 4. Crossing key values with content units in USI

No	Content unit	Key Values					
		1	2	3	4	5	6
1	Welcome	x	x				x
2	About the university						
3	Organisation	x		x			
4	Mission statement			x	x	x	
5	Statistics			x			
6	The University and the city ["The city name"]		x				
7	Campus		x				x

Identification with USI

Friendly and Familiar

Transparent

We are changing/active

- *Analysis by columns*. A column indicates which content units can we leverage upon to communicate a given key value. For example, column 3 indicates that key value 3 -"we are transparent" (i.e. a transparent institution at many levels, from internal procedures to communication) should be supported though content units "organization", "mission statement", "statistics". In other words, these content units should convey the message aimed at persuading users about the "transparency" of the university at various levels.

Crossing content units and key-values is useful also for checking the consistency and completeness of contents with respect the overall set of high level goals. If a content unit is not related to at least one key value, it means that it is not functional to persuasion purposes. Still, it may be useful for the fulfillment of other goals. If no other reasons for its existence are found, it is likely to be removed. Similarly, it is highly desirable that each key value finds a place in the content to enact its persuasive power. If no content unit is found for a key value, it may be advisable to brainstorm about a new content unit to add; alternatively, it may be discussed whether that key value can be communicated through other requirements (e.g. graphics or layout), or instead dropped out from the key values list. To use a biological metaphor, the content here acts like a "growth medium" for a key value, i.e., it is what provides the nutrients necessary to the growth of and enactment of key values. Being it through a compelling text, engaging pictures, audio or videos, key values can be mainly exploited through the communication of content.

Fig. 2. An example of value-driven author template, to support content authors to write texts conveying the predefined key values

The specification of a consistent set of value-referenced content units represents a very powerful tool for supporting and coordinating content authors (who are typically spread across an organization and need to work in a distributed, but collaborative fashion) in the creation of the actual content. For example the person who is responsible for writing the text relative to a specific information unit, e.g. the research in the university, can be guided by the requirement that she must convey the messages expressed by the key values "innovative, modern", "excellence in research, with emphasis on competitive research", "well connected to other institutions", "multidisciplinary". To this end, specific author templates can be easily derived from the mapping content units → key values. The template for the author of the content unit should include, among other editorial constraints, also the set of key values to be communicated through that content unit (see Figure 2).

Content requirements concern multimedia material. For example, the person who is in charge of selecting the proper images for the same subject, should try to express the same key values.

In USI project, using the Content Units → Key-Values matrix we build a simple value-driven authoring tool. It is based on a set of structured "sheets", with each sheet - one per content unit – indicates the key values that should be "played" in the text argumentation and in the images associated to that unit.

The matching Content Units → Key-Values represents the expected persuasive impact that the stakeholders would like to get through the content, and is complemented by the matrix Content Units → Communication Target, which represents content requirements with respect to the expectations of different communication targets. A snapshot of this matrix for USI case study is shown in Table 5.

Table 5. Crossing content units with communication targets

No	Content unit	Description	All targets	Prospect students			Funding agencies
				Bac	Master	PhD	
1	Welcome	General presentation of the institution; the president's welcome.	xxx				
2	About the university	Profile, historical landmarks	xxx				
3	Organisation	Description of the way the institution is run, administrative bodies, governance					xxx
4	Mission statement	Mission, vision, goals in research and education.		X	x	x	
5	Statistics	Statistical data					x

Again, we gain different insights when elaborating and reading the matrix along the different dimensions. An analysis by rows pinpoints who is potentially interested in a given content unit. An analysis by columns pinpoints which content units are useful (for any purpose) to a given communication target. Through activities of user analysis such as interviews and focus groups, we can enrich this representation, e.g., filling each intersection with a quantitative indicator of relevance.
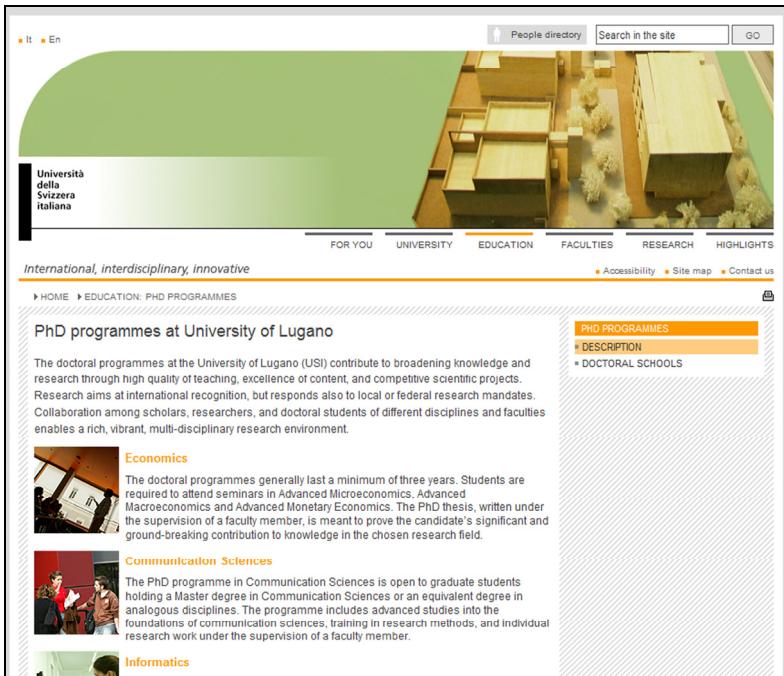
By iteratively filling in the matrix and reasoning on it among analysts and stakeholders about the decisions made, it becomes evident which content is an important focus of the application, since it serves a large number of communication targets. Blank rows should raise questions about the relevance of a given content unit ("Why should it be put in?"); blank columns should alert about the fact that we are failing to reach a given communication target (e.g. What are we offering to the local media?).

The association of content types to user targets is a common activity in user-centered design. In our approach, its role is mainly to support the validation the content requirements and communication analysis. In addition, it supports the bridge between communication requirements and information architectures requirements. For example, following the indications of the matrix, we can shape specific navigational access paths organized "by communication target group".

As shown in Figure 2 (left side popup menu), in USI we designed various groups of content units, each one specifically relevant for a given communication target (prospective students, current students, faculty and staff, etc.); these groups are aggregated into the section "For you", which is directly accessible from any page.

5.7   Informing Layout Requirements

Brand values inform many lay-out requirements, concerning the characteristics of all symbolic constructs - logo, pay-off, slogan, mottos, colors – that define the entity visual identity [6], and how they map onto the different pages. In most cases, visual

Fig. 3. An introductory page from the University of Lugano website redesigned, launched in December 2007

identity elements pre-exist in some form and must be simply adopted and consistently used in the lay-out definition in order to enforce the "corporate coordinated image" [35]. As previously mentioned, lay-out requirements deriving from brand values are pervasive across the entire application.

In USI project, the university pay-off ("International, Interdisciplinary, Innovative") is constantly present in all pages (see Figure 3), a requirements dictated by the brand identity of the university, as discussed in section 5.4. Similarly, the logotype (which is the institutional one) and the choices of the chromatic codes is a legacy of the brand values. Other requirements concerning the layout find their rationale in key values and in communication targets. Even if graphic designers normally hate working within a framed methodological guidance, these elements are typically at a level of abstraction that also strongly creative persons can accept, using them as guidelines for sketching their layout proposals. As it is visible from the page design proposal in Figure 2, all the overall layout profile was conceived to express the USI qualities "very young", "agile", "scientific excellence", "modern", "transparent", "at pace with times".

The choice of the thematic picture on top of the pages, which is different for each section, is for example guided by the need of evoking specific key values for a specific target. In Figure 3, a page of the section introducing PhD programmes for "Prospective Researchers", the image that have been selected should suggest the feeling of a "Friendly and familiar" USI.

It is interesting to notice how the adoption of a value driven approach has induced some organizational changes that are functional to make the value driven design project more efficient. For example, the whole USI database of institutional digital pictures was polished, reorganized, and enriched with new material, to better reflect the library of key values to be communicated.

6 Conclusions and Benefits

Introducing AWARE+ as a model for capturing and analyzing communication requirements provides a novel contribution and substantial benefits under several respects, including advances in requirements knowledge, transition to design, process support, and potential for adoption.

6.1 Advances in Requirements Knowledge

A substantial contribution is the introduction of the category of "infosuasive" applications as characterizing those websites (the greatest majority) that at the same time provide a large amount of information and try to persuade the user about something. In this perspective, the framework introduces and place "communication goals" as first order citizens to be considered together with other, more traditional, business goals during the process of requirements modeling and analysis.

More in general, the proposed methodological framework also contributes to smoothly bridge the early and late phases of the RE process of communication-intensive interactive applications, through the representation of the direct and indirect relationships between brand values, goals of different nature (business goals, conventional goals, and communication goals), target profiles, and application requirements. Having an explicit links among the above aspects allows design changes to be traced all the way to the originating source, i.e., the changes of high-level strategic decisions (including those that concern communication and brand) that ultimately lead to requirements.

The methodology presented in this chapter illustrates the need for modeling the intertwining of requirements and the social and organizational context in which the system-to-be will be eventually deployed. This aspect is particularly relevant for communication-intensive applications, where multiple stakeholders wish to communicate articulated messages to a variety of audiences for different purposes. The art and science of requirements is increasingly called for providing conceptual tools and guidance to master this complexity and smoothly make the requirements dialogue with the organizational environment. The case of university websites (which reflect the inner organizational complexity of the academic institutions) is emblematic in this respect.

6.2 Transition to Design

An important benefit of AWARE+ is that the complexity of the requirements apparatus gets smoothly coupled with the various design aspects to be developed. In fact, we have provided guidelines and examples about how to take into account communication requirements for the different parts of design, such as, for example, content design, layout design, and info-architecture design.

More in general, given the nature of the proposed approach, we are considering a picture potentially larger than the traditional RE focus on ICT applications. Once the communication goals and the corresponding segments of the target are identified, the choice of the "best" communication channel needs to be done. In this respect, we have also learned that a channel could be the Web, or pod-casting, but it could also be traditional communication media, such as leaflet, newspapers, magazines, posters, radio, TV, etc. This vision puts technological artifacts in a broader communication context, and assigns to requirements engineering a wider responsibility, which potentially includes the strategic definition of all communication actions and designs of a company and institution, with and without technological support.

6.3   Process Support

To support the requirements process in an agile fashion, we propose operational tools and documentation and elicitation heuristics to clearly express, in a concise manner, the ongoing outcome and decision making during the analysis. As organizational implication, by considering a wider spectrum of ingredients that inform design decisions, we foster an intense collaboration among different actors of the requirements analysis team: business strategists, marketing and branding experts, communication experts, designers, and software developers. This articulate set of stakeholders is the likely situation to be supported in complex and large projects.

6.4   Potential for Adoption

AWARE+ is a wonderful tool to build consensus in a complex community of stakeholders (as it can be found for a University website); the communication goals and values are so readable, in fact, that even the less expert stakeholders can understand them, express their opinion about them, and possibly accept them. Once communication goals and values are accepted, the design effort becomes much less arbitrary. From the direct project and teaching experience, we can confidently anticipate that the methodology is also lightweight and agile in two senses: it does not require much effort to adopt it and it does not require much effort to teach it to someone.

7   Agenda for Future Research

As far as future work is concerned, several are our current directions, but we only mention the most relevant ones.

7.1   Beyond Usability: Communication Impact Evaluation

Since we have set up precise communication goals, and we have transformed them into communication values, we are in the position to actually evaluate whether the wished impact is achieved or not. We can do this for different design components: e.g. does the content of this page convey the message(s)? Does the layout of this page convey the message(s)? Does the info-architecture of this part of the web site convey the message(s)? An important research direction is to expand current usability evaluation methods to include the analysis of the communication impact.

More in general, the scope of communication requirements evaluation is indeed quite complex, and substantial research is needed to explore the various levels of communication impact, including:

(a) *"Actual Communication" Level.*
This is the level of the *communication* between the application (as the ultimate reification of its stakeholders' communication intents) and the actual users. The complex of cognitive, emotional, rational, and contextual factors that come into play when users interact with a system (e.g., explore and read its content, look at its interface, or invoke its functions) is an increasingly explored research field. The purpose of our work is to measure the *final outcome* (and not the process) of this communication experience, in terms of *messages* that got across. At this level, evaluating communication requirements means to measure the *actual* values, beliefs, feelings, attitudes that the interaction with the application induces on the end user, and to compare them with the *expected* messages that stakeholders want to convey.

(b) *"Virtual Behaviour" Level.*
The communication impact can be revealed by *how* the users actually use the application. The *behaviour in the virtual world,* e.g., the users' interactions with the system, can be interpreted as enabling condition for user's beliefs and attitudes, and can be used as an indicator to assess whether the application has achieved its communication goals. Observable and measurable interactions can be, for example, the downloading of an online form, the access to key "conversion" pages (e.g., the advertised new product) or the execution of a "success" interaction flow that denotes the apparent user interest for specific aspects that are relevant for the main stakeholders. Various professional tools (based on log files data) exist that enable to set conversion measures on the user's behaviour (not only in terms of commercial transactions, but also in terms of visited content), and to track in real time the aggregated users' data. Google Analytics (www.google.com/analytics/) or ClickTracks (www.clicktracks.com/) are examples of popular commercial tools that are typically used for user profiling but can also support this level of communication impact evaluation.

(c) *"Real-world Behavior" Level.*
The most important level of communication impact, but, at the same time, the least under control by requirements engineers and application designers, is what users eventually do in the *real world* after having used an infosuasive application. For example, even if a university website effectively communicates the high quality of teaching, the advanced facilities and the friendly environment of the university (communication impact at level (a)), and users actually visit (even repeatedly) the expected content pages, and download the courses prospectus and application forms (communication impact at level (b)), this does not necessarily bring higher enrolment. Other factors, which are independent from the characteristics of the application, come into play in the articulate trajectory from communication to real-world behaviour. These elements are, for example, business opportunities and models (e.g., the university fee is to high), the competitive landscape (e.g., other universities are more attractive), or "environmental" factors that simply could be hardly controlled (e.g. the students' word of mouth about given courses is not positive, the bureaucratic

enrolment process is too complicated, the staff does not have enough time to properly assist the prospect students, or parents simply refrain from sending pupils too far from home).

Specific methods, process guidelines and metrics are needed to position the analysis on the desired communication impact level to assess and then to carry out a systematic evaluation against the pre-defined communication goals.

7.2 Requirements-Driven Guidance to Information Architecture

Whereas the impact of requirements on content and layout design decisions has been investigated in detail, still much research is needed to explore the interaction between communication goals, key values and requirements concerning information architecture and navigation. Traditionally, these requirements come from a different type of analysis, known and employed in HCI and human-centered design, through methods such as scenario-based design and information architecture design techniques.

The methodological tools for capturing the needs of the main stakeholders (centered around the notion of communication goals) and the ones used to model the needs of the users (user-centered design techniques) should point to a common ground and a common solution, in order to deliver a successful user experience. Co-operation and integration between these two worlds is needed. Information architecture requirements are a starting point to investigate this intersection: how communication goals influence the design of the information architecture, typically designed solely following usability criteria?

7.3 Transfer to Industry and Large-Scale Assessment

Further research is needed to more systematically assess the actual effectiveness of the methodology for real organization in complex projects. Is the framework effective and lightweight enough, that it can be easily adopted by practitioners in web development? We think that methodology-transfer is key item in the research agenda for any new methodological proposal. We are current discussing with a few industry representatives in Italy, in order to teach them the method and to assist them into adoption.

References

1. Pine, B.J., Gilmore, J.H.: The Experience Economy: Work Is Theater & Every Business a Stage. Harvard Business School Press, Boston (1999)
2. Wheeler, A.: Brand Identity: A Complete Guide to Creating, Building, and Maintaining Strong Brands. Wiley, Chichester (2006)
3. Holtzblatt, K.: Designing for the Mobile Device: Experiences, Challenges, and Methods. Communications of the ACM 48(7), 33–35 (2005)
4. David, M.W.: Co-design, China, and the commercialization of the mobile user interface. Interactions 13(5), 36–41 (2006)
5. Alben, L.: Quality of Experience. ACM Interactions 13(5), 12–15 (1996)
6. Andres, C., Fishel, C., Matson Kapp, P.: Identity Design Source Book. Rock Port Publisher (2004)

7. Friedman, B., Kahn Jr., P.H.: Human values, ethics, and design. In: Jacko, J.A., Sears, A. (eds.) The human-computer interaction handbook. Lawrence Erlbaum Associates, Mahwah (2003)
8. Friedman, B., Kahn Jr., P.H., Howe, D.C.: Trust online. Communications of the ACM 43(12), 34–40 (2000)
9. Cockton, G.: A Development Framework for Value-Centered Design. In: Proc. CHI 2003, pp. 1292–1295. ACM Press, New York (2003)
10. Fogg, B.J.: Persuasive Technology. Morgan Kaufmann, San Francisco (2003)
11. Norman, D.A.: Emotional Design. Basic Books (2004)
12. Gerstman, R.: Branding@thedigitalage, Palgrave (2001)
13. Marcus, A.: Branding 101. Interactions 11(5), 14–21 (2004)
14. Aurum, A., Wohlin, C.: A Value-Based Approach in Requirements Engineering: Explaining Some of the Fundamental Concepts. In: Sawyer, P., Paech, B., Heymans, P. (eds.) REFSQ 2007. LNCS, vol. 4542, pp. 109–115. Springer, Heidelberg (2007)
15. Gordijn, J., Yu, E., Raadt, B.: E-Service Design Using i* and e3value Modeling. IEEE Software 23(3), 26–33 (2006)
16. Gordijn, J., Tan, Y.: A Design Methodology for Modeling Trustworthy Value Webs. International Journal of Electronic Commerce 9(3), 31–48 (2005)
17. Gordijn, J., Akkermans, H.: Value based requirements engineering: Exploring innovative e-commerce idea. Requirements Engineering Journal 8(2), 114–134 (2003)
18. Yu, E.: Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering. In: Proc. 3rd IEEE Int'l Symp. Requirements Eng (RE 1997). IEEE Press, Los Alamitos (1997)
19. Tongrungrojana, R., Lowe, D.: WebML+: a Web modeling language for forming a bridge between business modeling and information modeling. In: Proc. SEKE 2003 15th Int. Conf. on Software Engineering & Knowledge Engineering, pp. 17–24 (2003)
20. Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., Matera, M.: Designing Data-Intensive Web Applications. Morgan-Kaufmann, San Francisco (2002)
21. Azam, F., Li, Z., Ahmad, R.: Integrating Value-based Requirement Engineering Models to WebML using VIP Business Modeling Framework. In: Proc.WWW 2007, pp. 933–942. ACM Press, New York (2007)
22. Bolchini, D., Garzotto, F., Paolini, P.: Branding and Communication Goals for Content Intensive Interactive Applications. In: Proc. Int. Requirements Engineering Conference RE 2007, pp. 173–182. IEEE Press, Los Alamitos (2007)
23. Bolchini, D., Paolini, P.: Goal-Driven Requirements Analysis for Hypermedia-intensive Web Applications. Requirements Engineering Journal – RE 2003 Special Issue (9), 85–103 (2004)
24. Bolchini, D., Paolini, P.: Interaction Dialogue Model: A Design Technique for Multichannel Applications. IEEE Trans. on Multimedia 8(3), 259–541 (2006)
25. MAPS Final Report - Marketing Activities for the Promotion of Syrian Cultural Heritage, Ministry of Tourism/Directorate of Museum and Antiquities - Syrian Arab Republic, EU Cultural Tourism Development Program, 01/2005 EC 119756 EUROP-AID
26. Anderson, R.I., Crakow, J., Joichi, J.: Improving the design of business and interactive system concepts in a digital business consultancy. In: Proc. DIS 2002 Designing Interactive Systems: processes, practices, methods, and techniques, pp. 213–223. ACM Press, New York (2001)
27. Scott, R.: Organizations: Rational, Natural, and Open Systems. Prentice-Hall, Englewood (1992)

28. Kuniavsky, M.: Observing the User Experience. A Practitioner's Guide to User Research. San Francisco (2003)
29. Muller, M.J., Kuhn, S. (eds.): Communications of the ACM – Special Issue on Participatory Design, vol. 36(6). ACM Press, New York (1993)
30. Dardenne, A., van Lamsweerde, A., Fickas, S.: Goal-Directed Requirements Acquisition. Science of Computer Programming 20(1), 3–50 (1993)
31. Peirce, C.S.: A Survey of Pragmaticism (1907)
32. Park, S., Harada, A., Igarashi, H.: Influences of Product Preferences on Product Usability. In: Proc. CHI 2006, pp. 87–92. ACM Press, New York (2006)
33. Gobé, M.: Emotional Branding. Allworth Press (2001)
34. Rossi, G., Pastor, O., Schwabe, D., Olsina, L. (eds.): Web Engineering: Modelling and Implementing Web Applications. Human-Computer Interaction Series. Springer, Heidelberg (2007)
35. Bassani, M., Sbalchiero, M.: Brand Design. Alinea (2002)

Requirements Engineering and Aspects

Yijun Yu[1], Nan Niu[2], Bruno González-Baixauli[3], John Mylopoulos[2],
Steve Easterbrook[2], and Julio Cesar Sampaio do Prado Leite[4]

[1] Department of Computing, The Open University, UK
y.yu@open.ac.uk
http://mcs.open.ac.uk/yy66
[2] University of Toronto, Canada
[3] University of Valladolid, Spain
[4] PUC-Rio, Brazil

Abstract. A fundamental problem with requirements engineering (RE) is to validate that a design does satisfy stakeholder requirements. Some requirements can be fulfilled locally by designed modules, where others must be accommodated globally by multiple modules together. These global requirements often crosscut with other local requirements and as such lead to scattered concerns. We explore the possibility of borrowing concepts from aspect-oriented programming (AOP) to tackle these problems in early requirements. In order to validate the design against such *early aspects*, we propose a framework to trace them into coding and testing aspects. We demonstrate the approach using an open-source e-commerce platform. In the conclusion of this work, we reflect on the lessons learnt from the case study on how to fit RE and AOP research together.

1   Introduction

Aspects are features of a software system that cut across architectural boundaries and impact on the design of multiple system components. Examples of aspects include data persistence and logging, security, performance, usability, maintainability, monitorability, testability, etc. Aspect-oriented programming (AOP) modularizes crosscutting concerns in software systems so that their evolution is less problematic [1,2]. One may apply AOP to reengineer a legacy software system, thereby obtain a library of aspects from an implementation that can be further configured and reused. Thus, uncovering aspects can be supported by program analysis techniques such as aspect mining [3], refactoring [4], and program visualization [5]. A common limitation to these techniques is that it is hard to validate identified and modularized code aspects: Were they really required or designed to be there?

Early aspects – the concerns that crosscut during the early stages of a software system's life cycle [6] are useful to study and analyze early on, before they are transformed into myriads of details that clutter code artifacts [7,8,9,10,11,12].

Although various approaches have been proposed to discover and modularize aspects at the requirements level [1], few address validation and traceability issues of identified

---

[1] http://www.early-aspects.net/ Last accessed on November 14, 2009.

early aspects. Specifically, it is of great interest to trace how early aspects manifest themselves and evolve throughout the software life cycle, and to validate the resulting system in light of stakeholder concerns that crosscut the problem domain.

In our previous work [9,11], we developed a framework for identifying and weaving candidate aspects during goal-oriented requirements analysis [13]. Aspects in goal models are captured as the operationalizations of softgoals and the relations to functional goals. These may be implemented as code aspects, but developers may choose other means to address these crosscutting concerns. Even in the latter case, it is desirable to keep early aspects modularized so that one does not have to recover them from the code at a later stage.

In this work, we investigate how to trace and validate identified early requirement aspects resulted from [9] through implementation and testing. Early aspects are either naturally mapped to code aspects, or recorded as issues to directly advise testing. Aspect testing is therefore guided by stakeholder goals. In our approach, aspects are intended to enhance system qualities by interacting with multiple system units, while preserving the functionalities defined by hard goals. The benefits of leveraging our approach are twofold. By separating crosscutting concerns throughout requirements, implementation, and testing phases, we achieve a high degree of modularity and traceability in software development. By validating implementation against stakeholder concerns, we achieve a high level of software quality and user satisfaction.

Our aim for this work is to explore the extent to which system implementation helps validate early aspects, and to propose a systematic means to trace the refinement of requirement aspects throughout the software life cycle. Our proposed approach to handling aspects is very general and does not depend on any specific programming language or particular type of applications. To demonstrate the idea, we present a study showing the approach's application to an open-source e-commerce platform written in PHP – osCommerce [2]. Our study effectively validates broadly scoped quality concerns (e.g., usability) against stakeholder requirements, systematically enables the tracing of aspects throughout the software life cycle, and empirically verifies improved system modularity claimed by aspect orientation.

The remainder of the chapter is structured as follows. Section 2 explains generally why stakeholder requirements contain early aspects, and Section 3 introduces one form of early aspects systematically discovered from goal-oriented requirements engineering. Section 4 shows some existing solution to the problem and their relation to us. Section 5 explains our approach in detail, by showing the syntax of our early aspect language and the PhpAspect language[3], explaining how to validate the traces between early aspects and code aspects using testing. Section 6 illustrates the application of the approach using the osCommerce case study extended from that of [9]. Reflecting on the lessons learnt, Section 7 concludes the paper and suggests future work.

---

[2] http://www.oscommerce.org/ Last accessed on November 14, 2009.

[3] http://phpaspect.org

2   Why Aspects in Requirements?

This section aims to address the research question: why do we need aspects in requirements? It situates our research within existing literature in Requirements Engineering (RE) and Aspect-Oriented Software Development (AOSD).

A requirement aspect is a stakeholder concern that cuts across other requirements concerns or artifacts [14]. It is broadly scoped in that it is found in and has an implicit or explicit impact on more than one requirements artifact [6].

Requirement aspects convey domain properties of interest to stakeholders, interact with base requirements dictate additional services, and need to be traced to stakeholder interests. For example, a performance requirement cuts across all functional requirements to deliver a "fast enough" computation. With respect to different functionalities, however, such performance requirement can be operationalized differently. For example, a "high-performance sort" function is required to have $O(n \log n)$ time complexity where the size of the problem, n, is large. On the other hand, a high-performance "enter keyword" function is not required to finish in milli-seconds: Most of elapsed time the computer is waiting for user input, thus more tolerable to computational complexity. Nonetheless, the overall system performance requirement depends on all functionality to be fast enough. Therefore we can regard the performance concern even a globally scoped aspect where all functional requirement artifacts are affected. Other requirements, such as security, usability, etc., that share the same characteristics are all requirement aspects. Once woven, requirement aspects may also become base requirements for other requirement aspects. For example, a usability requirement aspect can crosscut the previous example "high-performance sort", requiring a progress bar to assure end-users about how much progress the algorithm is moving forward. Base requirements include functional requirements as well as those woven with other requirement aspects, hence an inevitable problem is to validate them. The effectiveness of a requirement aspect can only be validated by considering base requirements that they are woven. For example, one may compare one sort algorithm with another sort algorithm in terms of measurable performance, rather than comparing one performance problem to do with sorting with another problem to do with entering password. It is therefore useful to instantiate the validation to the places where one can trace to the woven requirements.

Although in AOP (aspectJ-like), aspects are typically considered to be solutions that crosscut existing base solutions. In RE, such is not always the case. This is due to the fact that a base problem does not always exist before the crosscutting ones. Reflecting different viewpoints of a problem, early aspects can, in the extreme, crosscut each other symmetrically. In that sense, viewpoint merge/matching techniques in RE [15] can also be considered an early aspect weaving technique.

To interpret the notions involved in aspect orientation from an RE perspective, we follow the metaphor that every requirement aspect acts as a service provider to some base modules [16].

- **Advice** defines the content of the service that a specific requirement aspect provides. It describes *what* the service is about.
- **Join points** are points in the base which a requirement aspect interacts with. They describe *where* the service is provided.

Table 1. Conceptual relation between aspects and requirements

Approach	candidate aspects	advices	join points	pointcuts	weaving
[8,7,14,15]	concerns	requirements	viewpoints	contribution table	resolving conflicts
[9,10,11]	softgoals	advising tasks	goals and tasks	contribution links	composing algorithm
[19]	security	vulnerabilities	functional req.	threat descriptions	composition process
[6,17]	non-functional terms	occurrences	req. statements	indicator terms	information retrieval

- **Pointcut** represents a set of join points. It describes the *situational patterns* of the service that an aspect provides.
- **Weaving** is the process of coordinating service providers (requirement aspects) and consumers (base requirements). It describes *when* and *how* the service takes place.

In addition, the purpose of a requirement aspect describes *why* the service is needed in the first place. Since it is hard, and usually counterproductive [14], to carry out aspect-oriented RE activities on requirements documents that are ill-structured, existing approaches fall into two categories. [4]

The first category is based on linguistic clues where the scattered occurrences of terms in the document are considered as crosscutting concerns, and their relations to the terms in the context are considered as join points [6]. Information retrieval techniques [17] and natural language processing techniques [18] have been proposed to identify these scattered occurrences and associate them to the non-functional requirements (NFRs) as aspects.

The second category is based on structured requirements models, such as goal graphs [9] or problem frames [19]. The crosscutting concerns on these structures are often derived from the different nature of functional and non-functional domains. For example, a "higher performance" task in the "performance tuning" domain tends to cut across other functions in the system, leading to a natural crosscut to system functions. Therefore, the structures of the models used by the goal-oriented RE can be exploited to explain how early aspects are discovered from the stakeholders, rather than mined from the code.

To sum up, Table 1 classifies some approaches on early aspects with respect to the concepts of AOP.

In relation to the themes of this book, requirement aspects "interwine requirements with contexts" in the sense that requirements are allowed to be crosscutted by global concepts such as viewpoints, concerns, softgoals. Moreover, all these must be localised with business, organisational and community situations through contextual joinpoints. Requirement aspects also help "evolving designs and ecologies" because selectively freezing certain aspects while changing other aspect is realised easier by modularising the crosscutting concerns among the requirements that can significantly reduce the

---

[4] Please refer to the early aspects landscape reports (available at http://www.early-aspects.net) for more detailed surveys of aspect-oriented RE approaches.

co-changes of similar advices. Requirement aspects support "fluidity of designs, architecture, visualisation and representation" because a core architecture can be cleaned by separating messy crosscutting concerns, which can greatly help visualising and representing the base system. Last but not least, requirement aspects fit the theme of "complexity, business process focus, architecture" by hiding the complexity caused by tangled requirements. Although requirement aspects make it much easier to manage crosscutting concerns among requirements, common to these themes, there is still a research challenge to trace them into system implementations and validate them against stakeholder goals. In the remainder of the chapter, we will focus on demonstrating a possible answer to this challenge.

3   Discovering and Testing Aspects in Goals

This section introduces the terminology and concepts involved in our approach to discovering aspects in goal models and testing their validity.

Organizational goals lead to requirements. Goals justify and explain the presence of requirements, while providing the baseline for validating stakeholder concerns [13]. Goal modeling shifts the emphasis in requirements analysis to the actors within an organization, their goals, and the interdependencies between those goals, rather than focusing on processes and objects. This helps us understand *why* a new system is needed, and allows us to effectively link software solutions to business needs.
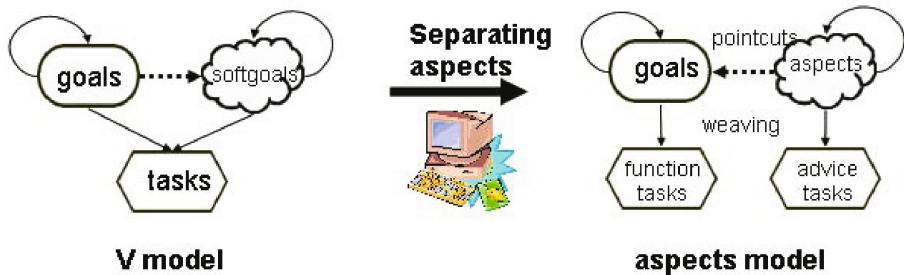
Goals provide basic information for detecting and resolving conflicts that arise from multiple viewpoints [20]. Goal analysis facilitates the discovery of trade-offs and the search of the full space of alternatives, rather than a subset. Goal modeling frameworks distinguish between *hard (functional) goals* – states that actors can attain – and *softgoals*, which can be satisfied only to certain degrees. System qualities, such as reliability, efficiency, and portability, are typically expressed as softgoals to suggest that intended software is expected to satisfy these NFRs within acceptable limits, rather than absolutely [21].

Aspects in goal models can be discovered using the correlations from hard goals to softgoals along with a goal eliciting and refinement process of a V-shape goal graph [9].

Figure 1 illustrates the process of separating early requirement aspects as a result of goal elicitation. A V-shape goal graph is a simplified metamodel where only functional and non-functional requirements are initially elicited as goals/softgoals respectively. Then through vertical refinements, they are operationalized into tasks. The horizontal contribution links between goals and softgoals provide a clue to separate tasks that operationalize softgoals from the tasks that operationalize hard goals.

The formal requirement aspect elicitation process can be briefly explained as follows. Initially, the stakeholders' high-level concerns are elicited as abstract goals. The functional ones are represented by hard goals and the non-functional ones are represented by softgoals. Relations are also elicited as abstract contribution (resp. correlation) links from the functional hard goals to the non-functional softgoals that must be fulfilled by the prescribed system-to-be.

During the refinement process, these abstract goals are recursively decomposed into more concrete ones through AND/OR decomposition rules. As a result, several

Fig. 1. Illustration the separation of requirement aspects [9]
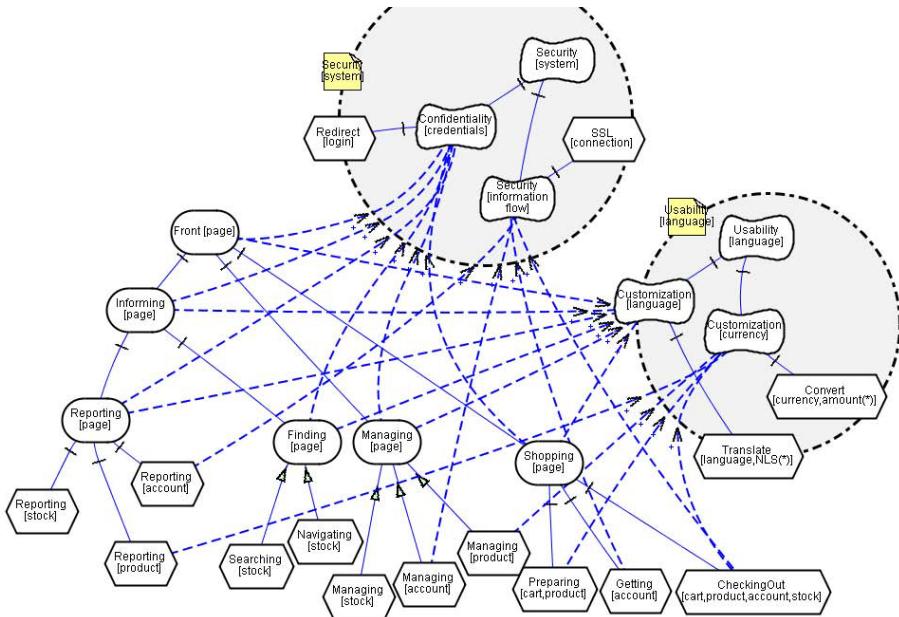
hierarchies of goal trees are derived. These are inter-related through abstract contribution (resp. correlation) links that represent semantic inter-dependencies between these goal trees.

At the end of the refinements, all abstract goals are decomposed into a set of goals that need no further decompositions. These leaf-level goals can be fulfilled by tasks that can be carried out by the system-to-be or external actors. The set of tasks are further categorized into functional ones and non-functional ones depending on whether they are at the bottom of the decomposition hierarchy of an abstract hard goal, or of an abstract softgoal. The refinement of abstract goals must be validated to maintain the abstract contribution links, which can often be fulfilled by weaving the concrete non-functional (operationalized) tasks into the functional tasks. As such, every OR-decomposed subgoal must fulfill the same commitment to the softgoals as their parent goal does. It is often the case, if not always, that non-functional tasks crosscut several functional ones that belong to different OR-decomposed subtrees.

Figure 2 illustrates early requirement aspects in the media shop study [23,9]. The top level softgoals such as "Security [system]" and "Usability [language]" are captured as goal aspects, which are operationalized into advising tasks. The aspect weaving is achieved by composing the advising tasks with the functional tasks of effected hard goals. As an example, the aspect "Customization [language]" is operationalized into an advising task "Translate [language, NLS]", meaning that the media shop is advised to translate occurrences of natural language strings (NLS) into the desired language. This advice crosscuts all hard goals that display Web pages and is intended to enhance system usability for native users of the desired language. Basic functionalities (e.g., "Informing", "Reporting" and "Shopping") defined by hard goals via functional tasks shall not be changed, though, by weaving such a usability aspect.

Aspect verification is an active area where research focuses on different ways to carry out tests, select test cases or generate test cases for a given AOP mechanism [24]. For example, unit testing aspects has been proposed by Lesiecki [25], equipped with eight patterns to verify the crosscutting behavior as well as the crosscutting specifications. The specifications correspond to the pointcuts in aspects, which are validated through aspectJ visualization and mock targets manually. The behavior is tested with a focus on the crosscutting functionality, i.e., advice.

In this work, we attribute the crosscutting behavior to both functional and non-functional requirements, and propose to reuse existing testing mechanisms to work out

Fig. 2. Illustration of early requirement aspects in media shop extended from i^* [22] notations. Aspects, together with their advising tasks, are represented as first-class modules in the upper-right corner of the goal model. The contribution links, from hard goals and functional tasks to requirement aspects – operationalized softgoals – modularize the crosscuts that would otherwise be tangled and scattered in the goal model.

a natural way to validate the implementation of the identified early aspects. Since requirement aspects address non-functional requirements, the functionality of the base components must not be modified by weaving the aspects. Thus, existing unit test cases that target at the original base components can be reused for testing the functional behavior of the aspects.

In our approach, it is crucial to validate whether the non-functional behavior of an implemented aspect fulfills its intended NFR defined by the early goal aspect. Certain qualities in a system with weaved aspects must outperform the one without aspects, so that the effort of managing aspects can be justified. Such examinations are guided by the quality metrics derived from requirement aspects, and are supported by modularizing testing aspects [26], or via other means depending on the type of NFRs under investigation.
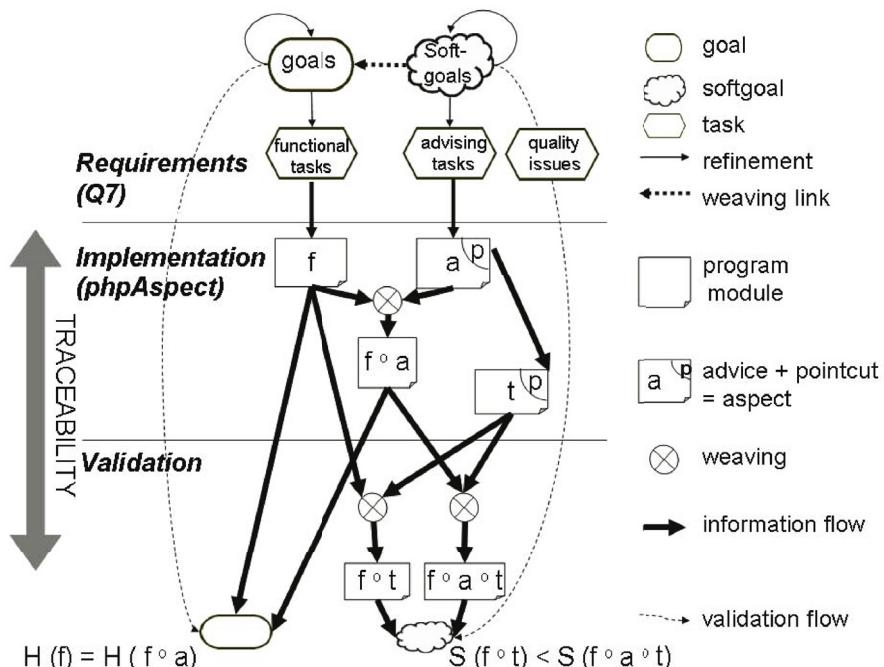
4   An Aspect Tracing and Validating Framework

In this section, we explain how tracing and validating requirement aspects are carried out throughout the software life cycle. Figure 3 overviews the process of our approach. The upper part of the figure highlights the early aspects discovery process discussed earlier. Advising tasks, which operationalize softgoals and relate to hard goals, are

modularized as aspects and weaved into goal models to enable aspect-oriented require-ments analysis.

Key concepts of AOP implementation are depicted in the middle of Figure 3. Func-tional modules (f) and code aspects (advice + pointcut) are derived from functional and advising tasks respectively. The weaved system ($f \circ a$) is obtained by composing advice (a) with bases according to the pointcut description (p). Some aspects identified at the requirements level may not be mapped to code at all. For example, a typical perfor-mance requirement might state that the system shall complete a task within 2 seconds. These early identified aspects play a key role in monitoring the system's behavior, and shall not be lost in software development. We record them as quality issues to estab-lish their traceability throughout the software life cycle, but the discussion of handling these quality issues is beyond the scope of this work. The success criteria for aspects are specified in t, which gathers quality metrics and shares the same pointcut with a. It is important to incorporate the metrics t so that one can measure system qualities with ($f \circ a \circ t$) and without ($f \circ t$) aspects. Note that our framework is viable for applying different programming languages and weaving mechanisms such as AspectC, aspectJ or HyperJ. We use phpAspect to illustrate our approach in the following discussion.

System validation is shown in the lower part of Figure 3. The weaved system ($f \circ a$) is subject to two kinds of tests on basis of respectively hard or soft goals of stakeholders. The first test ensures that systems with and without aspects have the same functionality defined by hard goals: $H(f) = H(f \circ a)$. Existing testing mechanisms, such as unit

Fig. 3. A process overview of tracing and validating aspects

testing, can be reused to validate whether the weaved system satisfies the functional requirements. The second test checks whether the weaved system indeed improves system qualities in terms of the degree of softgoal satisfaction: $S(f \circ t) < S(f \circ a \circ t)$. Our approach enables both forward and backward tracing of crosscutting concerns throughout the software life cycle.

4.1   Goal Aspects in Q7

Q7, or 5W2H – **why**, **who**, **what**, **when**, **where**, **how**, **how much** – is a pseudo programming language that captures the structure of requirements goal graphs, including the major entities of the NFR framework [21]. The syntax of the language is designed to facilitate the reuse of solutions in the non-functional domains while incorporating aspect orientation [27]. The seven questions are usually asked for eliciting and elaborating goal-oriented requirements following the Tropos methodology [23].

The answers to the **why** and **how** questions respectively indicate the composition and decomposition relations between abstraction and implementation. Adapted from the goal model of Figure 2, the following example shows the AND/OR decomposition relations among the hard goals in the media shop domain. The front page of the shop has the functionality for "informing" the end-users and administrators. This goal is decomposed into "finding" *and* (&) "reporting" relevant information. In order to find information, a user is allowed to "search" *or* (|) "navigate" the shop. The nesting structure of curly braces helps visualize the decomposition hierarchy of the goals.

```
Informing { &
  Finding { |
    Searching
    Navigating
  }
  Reporting
    ...
}
```

The answers to the **how much** question show the degree of contributions between hard goals and softgoals. Q7 uses the labels "++", "+", "−", and "−−" to indicate the "make", "help", "hurt", and "break" relations between the goals. The answers to the **what** question connect the goal to its subject matter [28]. In Q7, such information is placed inside square brackets as topics of the goals or softgoals. For example, when the system meets the shop's "Front [page]" goal, it also *makes* (++) major top-level softgoals ("⇒"), such as "Security [system]" and "Usability [language]".

```
Front [page] {
  ...
} => ++ Security [system],
     ++ Usability [language] ...
```

The answers to the **when** question indicate the feasibility of the goals under certain contexts [29], and those to the **who** question attribute a goal to an encapsulating module. In the i^* terminology [22], such a module is called an actor that either processes or delegates a goal or a task to other actors via strategic dependencies. In Q7, we use the idea of "namespaces" to represent the actor names. For example,

```
<MediaShop>::Front [page] { &
   Managing [page]
   ...
}
```

Here, "MediaShop" is the actor that processes the goal "Front [page]". If the actor is not explicitly specified, a goal inherits the namespace from its parent goal. Thus, "Managing [page]" belongs to the same "MediaShop" actor.

As an extension of the encapsulating actors, we create a new namespace to modularize the aspect that cuts across multiple entities in the goal model. As an example, the security aspect in Figure 2 is represented as follows.

```
<aspect>::Security [system] { &
  Confidentiality [credentials] <=+ [page] { &
    Redirect [login]
  }
  Security [information flow] <=+ [account] { &
    SSL [connection]
  }
}
```

The goal hierarchy within the aspect module is an *advice* and the leaf-level tasks in the hierarchy are called *advising tasks*. These tasks do not exist by themselves, since they have to be weaved into the functional goals by indicating where to attach the advice. The answers to the **where** question are designed to express the pointcut of an aspect, indicating which functional goals are suitable for applying the advice. For example, the following Q7 statements show a pointcut expression after the "⇐" symbol: + * [page], which matches the hard goals of any name (indicated by the wildcard *), of the subject matter Web "page", and those helping (+) achieve the usability softgoal. The advising task translates the "natural language string" (NLS) appeared in the Web page into the desired language (e.g., Spanish or German). Note that a pointcut can also be specified by enumerating the effected hard goals.

```
<aspect>::Usability [language] { &
  Customization [language] <= + * [page] { &
    Translate [language, NLS]
  }
}
```

All matched goals are therefore the *join points* of the aspect. A weaving algorithm [27] has been implemented in the OpenOME modeling tool (http://www.cs.toronto.edu/km/openome) to identify the join points and attach the advising tasks as siblings to the join point tasks. Both join point tasks and advising tasks then share the same parent, which is called the *weaved goal*. The weaving algorithm implemented in Q7 makes it possible to analyze the weaved goal model through a goal analysis tool, e.g., a goal reasoning algorithm [30].

As we can see from the examples presented above, Q7 provides a quality-based reuse mechanism for representing and modularizing crosscutting concerns in goal models. The Q7 language is not only capable of handling the characteristics of the quality knowledge, but also capable of relating those with functional descriptions. In addition, the textual form of Q7 greatly facilitates the tracing of stakeholder concerns throughout the software life cycle, as we shall demonstrate via a case study in Section 5.

4.2   Implementation in phpAspect

The early candidate aspects discovered in goal models are usually suited to be implemented as code aspects, but developers may choose other means to address these crosscutting concerns, as previously stated. Nevertheless, our approach explores the possibility to equip developers with a full-fledged aspect-oriented framework so that a clear separation of concerns is promoted throughout software development.

As one can see from Figure 3, functional and advising tasks from requirements are mapped into functional and aspectual modules in design and implementation, respectively. Since the subject in our case study – osCommerce – is implemented in PHP, we select a solution for AOP in this language, phpAspect [5], to facilitate the discussion in implementing early aspects.

PhpAspect is designed as an extension to the PHP language. It adds new constructs, such as: aspects, pointcuts, advices, and inter-types declarations, inspired by aspectJ for expressing aspects relating to objects and classes, while embracing specific features for Web-based applications. It provides pointcut expressions for constructions used in these applications, such as function call and execution, Web-based variable access, XML/HTML enclosing context identification, and the like. Moreover, phpAspect is able to weave aspect components in portions of code that are embedded into XML or HTML elements.

PhpAspect uses a static weaving process that performs source code transformation of a PHP program with aspect extensions into a standard PHP program. Both the source PHP program and the aspect modules are XMLized into abstract syntax trees, which can be weaved through customized XSLT stylesheets. The woven XML syntax trees, as a result, are transformed into the target PHP program via unparsing XSLT stylesheets

The following code shows an example of the security aspect for a Web application. This aspect first introduces a credential checking around all Web pages that require access authentication (captured with the `checkCredentials` pointcut on `goto` method call). This checking prevents users from accessing a Web page if they are not logged in or do not have the right credentials. In these cases, users are redirected to a more appropriate page, either the login or index page. Secondly, the security aspect checks that all cart operations performed by the client are done in an HTTPS (SSL) mode and deny them otherwise.

```php
<?php
aspect Security {
  //Intercept all instantiations of a page
  pointcut checkCredentials:call(Page->goTo($arg2));

  //Intercept all method execution of the cart
  pointcut checkSSL:exec(Cart->*(*));

  //Advice: Around all page instantiations, check the credentials
  around(User $user) checkCredentials {
    if($user->hasCredentials($_GET['page'],
                             $_GET['action'])) {
        proceed();
    } elseif (!$user->isLoggedIn()) {
        $thisJoinPoint->getObject()->goTo('login.php');
```

---

[5] Developed by William Candillon during the Google Summer of Code, see http://code.google.com/soc/php/about.html .

```
    } else {
        $thisJoinPoint->getObject()->goTo('index.php');
    }
}

//Advice: Around all method execution of the Cart,
//         We check whether the connection is SSL
around checkSSL {
    if(!$_SERVER['https']) {
        header("Location: https://{$_SERVER['HTTP_HOST']}
                        {$_SERVER['REQUEST_URI']}");
    } else {
        proceed();
    }
  }
}
?>
```

The above example not only demonstrates phpAspect's competence in working out the implementation of requirement aspects in question, but also shows its capacity to map and trace early identified crosscutting concerns in the code base.

4.3   Aspect Validation

It is crucial to validate the implementation against stakeholder requirements. We propose a goal-based testing approach to ensure that system functionalities are preserved and system qualities are enhanced by weaving aspects into base modules. This concept is highlighted by the validation flows in Figure 3.

In goal-oriented requirements engineering, when it is concrete enough to express the function of a task in terms of input and the expected output, a unit test case can be created to check whether the function is violated by comparing the output of the implemented function with the expected output of the required function. Therefore, the leaf-level functional task in the goal model corresponds to a set of unit test cases that tells whether the base program delivers the required functionality. Having enough unit test cases in terms of the coverage of the input domain, the functional task can be labeled "validated".

Aspects discovered in goal models provide a baseline for code aspects validation. If an advising task cuts across multiple functional tasks, the unit test cases of the functional tasks at the join points can be reused to test the functionality of the weaved system. This is because requirement aspects must not change basic functionalities defined by hard goals and functional tasks. The implementation of aspects, therefore, has to preserve this property.

On the other hand, the degree of certain softgoal satisfaction must be enhanced by the weaved system. Measuring quality attributes typically presents an obstacle to traditional testing mechanisms, since NFRs are not always easy to metricize. Our effort of modeling aspects early in the requirements pays off here. The results from goal-oriented analysis, including the quality metrics, the advising task and pointcut of requirement aspects, can be reused and extended to test softgoal satisfaction.

For example, the media shop keeps users from accessing a Web page if they are not logged in or do not have the right credentials. We model this requirement as a security aspect, and map it to a code aspect in phpAspect, as explained in Section 4.2. We can define a set of unit test cases that act as unauthorized agents and try to break

Table 2. Tracing the security ($\mathbb{S}$) and usability ($\mathbb{U}$) aspects in an osCommerce media shop

Concept	Q7	phpAspect
aspect ($\mathbb{S}$)	`<aspect>::Security [system]`	`aspect Security`
pointcut ($\mathbb{S}$)	`<= + * [page]`	`call(Page->goTo($arg2))`
	`<= + * [cart]`	`exec(Cart->*(*))`
advice ($\mathbb{S}$)	`{ & Redirect [login] }`	`checkCredentials{...}`
	`{ & SSL [connection] }`	`checkSSL{...}`
aspect ($\mathbb{U}$)	`<aspect>::Usability [language]`	`aspect Usability_Language`
pointcut ($\mathbb{U}$)	`<= + * [page]`	`call(Page->*printf(*))`
	`<= + * [date]`	`call(Data->strftime($arg2))`
	`<= + * [amount]`	`exec(Amount->display($arg2))`
advice ($\mathbb{U}$)	`{ & Translate [language, NLS] }`	`translatePage{...}`
	`{ & Display [format, date] }`	`dateTimeFormat{...}`
	`{ & Convert [currency, amount] }`	`convertCurrency{...}`

into the system. The expected output would be redirecting these malicious visits to the login or index page. Since these security-related test cases crosscut the ones devoted to testing system functionalities (e.g., shopping and searching), they can be regarded as *unit testing aspects* [25], thereby reusing the security aspect's pointcut description to perform the test case weaving.

It is worth pointing out that validating requirement aspects can be carried out by other means than defining unit testing aspects. For example, typical Web layer components do not lend themselves to unit testing, unless proper frameworks such as HttpUnit or PHPUnit are employed. In order to ensure that shopping is done securely, testing scripts can be developed to automatically verify that all cart operations are performed in an HTTPS (SSL) mode.

5   An Extended Case Study

We used an exploratory case study [31] as the basis for our empirical evaluation. The research question focused on how to leverage our approach in a real-world setting. Specifically, we derived the following hypotheses to guide the study design: (1) Modularizing and tracing broadly-scoped non-functional concerns throughout the software life cycle are enabled by our framework, and (2) Goal-based validation justifies why certain code aspects appear in the implementation.

The single case in our study is osCommerce, an open-source platform written in PHP, on which a Web-based media shop [23] development can be fully based. In our previous work [9], we used osCommerce to show how to discover aspects from media shop goal models. In particular, 7 requirement aspects were identified in [9], among which we choose security and usability aspects as 2 embedded units of analysis within the current case study. Such a selection is guided by the previous work in a familiar e-commerce domain, and represents a typical case and units of analysis since both security and usability are commonly discussed early aspects in the literature.

The data collection in our study consisted of three parts. First, the goal aspects of media shop were presented in [9] and further represented in Q7. Second, the

implementation of osCommerce in PHP was accessible through open-source reposi-
tories. Our implementation of osCommerce's code aspects in phpAspect was available
at [32]. Third, the goal-based validation instrumentation was developed and gathered
by the authors of this work (also available at [32]).

The analysis results of aspects tracing are presented in Table 2, from which the map-
pings between requirement aspects in Q7 and code aspects in phpAspect can be readily
spotted. Specifically, a one-to-one correspondence exists between the name of a require-
ment aspect and that of a code aspect. Moreover, we map goal's topics into parameter-
ized pointcuts, and map softgoal's operationalizations into advices.

We focus on the usability aspect in this section, as security is discussed in the pre-
vious section as an illustration of our approach. The requirement aspect "Usability
[language]" is AND-decomposed into 3 parts. One translates natural language strings
(NLS) appearing in a Web page to the local language. Another deals with displaying
date and time in the desired conventional format. The third converts money amounts
from a country's currency into the local currency. The Q7 representations for each point-
cut and advice of the usability aspect ($\mathbb{U}$) are given in the second column of Table 2.
Correspondingly, Table 2's third column highlights these concepts' counterparts in the
phpAspect implementation.

The implemented aspects were weaved into osCommerce's base modules by the ph-
pAspect weaver. We tested the weaved system in two respects: hard goal preservation
and softgoal enhancement, as indicated by the validation flows in Figure 3.

Unit test cases existed for validating the functional requirements of the osCommerce
system. Such test cases should not be affected by introducing the aspects that imple-
mented the NFRs. Therefore, we *reused* the functional testing units without any change
for checking the functionalities of the weaved system. For example, the shopping cart
sum computation must be the same regardless of which natural language being used by
the media shop customer. A unit test case using PHPUnit [6] was reused.
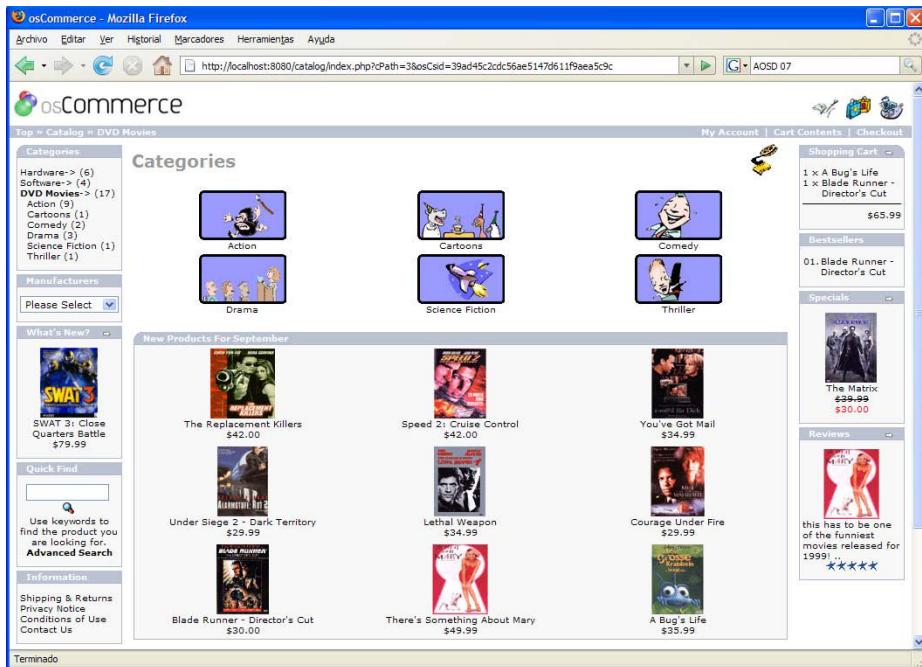
```php
require_once 'PHPUnit/Framework/TestCase.php';
require_once 'classes/cart.class.php';
class CheckoutTest extends
        PHPUnit_Framework_TestCase {
  private function getOrder(){
    $cart = new Cart();
    $cart->addItem('Bread', 2);
     // 2.20 each in USD
    $cart->addItem('Butter', 1);
     // 3.20 each in USD
    return $cart->getAmount();
  }
  public function testCheckoutTotal(){
    $this->assertEquals(Currency::convert(
     2*2.20+1*3.20, 'usd'), $this->getOrder());
  }
}
```

We reused 22 functional unit test cases for the weaved system to make sure that
introducing requirement aspects does not change the function of osCommerce. If one
introduces an aspect that does change the functionality of the original system, we con-
sider either the function is not intended originally, or new test case needs to be designed

---

[6] http://phpunit.sourceforge.net/ Last accessed on November 14, 2009.

Fig. 4. Screen shot of an osCommerce media shop shown in default language (English)

and weaved into the original set of test cases along with the code aspect. However, it is beyond our scope to discuss how an aspect should implement a functional requirement, and how such an aspect should be traced and validated.

Having checked that the weaved system preserved system functionalities, we wanted to test whether the aspects indeed addressed the quality concerns, and more importantly, whether they helped better achieve the original stakeholder softgoals. Such a validation was guided by the quality metrics derived from goal-oriented analysis. Take "Usability [language]" for example, osCommerce currently supported English, German, and Spanish users. Figure 4 shows a Web page in the default language – English. The usability aspect should render a Web page by using the language chosen by the user as natural as possible. This included showing textual strings, date, and currency in the desired language and format, as described earlier and indicated in Table 2. Figure 5 shows a screen shot of the weaved system after the language customization aspect is applied.

We validated the usability aspect via two means. A Spanish tester confirmed that the language customization aspect worked very well, in that most Web page contents shown in the desired language, including date and currency, were semantically correct. To evaluate this result in a triangulating fashion [31], we also chose the pspell testing harness [7] to check the syntax of the resulting Web page texts automatically. The fact that all customized pages contained less than 5% syntactic errors increased our confidence that the aspects' weaved system indeed helped better meet stakeholders' usability requirement.

---

[7] http://php.net/manual/en/ref.pspell.php Last accessed on November 14, 2009.
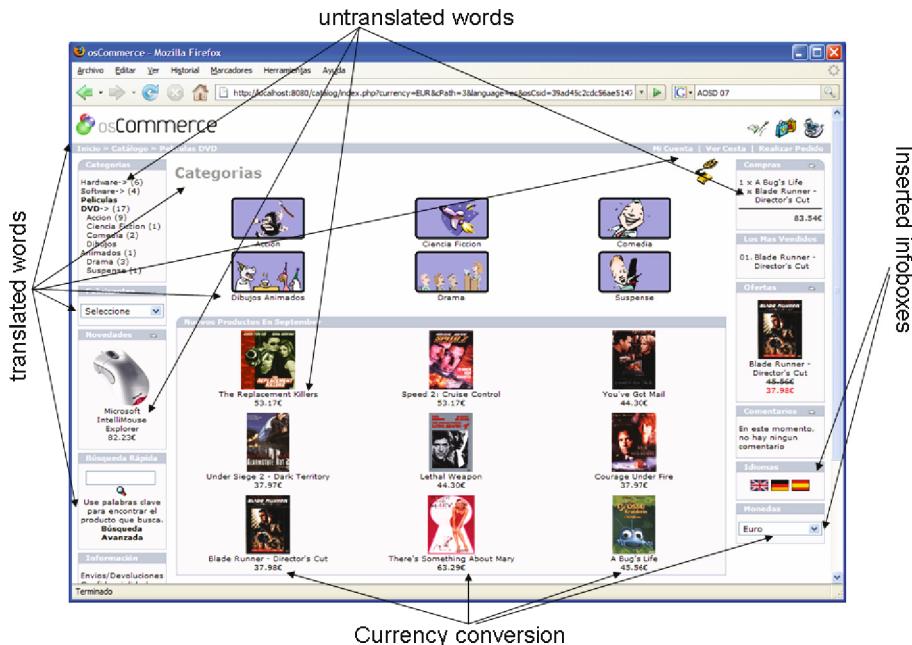
Fig. 5. Screen shot of the weaved system that enhances usability for Spanish users

The analysis results of the 2 embedded units – security and usability – within the case study presented positive empirical evidence for accepting our two initial hypotheses. However, we did observe that existing goal models might not be complete to justify *all* code aspects. On the contrary, certain aspects in the implementation could help justify stakeholder goals and uncover the missing parts in the requirements.

When reengineering osCommerce using aspects, developers wanted to achieve a high degree of maintainability to facilitate modification and reuse. Aspects modularized tangled and scattered code, which led to a cleaner code base. For instance, in the original implementation, 603 natural language string variables were defined in each of the English, German, and Spanish language header files to be included in specific Web pages. This caused scattered code duplication. We defined a single usability aspect to modularize these language customization concerns, and removed 3,990 lines of code, 7.6% from the whole code base. This helped address the maintainability softgoal from the developer's perspective, which uncovered a missing part of media shop goal models presented in [9]. In this sense, one shall not apply our aspect tracing and validating framework in a strict forward-engineering way, but in an iterative fashion within an integrated software development process.

Several factors can affect the validity of our exploratory case study: construct validity, external validity, and reliability [31]. The key construct is the idea of a requirement aspect. Although softgoals have huge potential to become early aspects [33], others may argue that requirement aspects can be functional as well. We believe that requirement aspects are intended to enhance system qualities while preserving functionalities, and the early aspects community needs to make it more carefully. In regard to external

validity, we chose Q7, phpAspect, and various testing mechanisms in tracing and validating requirement aspects for a common e-commerce application. Further empirical studies are needed to examine the applicability and generality of our framework in coping with other modeling notations, programming languages, and application domains. To reduce the threats to reliability, we selected an open-source project and made all our empirical data publicly accessible [32]. Thus, our reported study is replicable, and we believe we would obtain similar results if we repeated the study.

6   Related Work

Grundy *et al* [34,35] and Rashid *et al* [8] are among the earliest authors who recognized the advantage of aspects for reasoning about component-based software at the requirements level. Araujo *et al* [36] compose aspectual and non-aspectual early design models (e.g. Scenarios). They use sequence diagrams and executable state machines to weave aspectual scenarios represented by interaction pattern specifications. When this state machine-based approach scales up, it is possible to establish further traceability between the statecharts-based models with implementations [37,38].

Haley *et al* [19] proposed an early aspect approach focusing on deriving security requirements from crosscutting threat descriptions. The security aspect in our case study can be considered as an operationalization of crosscutting anti-threat advices. In our validation framework, security is treated as an NFR that cannot be sacrificed.

In comparing the related work listed in a literature survey of early aspects [16], our key contribution brings early aspects traceability links and validation together. In [18], traceability links between early aspects and designs are set up by comparing naming conventions, term co-occurrences, etc. It is not clear how traceability through such queries can be verified, as the leap from design to implementation may distort the precision. For example, a design element with a similar name as the requirements may be implemented differently from what is specified. In our work, we explicitly rely on the decomposition and contribution links in the goal models to build traceability among goals and aspects. Therefore, our goal-based testing for aspects helps to show how well requirements are carried out by the advices.

The work of Cleland-Huang *et al.* [17] extracts NFRs based on information retrieval techniques. The strength of traceability is quantified as precision and recall of the keyword-based search. When naming conventions mismatches the functionality specified in the program, our goal-based validation of traceability may improve their precision through the semantics of executed test cases.

Reflecting different viewpoints of the problem, early aspects can, in the extreme, crosscut each other symmetrically. In that sense, viewpoint merge/matching techniques in requirements engineering [15] can also be considered an early aspect weaving technique. A main technical difference is that model merging requires explicit representation of joinpoints between multiple viewpoints, while in early aspects such joinpoints are typically expressed implicitly by pointcuts expressions.

In [39], proof obligations were introduced to formalize the validation to the requirement aspects. Their approach can be applied to programs of well-defined axiomatic semantics. For the quality attributes that do not have a clear-cut answer to satisfaction,

it is necessary to validate whether and how much the system can be improved after weaving the proposed aspects. For example, instead of proving that a word is Spanish, we show how well it is understandable by the Spanish-speaking users. Although we reuse unit testing for functional requirements, we believe a complementary approach based on generating proof obligations can better guide the validation of functional requirements.

In [40], Neil Maiden has called for *quantifiable requirements*. Frameworks, such as NFR [21], need to have a metricized way to verify that requirements are addressed properly using operationalizations. Most quantifications of NFR help selection of alternatives [41] when the satisfaction of the metricized NFR can be visualized [42,43]. Our proposed goal-based testing framework can be regressively applied to join points of the softgoal operationalizations, thus more precisely measuring the satisfaction of the NFRs.

7   Conclusions and Further Work

We have proposed an aspect-oriented framework to investigate how aspects discovered in requirements can be validated and traced throughout the software life cycle. We evaluated the approach via an exploratory case study that reengineered a public domain e-commerce platform. The study presented our observations of and insights into the problems of tracing and validating requirement aspects. Even though the study collected positive evidence on the applicability and usefulness of our approach, more in-depth empirical studies are needed to lend strength to the preliminary findings reported here. In addition to confirming our hypotheses, we verified the initial AOP claim that it is natural to implement globally concerned NFRs as aspects that cut across subsystems [1]. From the results obtained, we believe that aspect orientation is a promising solution to tackling crosscutting concerns early on, and that our framework systematically handles traceability and validation issues for requirement aspects.

In future work, we will continue to explore concepts and mechanisms for tracing early aspects throughout the software lifecycle and validating them in a given design. The case study suggests that some candidate early aspects are easier to turn into coding aspects, and some softgoals are easier to measure through testing aspects. Automatic tracing and validation of early aspects may require in-depth knowledge of the application domain and quality requirements.

We will also investigate how to trace other forms of early aspects into validateable design. The composition of problem and solution structures may share common validation concerns in problem frames [19], it is interesting to know whether problem frame concerns lead to coding aspects and whether the proof-obligations of trust assumptions can be validated using testing aspects.

Along another line of research, we have found aspects useful in instrumenting the monitors into the code such that one can dynamically diagnose problems in requirements satisfaction [44]. Such uses of aspects help to satisfy quality requirements of the software development process, rather than those for the software products. These can be generally regarded as software maintenance aspects, which may indirectly help improve the software product quality. Relating process-oriented early aspects to product-oriented

ones may give rise to a better traceability on how software process helps produce high-quality software products.

References

1. Kiczales, G., Lamping, J., Menhdhekar, A., Maeda, C., Lopes, C., Loingtier, J.M., Irwin, J.: Aspect-oriented programming. In: Aksit, M., Matsuoka, S. (eds.) ECOOP 1997. LNCS, vol. 1241, pp. 220–242. Springer, Heidelberg (1997)
2. Kiczales, G., Mezini, M.: Aspect-oriented programming and modular reasoning. In: Intl. Conf. on Software Engineering, pp. 49–58 (2005)
3. Hannemann, J., Kiczales, G.: Overcoming the prevalent decomposition of legacy code. In: Wkshp on Advanced Separation of Concerns at ICSE (2001)
4. van Deursen, A., Marin, M., Moonen, L.: Aspect mining and refactoring. In: Wkshp on Refactoring: Achievements, Challenges, Effects at WCRE (2003)
5. Seybold, C., Glinz, M., Meier, S., Merlo-Schett, N.: An effective layout adaptation technique for a graphical modeling tool. In: Intl. Conf. on Software Engineering, pp. 826–827 (2003)
6. Baniassad, E., Clements, P.C., Araújo, J., Moreira, A., Rashid, A.: Discovering early aspects. IEEE Software 23(1), 61–70 (2006)
7. Rashid, A., Moreira, A., Araújo, J.: Modularisation and composition of aspectual requirements. In: Intl. Conf. on Aspect-Oriented Software Development, pp. 11–20 (2003)
8. Rashid, A., Sawyer, P., Moreira, A., Araújo, J.: Early aspects: A model for aspect-oriented requirements engineering. In: Intl. Requirements Engineering Conf., pp. 199–202 (2002)
9. Yu, Y., do Prado Leite, J.C.S., Mylopoulos, J.: From goals to aspects: Discovering aspects from requirements goal models. In: RE, pp. 38–47 (2004)
10. Zhang, C., Jacobsen, H.A., Yu, Y.: Linking goals to aspects. In: Early Aspects Wkshp at AOSD (2005)
11. Yu, Y., Niu, N., González-Baixauli, B., Candillon, W., Mylopoulos, J., Easterbrook, S., Leite, J.C.S.d.P., Vanwormhoudt, G.: Tracing and validating goal aspects. In: Proc. of the 15th IEEE International Requirements Engineering Conference (RE 2007), 15-19 October 2007, pp. 53–56 (2007)
12. Niu, N., Yu, Y., González-Baixauli, B., Ernst, N., Leite, J.C.S.d.P., Mylopoulos, J.: Aspects across software life cycle: A goal-driven approach. Trans. on Aspect-Oriented Software Development IV (to appear, 2008)
13. van Lamsweerde, A.: Goal-oriented requirements engineering: A guided tour. In: Intl. Symp. on Requirements Engineering, pp. 249–262 (2001)
14. Nuseibeh, B.: Crosscutting requirements. In: Intl. Conf. on Aspect-Oriented Software Development, pp. 3–4 (2004)
15. Sabetzadeh, M., Easterbrook, S.M.: View merging in the presence of incompleteness and inconsistency. Requir. Eng. 11(3), 174–193 (2006)
16. Niu, N., Easterbrook, S., Yu, Y.: A taxonomy of requirements aspects. In: Early Aspects Wkshp at AOSD (2007)
17. Cleland-Huang, J., Settimi, R., Zou, X., Solc, P.: The detection and classification of non-functional requirements with application to early aspects. In: Intl. Requirements Engineering Conf., pp. 39–48 (2006)

18. Clarke, S., Baniassad, E.: Aspect-Oriented Analysis and Design: The Theme Approach. Addison-Wesley, Reading (2005)
19. Haley, C.B., Laney, R.C., Nuseibeh, B.: Deriving security requirements from crosscutting threat descriptions. In: Intl. Conf. on Aspect-Oriented Software Development, pp. 112–121 (2004)
20. Dardenne, A., van Lamsweerde, A., Fickas, S.: Goal-directed requirements acquisition. Science of Computer Programming 20(1-2), 3–50 (1993)
21. Chung, L., Nixon, B., Yu, E., Mylopoulos, J.: Non-Functional Requirements in Software Engineering. Kluwer Academic Publishers, Dordrecht (2000)
22. Yu, E.: Towards modeling and reasoning support for early-phase requirements engineering. In: Intl. Symp. on Requirements Engineering, pp. 226–235 (1997)
23. Castro, J., Kolp, M., Mylopoulos, J.: Towards requirements-driven information systems engineering: The Tropos project. Information Systems 27(6), 365–389 (2002)
24. Xie, T., Zhao, J.: A framework and tool supports for generating test inputs of AspectJ programs. In: Intl. Conf. on Aspect-Oriented Software Development, pp. 190–201 (2006)
25. Lesiecki, N.: Unit test your aspects – eight new patterns for verifying crosscutting behavior. IBM DeveloperWorks (2005)
26. Sokenou, D., Herrmann, S.: Aspects for testing aspects? In: Wkshp on Testing Aspect-Oriented Programs at AOSD (2005)
27. Leite, J., Yu, Y., Liu, L., Yu, E., Mylopoulos, J.: Quality-based software reuse. In: Intl. Conf. on Advanced Information Systems Engineering, pp. 535–550 (2005)
28. Zave, P., Jackson, M.: Four dark corners of requirements engineering. Trans. of Software Engineering and Methodology 6(1), 1–30 (1997)
29. Salifu, M., Yu, Y., Nuseibeh, B.: Specifying monitoring and switching problems in context. In: Proc. of the 15th IEEE International Requirements Engineering Conference (RE 2007), pp. 211–220 (2007)
30. Giorgini, P., Mylopoulos, J., Nicchiarelli, E., Sebastiani, R.: Reasoning with goal models. In: Spaccapietra, S., March, S.T., Kambayashi, Y. (eds.) ER 2002. LNCS, vol. 2503, pp. 167–181. Springer, Heidelberg (2002)
31. Yin, R.: Case Study Research: Design and Methods. Sage Publications, Thousand Oaks (2003)
32. Yu, Y., et al.: osCommerce's phpAspect report (Last accessed on August 15, 2008), http://www.cs.toronto.edu/~yijun/aspectPHP
33. Niu, N., Easterbrook, S.: Analysis of early aspects in requirements goal models: A concept-driven approach. Trans. on Aspect-Oriented Software Development III, 40–72 (2007)
34. Grundy, J.C.: Aspect-oriented requirements engineering for component-based software systems. In: Intl. Symp. on Requirements Engineering, pp. 84–91 (1999)
35. Grundy, J.: Multi-perspective specification, design and implementation of software components using aspects. Intl. Journal of Software Engineering and Knowledge Engineering 10(6), 713–734 (2000)
36. Araújo, J., Whittle, J., Kim, D.K.: Modeling and composing scenario-based requirements with aspects. In: Intl. Requirements Engineering Conf., pp. 58–67 (2004)
37. Yu, Y., Wang, Y., Mylopoulos, J., Liaskos, S., Lapouchnian, A., do Prado Leite, J.C.S.: Reverse engineering goal models from legacy code. In: Intl. Requirements Engineering Conf., pp. 363–372 (2005)
38. Whittle, J., Jayaraman, P.K.: Generating hierarchical state machines from use case charts. In: Intl. Requirements Engineering Conf., pp. 16–25 (2005)
39. Katz, S., Rashid, A.: From aspectual requirements to proof obligations for aspect-oriented systems. In: Intl. Requirements Engineering Conf., pp. 48–57 (2004)
40. Maiden, N.: Improve your requirements: Quantify them. IEEE Software 23(6), 68–69 (2006)

41. Mylopoulos, J., Chung, L., Liao, S., Wang, H., Yu, E.: Exploring alternatives during requirements analysis. Software 18(1), 92–96 (2001)
42. González-Baixauli, B., do Prado Leite, J.C.S., Mylopoulos, J.: Visual variability analysis for goal models. In: Intl. Requirements Engineering Conf., pp. 198–207 (2004)
43. Ernst, N.A., Yu, Y., Mylopoulos, J.: Visualizing non-functional requirements. In: Wkshp on Requirements Engineering Visualization at RE (2006)
44. Wang, Y., McIlraith, S.A., Yu, Y., Mylopoulos, J.: An automated approach to monitoring and diagnosing requirements. In: Proc. of the 22nd ACM/IEEE International Conference on Automated Software Engineering (ASE 2007), pp. 293–302 (2007)

Section 5: Adapting Requirements Practices in Different Domains

William Robinson

Technology has a tremendous impact on society. In recent years, the Internet, World Wide Web, and Web 2.0 has changed the nature of commerce, government, and of course software development. It affects the practices of producing requirements and as well as the kinds of systems to be designed. The effect of converging technologies on the role of requirements engineering is considered in the first article by Matthias Jarke, while the effect of technology on requirements practices is considered in the second article by Walt Scacchi. Together, they provide theoretical and practical perspective on requirements engineering issues faced in a modern, technology driven world.

The first article by Matthias Jarke titled "On Technology Convergence and Platforms: Requirements Challenges from New Technologies and System Architectures" presents opportunities and challenges for requirements engineering resulting from major technological changes. Technology convergence is a central issue for Requirements Engineering. Formerly, specialized Requirements Engineering techniques and localized policies could be applied to technology islands. Now, technologies have converged to create "the system"—from the users view—that combines computing, media, internet, into a pervasive anytime, anywhere environment. Consequently, formally isolated systems and policies are now integrated. Cultures clash as heterogeneous technologies and their stakeholders interoperate using "the system". Requirements Engineering may address fundamental problems of technology convergence: (1) Requirements Engineering can serve as a *lingua franca* across disciplines, (2) Requirements Engineering can guide the specification of social systems, striking a balance between autonomy, flexibility, and governance, and (3) Requirements Engineering can exploit emerging software architecture standards, which constrained and drive new systems. Two examples illustrate how Requirements Engineering is uniquely positioned to address these fundamental problems.

The second article by Walt Scacchi titled "Understanding Requirements for Open Source Software" describes a study of the roles, forms, and consequences arising in requirements within open source software development. Many open source projects are successful in their exponential growth and high quality. Requirements practices may account for open source project successes, which compare favorably with traditional development. Open source software requirements practices include: (1) continuous requirements evolution, (2) community development and participation, (3) developers as end-users, and (4) decentralized requirements. Web technology enables these practices, which engender software "informalisms," such as online forums, that capture requirements in a variety of forms. Examples are drawn from projects in the domains of networked computer games, Internet/Web infrastructure, bioinformatics, higher education computing, and military computing.

Together, these articles show how technology affects the theoretical perspective and practical application of requirements engineering for software intensive systems. Jarke considers the technology-driven, multidisciplinary nature of new computing systems, while Scacchi considers the successful practices of open source software development, enabled by and applied to new technologies. Together, they imply important considerations for future Requirements Engineering: (1) awareness of architectures and standards, (2) systems flexibility via self-monitoring and evolution, (3) convergence of multiple disciplines, and (4) non-traditional informalisms of decentralized, community-driven development. These issues provide opportunities for technical and empirical requirements engineering research.

On Technology Convergence and Platforms: Requirements Challenges from New Technologies and System Architectures

Matthias Jarke

Information Systems, RWTH Aachen University & Fraunhofer FIT
Ahornstr. 55, 52074 Aachen, Germany
`jarke@cs.rwth-aachen.de`

Abstract. In this chapter, we investigate some opportunities and challenges for requirements engineering resulting from major changes in the technical context in which ICT systems operate, in particular from the continuous trend towards information and communication technology convergence. We illustrate these challenges with two major examples, one concerning requirements monitoring as a self-governance mechanism in Internet-based social networks, the other concerning the role of requirements modeling as a mediator between different cultures in embedded systems engineering for the automotive industry. Starting from a brief re-iteration of Thomas Friedman's argument on standards evolution, we finally discuss platform strategies as an important emerging challenge for organizational RE.

1 Introduction

It is widely recognized that requirements engineering no longer concerns just individual systems with a well-defined narrow user population. Today, RE takes place in complex "ecosystems" where the system under study co-evolves and inter-depends with many different context systems, ranging from technological advances in embedding systems, hardware, software, and communications, to social changes in the user environment and changes in organizational structure and processes.

This observation is not new. The DAIDA project has tried to capture the interrelationships between systems and their context in a "four-worlds" model of interacting subject world, usage world, system world, and development world almost twenty years ago [1], [2]. However, the interplay of contexts is continuously becoming more pronounced. In terms of technological innovation, the *convergence* of communication, computing, and media in the Web 2.0 and next in the Mobile Internet, is followed up by networked embedded systems in the so-called Internet of Things [3], [4]. At the organizational level, worldwide platform strategies in global company networks enable an accelerating stream of new information-integrated products and services.

To cope with a world whose complexity and dynamics continuously increase, RE should therefore intensify its interdisciplinary interaction with engineers and architects, marketing specialists and business strategists, in order to develop a deeper understanding of the different aspects of context evolution.

In this chapter, we look at some specific challenges from new technologies and system architectures, using examples from our recent research for illustration. The presentation illustrates three hypotheses concerning the future role of RE:

– *The confluence of software engineering and traditional engineering disciplines in embedded and networked systems offers a chance for RE to serve as a* lingua franca *across disciplines*. In the German ZAMOMO project, researchers and practitioners from control engineering and software engineering are attempting to evolve requirements engineering methods such as i* towards a unified theory of model-based development for engine control.
– *Successful Internet information systems must strike a delicate balance between autonomy, flexibility, and governance.* RE must help open up structured standard software for social exchange, and it must help social web communities to organize themselves better to function within organizations. Complementing some large-scale ongoing projects by ERP vendors towards a semantic business web, we have experimented with a reflective webservice architecture called ATLAS by which Web 2.0 communities can organize and monitor their cooperation.
– *Requirements and innovation opportunities are constrained and driven by emerging software architecture standards and widely used software platforms.* RE thus needs to understand the role of standards and platforms better. We look at the systematic exploitation of platform strategies in large as well as small software companies, transferring experiences from other industries.

2   The Impact of Technology Convergence

The convergence of computing, media, internet, and possibly embedded systems technologies creates culture clashes that involve issues of terminology and method integration, but also of governance and power in both designer and user communities. Thus, the requirements engineering processes accompanying the development of software-intensive systems almost always also imply a clash of cultures. But this is not just a challenge for RE but also an opportunity. In fact, who if not RE could deal with such interdisciplinary settings?

In this section, we present two examples of such clashes together with RE concepts we have developed in an attempt to overcome them. The first of these cases deals with the interrelationship between structured organizational IS and unstructured web communications along the lines of Web 2.0, while the second addresses the interplay between control engineers and software engineers in the development of embedded engine control systems for the automotive industries.

2.1   Self-monitoring of Requirements in Internet Communities

The struggle between work practice culture and structured organizational information management that has traditionally defined the clash between the HCI/ CSCW research community and the business informatics/ information systems community [5], has continued into the Internet Age. The "Social Web" or "Web 2.0" [6] complements and sometimes undermines structured ERP systems, cf. fig. 1. Social software tools such as blogs, wikis, and the like are often used for work-arounds of standard

enterprise software. Techno-organizational attempts to contain these work-arounds by semantic standardization – such as the huge German semantic business web project THESEUS (theseus-programm.de) – are still in their early stages.
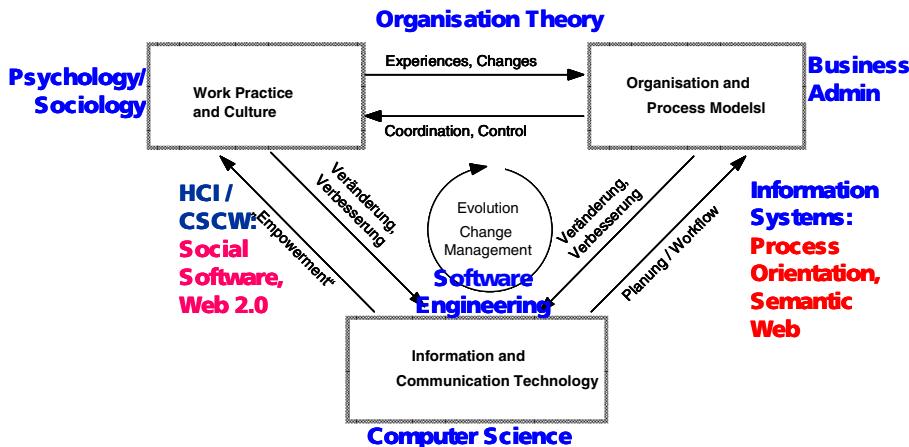
Fig. 1. Cooperative information systems framework: social vs. semantic web [7]

Requirements engineering must help open up structured standard software for social exchange, and it must help social web communities to organize themselves better to function within organizations. Complementing some large-scale ongoing projects by ERP vendors towards a semantic business web, we have experimented with a reflective architecture called ATLAS by which Web 2.0 communities can organize and monitor their cooperation [7].
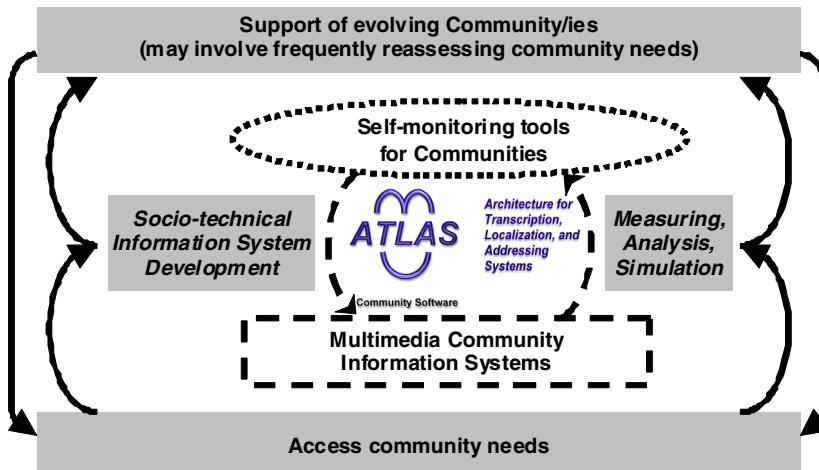
Fig. 2. ATLAS reflective architecture for community information systems

As shown in fig. 2, ATLAS comprises two interacting layers. The *operational layer* comprises an extensible lightweight application server (LAS) which has a small enough footprint to operate also on small mobile devices if necessary. At the same time, LAS is specifically designed for cross-media applications because it relies on the MPEG 7 and MPEG 21 metadata standards for multimedia objects rather than on proprietary "sticky" metadata as most current commercial systems do. Second, it contains a customizable security concept which is oriented both towards user roles and individual objects. Similar to other application servers, LAS includes standard backend links to many different kinds of databases (including XML databases), clients and programming language environments.

Most community environments have proven brittle with respect to changing community rules and lacking features. Rather than influencing their existing internet communities, users often vote with their feet and simply move on to other community platforms if community technology or community behavior does not suit their interests any more. A good example is the extremely rapid move of millions of highschool and university students when unwanted intrusion from commerce and older persons happened in generic contact platforms such as MySpace or Xing, into student-specific like Facebook or StudiVZ with strict access rules and directly domain-relevant features.

The *reflection layer* of ATLAS therefore strives for an integrated community requirements monitoring and revision process that enables the community itself to change its rules rather than being left or destroyed. ATLAS offers a pattern-based set of self-monitoring tools whose patterns encode ideas from requirements engineering (e.g. i* [8]), Social Network Analysis [9], and Actor-Network Theory [10] to identify e.g. undesirable patterns of behavior in a community. For example, i*-based patterns can identify dependencies between subcommunities or goal-related issues, SNA-based patterns evaluate subcommunity coherence, actor relevance and the like, and ANT-based patterns specify desirable or undesirable relationships between human actors and media objects. Moreover, dynamic changes of these patterns over time can also be monitored [11]. Patterns can be defined (or selected from a library) by the community itself, and the community interaction rules can subsequently be adapted e.g. via the security mechanisms of the LAS.

ATLAS provides communities with great flexibility to organize their multimedia communications. This has been exploited by a large number of communities ranging from eLearning environments for students of the Humanities, via domain-specific discussion groups e.g. of aphasics patients, to mobile cultural reconstruction work in emerging countries like Afghanistan. The experiments show a great deal of acceptance from users with very limited IT training, but we still see it a great challenge in linking such concepts with organizational strategy concepts, such as the people integration level in SAP's NetWeaver architecture (see section 3). Nevertheless, the conclusion can be drawn that – in this age of rapidly changing community-determined requirements – research on requirements monitoring [12] will take center-stage to guide end-user driven systems evolution [13], [14].

2.2   Managing Culture Clash in Embedded Systems Engineering

For many years, the control systems for car engines were designed by control engineers. However, in the last decade, it has been recognized that massive reductions in

pollution and gas consumption can only be achieved if software-based controls are embedded in these systems. This kind of embedded control systems has become a very complex market, with over 1500 variants of gasoline engine control systems sold per year by a single major supplier [15].

However, the development culture has not yet followed this technology integration. Most pressing cost problems are nowadays caused by software development issues such as requirements, reusability, and customer-relevant quality aspects, rather than by the functional requirements specified by control engineers. Nevertheless, few cost models exist for software in such systems, product prices are determined by the hardware, and this hardware is often fixed before the software development even begins.

Behind these problems lie, according to a study in the above-mentioned supplier organization [15], strong differences in perception.
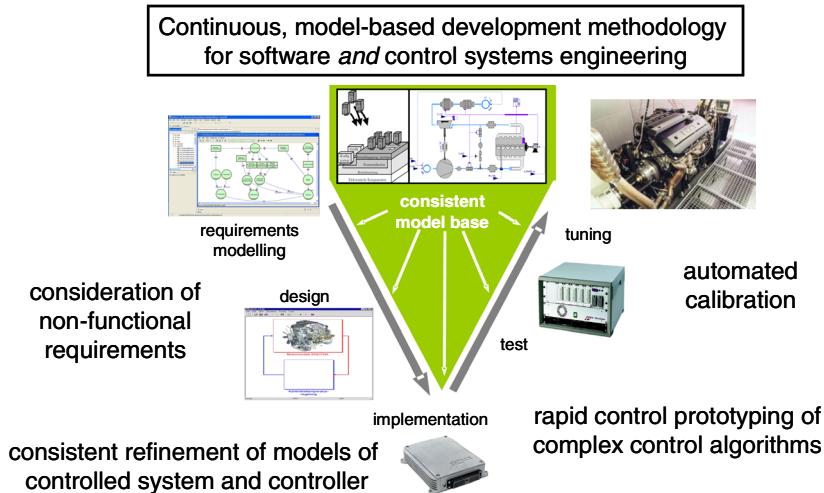
Control engineers tend to consider software system structure trivial and believe the control requirements to be the key issue in design. The algorithms supposedly follow easily from the control requirements which are typically determined using complex mathematical simulation models based on differential equations. The task of software engineers remains simply the implementation of these models, with a responsibility for software quality alone.

Understandably, the software engineers see it differently. They consider that control engineers botch up the overall software architecture by their purely functional control systems designs in a hardly repairable manner, and believe to the contrary that system structure should be defined from the non-functional quality requirements.

Traditionally, this conflict has been circumvented by computer-aided control design tools with automatic code generation to avoid software engineering considerations. This work-around is no longer acceptable in today's extremely competitive market with its customer demand for excellent driving satisfaction and reliability at reduced environmental impact and gas consumption, and related pressure by the public and new legislation.

Besides the political power struggle between control engineers and software engineers that is obviously involved here, there is also an RE problem of severe technical misunderstandings between the two disciplines. In the German ZAMOMO project [15], [16], a consortium from RE, embedded software engineering, control engineering researchers, plus some vendor companies, has therefore been looking for a conceptual bridge across this traditional chasm.

A key observation underlying our approach was that both research communities are drifting towards *model-based paradigms*: Control engineers employ model-based techniques for the systematic simulation-based analysis of control models by which different very subtle variants of proposed cybernetic control cycles can be evaluated against each other and from which control code can be generated. Software engineers strive for model-based architectural designs for functional as well as non-functional requirements, in order to reduce the effort to specify, quality-check and implement different trade-offs between competing requirements, again including semi-automated code production from model-based specifications. However, the kinds of models (differential equations of largely analog hardware systems vs.

Fig. 3. i*-based requirements modeling as common ground for control and software engineers

discrete specification models of software systems verified e.g. by model-checking) are very different.

In ZAMOMO, we are therefore trying to abstract the basic concepts of model-based controller design and model-based software development, to a joint conceptual level based on a variant the i* model similar to the ones pursued in the Tropos project [17]. In the requirements phase, it becomes thus possible to specify the functional requirements for the technical side interacting with the non-functional requirements related to the business side. Both together are embedded in an overall architectural conceptual model. For the overall optimization of a specific engine control system, the non-functional driving qualities and the costs (things seen by the customer) are then the main optimization criteria, whereas the functional specifications of the control cycle act more as optimization constraints. Fig. 3 shows how the thus generated requirements models are exploited in the further process, i.e. as a basis for simulation, rapid prototyping, and testing.

Fig. 4 shows some more details of the requirements analysis process itself. An important side effect of ZAMOMO has been that, for the first time, it becomes possible to introduce a decent version and configuration management among the thousands of engine control variant designs that accumulate quickly even in a small vendor company. Interestingly, this aspect was the first of the ZAMOMO results which has been commercially installed by one of the project partners and is further developed into a marketable product for other engineering organizations.

The ZAMOMO project is still ongoing, but initial experiences indicate that the shared conceptual abstraction of mathematically and technically rather heterogeneous development approaches can indeed assist in bridging the gaps of understanding inherent in embedded systems design. Many questions in this area remain open, however, and require future much deeper research.
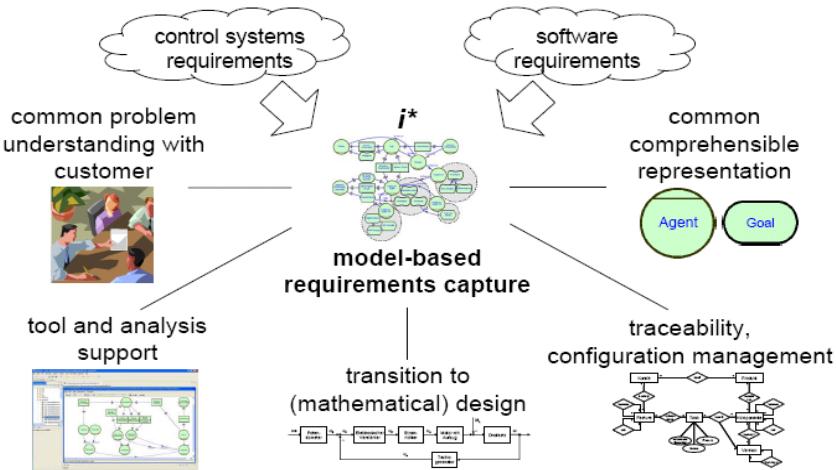
Fig. 4. Requirements modeling, model analysis and management in ZAMOMO

3 The Impact of Standards and Platforms

So far, we have looked at some RE issues induced by individual new technologies and architectures. However, the model and culture clashes discussed there, and the processes and governance mechanisms involved, are embedded in the larger context of evolving technological standards and organizational platform strategies.

In his bestselling book "The World is Flat" [18], T.L. Friedman makes the argument that chances for massively successful innovation are defined by the maturity of open software standards and related infrastructures. From his discussion, it is furthermore obvious that open standards in the field of ICT have evolved *bottom-up* -- lower-level standards build platforms on which innovations at the next-higher level of systems and standards can become successful:

- In the early 1990's, the fiber-optics "*Information Highway*" started by the Clinton/ Gore administration enabled the wide success of the Internet protocol TCP/IP.
- Berners-Lee's HTTP/HTML protocol in combination with browser technology enabled the success of the *World Wide Web*, initially as a publication medium. In parallel, the GSM standard enabled explosive growth in mobile telephony; in 2007 alone, the number of mobile phones rose by 50% to 3.1 billion worldwide, making mobile phones by far the most promising medium for information dissemination and communication.
- The advent of open source development tools such as Apache allowed the broad deployment of *web servers*, and thus paved the way towards the social web.
- Data and service "atom standards" such as XML or SOAP enable workflow "molecule standards" such as AJAX to execute heterogenous business web applications across platforms with relative ease, at least at a syntactic level.

A few proprietary (and very expensive) solutions on upper-layer systems usually exist prior to the advent of standards at their layer. When new standards take grip, whole layers of technical requirements – and related skill needs – are threatened, usually to be replaced by new higher-level requirements and required skill sets of developers and users. Large-scale proprietary solutions suddenly become legacy software. They can become dinosaurs in danger of extinction, unless they can preserve competitive advantage through deep application domain knowledge.

Research in ICT has often been in this situation. By its very nature, it has frequently addressed problems "too early", jumping layers of standards. When finally the time came that this research could become relevant for practice innovation, researchers were often too slow to take advantage of the resulting window of opportunity, or they are simply too bored to take up again an "old field".

The interface to the evolution of standards is therefore an important, and largely overlooked topic in requirements engineering. How can RE understand and manage systems development under the perspective of expected new standards, and how can we educate our students better about this?

From an organizational perspective, a related important issue for RE is the emergence of *company-owned software platforms* as a competitive strategy. Such a strategy involves exploiting as well as setting standards, often in a limited application domain.

Strategic use of IT platforms is not a completely new phenomenon but was already a topic in the 1980's [19]. For example, early airline reservation systems provided additional income to help airlines through financially difficult times, or structured the birth of large-scale airline alliances [20]; early strategic IS also gave financial institutions like American Express competitive advantage in the young credit card business. Nevertheless, RE research has only recently begun to consider platform strategies as an important piece of context, probably due to the fact that platform construction at the application level is becoming much easier because of the above-mentioned basic standards. Therefore, it may be interesting to look at other industries where platform research has a longer traditiona to guide RE research strategies.

In engineering domains such as the automotive industries, platforms have been a highly successful strategic concept . They are often considered one of the main reasons behind the renewed competitiveness of European car manufacturers after the Japanese dominance [21]. In such a strategy, a car manufacturer selects a few strategic platforms in which – simplistically speaking – everything is standardized that the customer cannot see or otherwise experience directly. Differentiation of individual products and their prices focusses on design aspects directly visible to the customer. Very large car manufacturers such as the Volkswagen group successfully standardize on a few platforms (for subcompacts, compact cars, medium and premium segments) while offering a huge variety of products through different vendors in different countries.

Key arguments for a platform strategy in the literature on this domain include the creation of market entry barriers, the growth of sales through product differentiation and faster time to market, finally cost reduction by economies of scale at the platform level. Additional advantages concern learning effects across different product variants and market segments (i.e. learning from others' mistakes), and synergy effects in the complex multi-level value chain of suppliers and sellers all the way to repair shops.
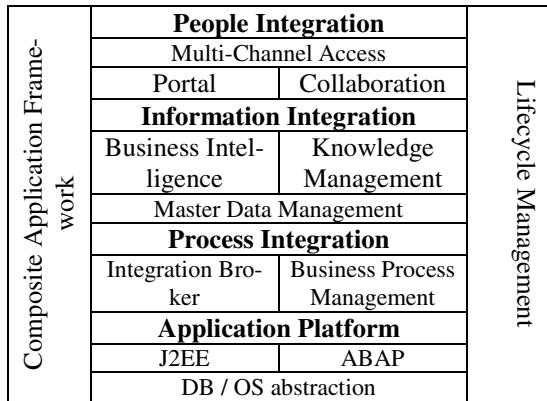
Obviously, there are also *challenges resulting from platform strategies*, mostly due to two aspects: the strictness of managed variability within a platform, and the high costs involved in developing or changing platforms. Platform strategies abandon the 1990's credo of buying "best-of-breed" island solutions, by defining certain design spaces, thus introducing a concept of "managed variability". If these design spaces are too open, standardization within a platform is limited to the lowest common denominator and will not really gain competitive advantage. If they are too narrow, they exclude opportunities for radical innovation, thus threatening to make the whole platform outdated. Last not least, the choice of the right architectural abstractions is important, as market segmentation may change very quickly with changing customer perceptions. A good example is the recent shift of emphasis from sportivity and impressive size of cars to energy savings, pollution reduction, and sustainability.

Many of these challenges are much more striking in the software industry than in the car industry. One of the reasons may be the different cost structures. For example, manufacturing costs for individual cars constitute a high proportion of total costs in the car industry but are neglegible in the software industry where the main costs involve R&D as well as sales. After an often elaborated R&D process, software platforms can therefore grow much quicker than in other engineering sectors, as evidenced by recent examples such as Google or Apple iTunes. This leads to different competitive patterns in which the timing of market entry plays a crucial role, but also the availability of a stable competitive advantage in a certain application domain.

Unfortunately, research in software platforms is much less developed than in other engineering sectors. In [22], approaches from other engineering sectors are merged with traditonal SE approaches to develop a methodology for software product line engineering which divides the software development process in the two separate activities of domain engineering and application engineering. Variability is mostly managed at the level of document specifications, and by constraining interface specifications within a reference architecture. Success stories in large and medium-sized organizations including well-known companies such as Philips, Boeing, Bosch, and ABB, often align the software product families to existing hardware product families and thus stay somewhat behind the strategic platform idea mentioned above.

Poh et al. [22] investigate software product lines from a software engineering perspective and do not really include business considerations. We have therefore recently conducted a comparison of platform strategies in the automotive and software industries from a more strategic perspective [23]. Some of the key decision parameters in this context focus on boundary definitions such as: (a) the differentiation between core and context processes of the organization offering the platform, (b) the market power and the partner network to be involved, (c) coverage of expected technical developments (e.g. convergence of technologies, integration of product and service engineering) promoting or hindering success of the platform, (d) choice of abstractions and granularities based on market needs.

Global players try to establish complex *multi-level platforms* to facilitate their internal development processes as well as third-party contribution from their ecosystem of suppliers, partners, and customers. Examples include Microsoft .NET, IBM Websphere, or SAP NetWeaver, in addition to some open source products. In terms of domain standardization, NetWeaver (cf. fig. 5 and [24]) is probably the most complex

Composite Application Framework	People Integration		Lifecycle Management
	Multi-Channel Access		
	Portal	Collaboration	
	Information Integration		
	Business Intelligence	Knowledge Management	
	Master Data Management		
	Process Integration		
	Integration Broker	Business Process Management	
	Application Platform		
	J2EE	ABAP	
	DB / OS abstraction		

Fig. 5. Layers of SAP's NetWeaver Platform

of these platforms, trying to support the broad diversity of ERP, SCM, CRM products by SAP and its partners at four different levels: application integration (proprietary via ABAP, and open via J2EE), process integration, information integration (business intelligence as well as knowledge management aspects), and people integration (via multiple usage channels and collaboration rooms).

Smaller vendors typically focus on more specialized problem-oriented and/or customer-group oriented platforms. This avoids the need to address legacy issues, but often requires interaction with the platforms of larger players for market access. An interesting example is the recent German start-up Crossgate (www.crossgate.de). Recognizing that the need for *semantically* correct data and document exchange between companies cannot be satisfied by syntactic means such as EDI or XML, Crossgate set up a reference conceptual model of important business terms and their interrelationships, and uses this model as a basis for open document and data exchange across companies. A customer organization just has to map its own data models to this model in order to become part of a growing network of partners that can easily integrate their business processes and data at a semantic level.

The Crossgate platform – a very nice real-world application of what has been recently called model management in the database literature [25], [26] -- has been kickstarted by providing mappings between leading SAP products and the Crossgate reference models, making it extremely easy for SAP customers to join. This cooperation between the SAP platform and the Crossgate platform creates a strategic win-win situation among all partners: SAP keeps customers by offering them more openness than in the past. The customers have to construct at most one model mapping instead of one for each of their business partners and profit automatically from a growing network. Crossgate gains SAP customers as an important seed customer base. For the customers, another advantage is maintainability : a change in internal data structures needs to be only reflected in a single mapping and not in all customer relations.

In terms of challenges for the platform vendors and their partners, competition can grow due to the increased accessibility of services, driving down costs (which correspond to reduced income on the vendor side) and possibly leading to cannibalization of earlier products. For customers, relying too much on specific platforms also entails

obvious risks in terms of vendor dependencies, e.g. if vendors suddenly increase maintenance fees or face financial problems endangering technical support for the platform.

All these issues have major strategic consequences for business process design, inter-organizational collaboration design, and internal IT flexibility for platform customers. They must therefore be considered in their requirements engineering processes. Thus, much more research is needed to better understand software platforms and their implications for requirements engineering, both for the evolution of the platforms themselves and for processes in the ecosystems of these platforms.

4   Concluding Remarks

The convergence of technologies and the emergence of widely used open standards has a strong impact on the future of requirements engineering. As information and communication technologies transcend all science and engineering disciplines, we can summarize the examples given in this paper by claiming three important principles for future RE: *strategic standards awareness, fundamental transdisciplinarity,* and *flexibility via self-control*. The decision about where we define the boundaries of a company platform (if any), and where we begin product differentiation and customization, will also have a strong impact on requirements engineering methods and strategies, both for the platform vendors and for their customers. All of these topics provide rich fields for technical as well as empirical requirements engineering research.

References

1. Mylopoulos, J., Borgida, A., Jarke, M., Koubarakis, M.: Telos: representing knowledge about information systems. ACM Trans. Information Systems 8(4), 325–362 (1990)
2. Jarke, M., Mylopoulos, J., Schmidt, J.W., Vassiliou, Y.: DAIDA: an environment for evolving information systems. ACM Trans Information Systems 10(1), 1–50 (1992)
3. Bullinger, H.J., ten Hompel, M.: Internet der Dinge. Springer, Heidelberg (2007)
4. Floerkemeier, C., Langheinrich, M., Fleisch, E., Mattern, F., Sarma, S.E. (eds.): IOT 2008. LNCS, vol. 4952. Springer, Heidelberg (2008)
5. de Michelis, G., Dubois, E., Jarke, M., Matthes, F., Mylopoulos, J., Schmidt, J.W., Woo, C.C., Yu, E.S.K.: A three-faceted view of information systems. Comm. ACM 41(12), 64–70 (1998)
6. O'Reilly, T.: What is Web 2.0? (last accessed: August 26, 2008), http://www.oreillynet.com
7. Jarke, M., Klamma, R.: Reflective community information systems. In: Manolopoulos, et al. (eds.) Enterprise Information Systems. Proc. 8th Intl. Conf ICEIS, Paphos, Cyprus. LNBIP, vol. 3, pp. 17–28. Springer, Heidelberg (2008)
8. Yu, E.: Modeling Strategic Relationships for Process Reengineering. PhD Thesis, University of Toronto (1995)
9. Scott, J.: Social Network Analysis – A Handbook, 2nd edn. Sage Publ., London (2000)
10. Latour, B.: On recalling ANT. In: Law/Hassard (ed.) Actor-Network Theory and After, Oxford, pp. 15–25 (1999)

11. Klamma, R., Spaniol, M., Denev, D.: PALADIN – a pattern-based approach to knowledge discovery in digital social networks. In: Proc. I-KNOW 2006: 6th Intl. Conf. Knowledge Management, Graz, Austria, pp. 457–464. Journal of Universal Computer Science, Springer, Heidelberg (2006)

12. Robinson, W.N.: Monitoring software requirements using instrumened code. In: Proc. HICSS, pp. 276 (2002)

13. Wulf, V., Jarke, M.: The economics of end user development. Comm. ACM 47(9), 41–42 (2004)

14. Liebermann, H., Paterno, F., Wulf, V. (eds.): End-User Development. Kluwer, Dordrecht (2005)

15. Kowalewski, S.: On the relation between software development and control function development in automotive embedded systems. In: ARTIST Workshop Beyond AUTOSAR, Innsbruck, Austria (March 2006) (invited talk)

16. Schmitz, D., Nissen, H.W., Jarke, M., Rose, T., Drews, P., Hesseler, F.J., Reke, M.: Requirements engineering for control systems development in small and medium-sized enterprises. In: Proc. 12th Intl. Conf. Requirements Engineering (RE 2008), Barcelona, Spain (to appear, 2008)

17. Bresciania, P., Perini, A., Giorgini, P., Giunchiglia, F., Mylopoulos, J.: Tropos – an agent-oriented software development methodology. Autonomous Agents and Multi-Agent Systems 8(3), 203–236 (2004)

18. Friedman, T.L.: The World is Flat. Penguin Books (2005)

19. Ives, B., Learmonth, G.P.: The information system as a competitive weapon. Comm. ACM 27(12), 1193–1201 (1984)

20. Copeland, G.C., McKenney, J.L.: Airline reservation systems: lessons from history. MIS Quarterly 12(3), 353–370 (1988)

21. Goldman Sachs Global Equity Research. Europe Automobiles Competitive Analysis (October 2003)

22. Pohl, K., Böckle, G., Linden, F.: Software Product Line Engineering: Foundations, Principles, and Techniques. Springer, Heidelberg (2005)

23. Langehenke, C., Göttler, M.: Comparison and application of platform concepts for the transformation of business models in automotive and software industries. Unpublished Executive MBA Thesis (in German), RWTH Aachen University (2008)

24. SAP AG: SAP Netweaver and Enterprise Services Architecture: Using IT Practices to Bridge the Worlds of Business and IT, SAP 2006 (last accessed: August 26, 2008), `http://www.sap.com/platform/netweaver/pdf/` `BWP_SID_SAP_NetWeaver_and_ESA.pdf`

25. Bernstein, P.A., Haas, L.M., Jarke, M., Rahm, E., Wiederhold, G.: Panel: Is generic metadata management possible? In: Proc. 26th Intl. Conf. Very Large Data Bases (VLDB 2000), Cairo, Egypt, pp. 660–662 (2000)

26. Bernstein, P.A., Melnik, S.: Model management 2.0: manipulating richer mappings. In: Proc. ACM SIGMOD Conf., Beijing, pp. 1–12 (2007)

Understanding Requirements for Open Source Software

Walt Scacchi

Institute for Software Research, University of California-Irvine
Irvine, CA 92697-3425 USA
wscacchi@ics.uci.edu

Abstract. This study presents findings from an empirical study directed at understanding the roles, forms, and consequences arising in requirements for open source software (OSS) development efforts. Five open source software development communities are described, examined, and compared to help discover what differences may be observed. At least two dozen kinds of software informalisms are found to play a critical role in the elicitation, analysis, specification, validation, and management of requirements for developing OSS systems. Subsequently, understanding the roles these software informalisms take in a new formulation of the requirements development process for OSS is the focus of this study. This focus enables considering a reformulation of the requirements engineering process and its associated artifacts or (in)formalisms to better account for the requirements when developing OSS systems. Other findings identify how OSS requirements are decentralized across multiple informalisms, and to the need for advances in how to specify the capabilities of existing OSS systems.

Keywords: Open source software, Requirements process, Empirical studies, Decentralized software development, Artifacts.

1 Introduction

The focus in this paper is directed at understanding the requirements processes for open source software (OSS) development efforts, and how the development of these requirements differs from those traditional to software engineering and requirements engineering [1], [2], [3], [5]. This study is about ongoing discovery, description, and abstraction of OSS development (OSSD) practices and artifacts in different settings across different communities. It is about expanding our notions of what requirements need to address to account for OSSD. Subsequently, these are used to understand what OSS communities are being examined, and what characteristics distinguish one community from another. This chapter also builds on, refines, and extends earlier study on this topic [6], [7], [8], [9], [10], as well as identifying implications for what requirements arise when developing different kinds of OSS systems.

This study reports on findings and results from an ongoing investigation of the socio-technical processes, work practices, and community forms found in OSSD [10], [11], [12]. The purpose of this multi-year investigation is to develop narrative, semi-structured (i.e., hypertextual), and formal computational models of these processes, practices, and community forms [8], [13]. This chapter presents a systematic narrative

model that characterizes the processes through which the requirements for OSS systems are developed. The model compares in form, and presents an account of, how software requirements differ across traditional software engineering and OSS approaches. This model is descriptive and empirically grounded. The model is also comparative in that it attempts to characterize an open source requirements engineering process that transcends the practice in a particular project, or within a particular community. This comparative dimension is necessary to avoid premature generalizations about processes or practices associated with a particular OSS system or those that receive substantial attention in the news media (e.g., the GNU/Linux operating system). Such comparison also allows for system projects that may follow a different form or version of OSSD (e.g., those in the higher education computing community or networked computer game arena). Subsequently, the model is neither prescriptive nor proscriptive in that it does not characterize what should be or what might be done in order to develop OSS requirements, except in the concluding discussion, where such remarks are bracketed and qualified.

Comparative case studies of requirements or other software development processes are also important in that they can serve as foundation for the formalization of our findings and process models as a process meta-model [14]. Such a meta-model can be used to construct a predictive, testable, and incrementally refined theory of OSSD processes within or across communities or projects. A process meta-model is also used to configure, generate, or instantiate Web-based process modeling, prototyping, and enactment environments that enable modeled processes to be globally deployed and computationally supported (e.g., [8], [13], [15], [16]). This may be of most value to other academic research or commercial development organizations that seek to adopt "best practices" for OSSD processes that are well suited to their needs and situation. Therefore, the study and results presented in this report denote a new foundation on which computational models of OSS requirements processes may be developed, as well as their subsequent analysis and simulation (cf. [13], [17]).

The study reported here entails the use of empirical field study methods [18] that conform to the principles for conducting and evaluating interpretive research design [19] as identified earlier [9].

2   Understanding OSS Development Across Different Communities

We assume there is no general model or globally accepted framework that defines how OSS is or should be developed. Subsequently, our starting point is to investigate OSS practices in different communities from an ethnographically informed perspective [19], [40], [64]. We have chosen five different communities to study. These are those centered about the development of software for networked computer games, Internet/Web infrastructure, bioinformatics, higher education computing, and military computing. The following sections briefly introduce and characterize these OSS subdomains. Along the way, example software systems or projects are *highlighted* or identified via external reference/citation, which can be consulted for further information or review.

2.1 Networked Computer Game Worlds

Participants in this community focus on the development and evolution of first person shooters (FPS) games (e.g., *Quake Arena*, *Unreal Tournament*), massive multiplayer online role-playing games (e.g., *World of Warcraft, Lineage, EveOnline, City of Heroes*), and others (e.g., *The Sims* (Electronic Arts), *Grand Theft Auto* (Rockstar Games)). Interest in networked computer games and gaming environments, as well as their single-user counterparts, have exploded in recent years as a major (now global) mode of entertainment, playful fun, and global computerization movement [22]. The release of *DOOM* [4], an early first-person action game, onto the Web in open source form[1] in the mid 1990's, began what is widely recognized the landmark event that launched the development and redistribution of computer game *mods* [9], [22], [23].

Mods[2] are variants of proprietary (closed source) computer game engines that provide extension mechanisms like game scripting languages (e.g., UnrealScript for mod development with *Unreal* game engines) that can be used to modify and extend a game, and these extensions are licensed for distribution in an open source manner. Mods are created by small numbers of users who want and are able to modify games, compared to the huge numbers of players that enthusiastically use the games as provided. The scope of mods has expanded to now include new game types, game character models and skins (surface textures), levels (game play arenas or virtual worlds), and artificially intelligent game bots (in-game opponents).

Perhaps the most widely known and successful game mod is *Counter-Strike*, which is a total conversion of Valve Software's *Half-Life* computer game developed by two game programmers (Valve Software has since commercialized *CS* and many follow-on versions). Millions of copies of *CS* have been distributed, and millions of people have player *CS* over the Internet, according to http://counterstrikesource.net/. Other popular computer games that are frequent targets for modding include the *Quake, Unreal, Half-Life,* and *Crysis* game engines, *NeverWinter Nights* for role-playing games, motor racing simulation games (e.g., *GTR* series), and even the massively popular *World of Warcraft* (which only allows for modification of end-user interfaces). Thousands of game mods are distributed through game mod portals like MODDB.com. However, many successful game companies including Electronic Arts and Microsoft do not embrace nor encourage game modding, and do not provide end-user license agreements that allow game modding and redistribution.

---

[1] The end-user license agreement for games that allow for end-user created game mods often stipulate that the core game engine (or retail game software product) is protected as closed source, proprietary software that cannot be examined or redistributed, while any user created mod can only be redistributed as open source software that cannot be declared proprietary or sold outright, and must only be distributed in a manner where the retail game product must be owned by any end-user of a game mod. This has the effect of enabling a secondary market for retail game purchases by end-users or other game modders who are primarily interested in accessing, studying, playing, further modifying, and redistributing a game mod.

[2] For introductory background on computer game mods, see
http://en.wikipedia.org/wiki/Mod_(computer_gaming).

2.2 Internet/Web Infrastructure

Participants in this community[3] focus on the development and evolution of systems like the *Apache* web server, *Mozilla/Firefox* Web browser[4], *GNOME* and *K Development Environment* (KDE) for end-user interfaces, the *Eclipse* and *NetBeans* interactive development environments for Java-based Web applications, and thousands of others. This community can be viewed as the one most typically considered in popular accounts of OSS projects. The GNU/Linux operating system environment is of course the largest, most complex, and most diverse sub-community within this arena, so much so that it merits separate treatment and examination. Many other Internet or Web infrastructure projects constitute recognizable communities or sub-communities of practice. The software systems that are the focus generally are not standalone end-user applications, but are often targeted at *system administrators* or *software developers as the targeted user base,* rather than the eventual end-users of the resulting systems. However, notable exceptions like Web browsers, news readers, instant messaging, and graphic image manipulation programs are growing in number within the end-user community

2.3 Bioinformatics

Participants in this community[5] focus on the development and evolution of software systems supporting research into bioinformatics and related computing-intensive biological research efforts. In contrast to the preceding two development oriented communities, OSS plays a significant role in scientific research communities. For example, when scientific findings or discoveries resulting from *in silico* experimentation or observations are reported[6], then members of the relevant scientific community want to be assured that the results are not the byproduct of some questionable software calculation or opaque processing trick. In scientific fields like bioinformatics

---

[3] The SourceForge web portal (http://www.sourceforge.net), the largest associated with the OSS community, currently stores information on more than 1,750K registered users and developers, along with nearly 200K OSSD projects (as of July 2008), with more than 10% of those projects indicating the availability of a mature, released, and actively supported software system. However, some of the most popular OSS projects have their own family of related projects, grouped within their own portals, such as for the Apache Foundation and Mozilla Foundation.

[4] It is reasonable to note that the two main software systems that enabled the World Wide Web, the *NCSA Mosaic* Web browser (and its descendants, like Netscape Navigator, Mozilla, Firefox, and variants like *K-Meleon*, *Konqueror*, *SeaMonkey*, and others), and the Apache Web server (originally know as *httpd*) were originally and still remain active OSSD projects.

[5] For information about OSS projects, activities, and events in this community, see http://www.bioinformatics.org, http://www.open-bio.org, and http://www.open-bio.org/wiki/Upcoming_BOSC_conference.

[6] For example, see [24]. The OSS processing pipelines for each sensor or mass spectrometer are mostly distinct and are maintained by different organizations. However, their outputs must be integrated, and the data source must be registered and oriented for synchronized alignment or overlay, then composed into a final representation (e.g., see [24]). Subsequently, many OSS programs may need to be brought into alignment for such a research method and observation,for a scientific discovery to be claimed and substantiated [25].

that critically depend on software, open source is considered an essential precondition for research to proceed, and for scientific findings to be trusted and open to independent review and validation. Furthermore, as discoveries in bioinformatics are made, this in turn often leads to modification or extension of the astronomical software in use in order to further explore and analyze newly observed phenomena, or to modify/add capabilities to how *in silico* mechanisms operate.

2.4   Higher Education Computing

Participants in this community focus on the development and evolution of software supporting educational and administrative operations found in large universities or similar institutions. This community should not in general be associated with the activities of academic computer scientists nor of computer science departments, unless they specifically focus on higher education computing applications (which is uncommon). People who participate in this community generally develop software for academic teaching or administrative purposes in order to explore topics like course management (*Sakai, Moodle*), campuswide information systems/portals (*uPortal*), Web-based academic applications (*Fluid*), and university e-business systems [26] (for collecting student tuition, research grants administration, payroll, etc. -- *Kuali*). Projects in this community[7] are primarily organized and governed through multi-institution contracts, annual subscriptions, and dedicated staff assignments [27]. Furthermore, it appears that software developers in this community are often not the end-users of the software the develop, in contrast to most OSS projects. Accordingly, it may not be unreasonable to expect that OSS developed in this community should embody or demonstrate principles or best practices in administrative computing found in large public or non-profit enterprises, rather than commercial for-profit enterprises. This includes the practice of developing explicit software requirements specification documents prior to undertaking system development. Furthermore, much like the bioinformatics community, members of this community expect that when breakthrough technologies or innovations have been declared, such as in a refereed conference paper or publication in an educational computing journal, the opportunity exists for other community members to be able to access, review, or try out the software to assess and demonstrate its capabilities. Furthermore, there appears to be growing antagonism toward commercial software vendors (Blackboard Inc., PeopleSoft, Oracle) whose products target the higher education computing market (e.g., WebCT). However, higher education computing software is intended for routine production use by administrative end-users and others, and not research-grade "proof of concept" demonstration or prototype systems that are found in academic research laboratories.

---

[7]   For information about OSS projects, events, and activities in this community, see http://www.sakaiproject.org, http://www.moodle.org, http://www.uportal.org, http://www. fluidproject. org, http://www.kuali.org, as well as EDUCAUSE (http://www. educause.edu/), a non-profit association focusing on current issues in information technology for higher education, including OSS development and OSS policy in academia.

2.5 Military Computing

Participants in this community[8] focus on the development and deployment of computing systems and applications that support secured military and combat operations. Although information on specific military systems may be limited, there are a small but growing number of sources of public information and OSS projects that support military and combat operations. Accordingly, it is becoming clear that the future of military computing, and the future acquisition of software-intensive, mission-critical systems for military or combat applications will increasingly rely on OSS [28], [29], [30], [31], [32], [33], [34], [35]. For example, the U.S. Army relies on tactical command and control systems hosted on Linux systems that support *Apache Tomcat* servers, *Jabber/XMPP* chat services, and *JBoss*-based Web services [30]. Other emerging applications are being developed for future combat systems, enterprise systems (the U.S. Department of Defense is the world's largest enterprise, with more than 1 million military and civilian employees), and various training systems, among others [33], [34], [35]. The development of software systems for developing simulators and game-based virtual worlds [36] are among those military software projects that operate publicly as a "traditional" OSS project that employs a GPL software license, while other projects operate as corporate source (i.e., OSS projects behind the corporate firewall) or community source projects, much like those identified for higher education computing [27].

2.6 Overall Cross-Community Characteristics

In contrast to efforts that draw attention to generally one (but sometimes many) open source development project(s) within a single community (e.g., [37], [38], [39]), there is something to be gained by examining and comparing the communities, processes, and practices of OSSD in different communities. This may help clarify what observations may be specific to a given community (e.g., GNU/Linux projects), compared to those that span multiple, and mostly distinct communities. In this study, two of the communities are primarily oriented to develop software to support scholarly research or institutional administration (bioinformatics and higher education computing) with rather small user communities. In contrast, the other three communities are oriented primarily towards software development efforts that may replace/create commercially viable systems that are used by large end-user communities. Thus, there is a sample space that allows comparison of different kinds.

   Each of these highlighted items point to the public availability of data that can be collected, analyzed, and re-represented within narrative ethnographies [40], [41], computational process models [13], [14], [17], or for quantitative studies [42], [43]. Significant examples of each kind of data have been collected and analyzed as part of this ongoing study. This paper includes a number of OSSD artifacts as data exhibits that empirically ground our analysis. These artifacts serve to document the social actions and technical practices that facilitate and constrain OSSD processes [7], [10],

---

8   The primary source of information about OSS projects in the military comes from the cited references, rather than from publicly accessible Web sites. However, there are a few Military OSS projects accessible on the Web such as the Delta3D game engine at http://www.Delta3D.org, used to developed military training simulations.

[12], [44], [45]. Subsequently, we turn to review what requirements engineering is about, in order to establish how the process of developing OSS system requirements is similar or different than is common to traditional software engineering and information system development practices.

3   Informalisms for Describing OSS Requirements

The functional and non-functional requirements for OSS systems are elicited, analyzed, specified, validated, and managed through a variety of Web-based artifacts. These descriptive documents can be treated as software informalisms. *Software informalisms* [9] are the information resources and artifacts that participants use to describe, proscribe, or prescribe what's happening in a OSSD project. They are informal narrative resources codified in lean descriptions [cf. 46] that coalesce into *online document genres* (following [47], [48]) that are comparatively easy to use, and publicly accessible to those who want to join the project, or just browse around. In earlier work, Scacchi [9] demonstrates how software informalisms can take the place of formalisms, like "requirement specifications" or software design notations which are documentary artifacts seen as necessary to develop high quality software according to the software engineering community [1], [2], [3], [5]. Yet these software informalisms often capture the detailed rationale, contextualized discourse, and debates for why changes were made in particular development activities, artifacts, or source code files. Nonetheless, the contents these informalisms embody require extensive review and comprehension by a developer before contributions can be made (cf. [49]). Finally, the choice to designate these descriptions as informalisms[9] is to draw a distinction between how the requirements of OSS systems are described, in contrast to the recommended use of formal, logic-based requirements notations ("formalisms") that are advocated in traditional approaches (cf. [1], [2], [3], [5]).

In OSSD projects, software informalisms are the preferred scheme for describing or representing OSS requirements. There is no explicit objective or effort to treat these informalisms as "informal software requirements" that should be refined into formal requirements [3], [51], [52] within any of these communities. Accordingly, each of the available types of software requirements informalisms have been found in one or more of the five communities in this study. Along the way, we seek to identify some of the relations that link them together into more comprehensive stories, storylines, or intersecting story fragments that help convey as well as embody the requirements of an OSS system. Knowledge about who is doing what, where, when, why, and how is captured in different or multiple informalisms.

Two dozen types of software informalisms can be identified, and each has subtypes that can be identified. Those presented here are members of the set of software informalisms that are being used by different OSSD projects. Each OSSD project usually employs only a subset as its informal document ecology (cf. [47], [48]) that meets their interests or needs. There are no guidelines for which informalisms to use

---

9   As Goguen [50] observes, formalisms are not limited to those based on a mathematical logic or state transition semantics, but can include descriptive schemes that are formed from structured or semi-structured narratives, such as those employed in Software Requirements Specifications documents.

for what, only observed practices that recur across OSSD projects. Thus it is premature and perhaps inappropriate to seek to further organize these informalisms into a classification or taxonomic scheme whose purpose is to prescribe when or where best to use one or another. Subsequently, they are presented here as an unordered list since to do so otherwise would transform this analysis from empirically ground, interpretative descriptions into untested, hypothetical prescriptions (cf. 19], [21]).

The most common informalisms used in OSSD projects include (i) communications and messages within project Email [46], (ii) threaded message discussion forums (see Exhibit 1), bulletin boards, or group blogs, (iii) news postings, and (iv) instant messaging or Internet relay chat. These enable developers and users to converse with one another in a lightweight, semi-structured manner, and now use of these tools is global across applications domains and cultures. As such, the discourse captured in these tools is a frequent source of OSS requirements. A handful of OSSD projects have found that summarizing these communications into (v) project digests [7] helps provide an overview of major development activities, problems, goals, or debates. These project digests represent multi-participant summaries that record and hyperlink the rationale accounting for focal project activities, development problems, current software quality status, and desired software functionality. Project digests (which sometimes are identified as "kernel cousins") record the discussion, debate, consideration of alternatives, code patches and initial operational/test results drawn from discussion forums, online chat transcripts, and related online artifacts (cf. [7]). Exhibit 1[10] provides an example of a project digest from the GNUe electronic business software project.

As OSS developers and user employ these informalisms, they have been found to also serve as carriers of technical beliefs and debates over desirable software features, social values (e.g., reciprocity, freedom of choice, freedom of expression), project community norms, as well as affiliation with the global OSS social movement [6], [10], [44].

Other common informalisms include (vi) scenarios of usage as linked Web pages or screenshots, (vii) how-to guides, (viii) to-do lists, (ix) Frequently Asked Questions, and other itemized lists, and (x) project Wikis, as well as (xi) traditional system documentation and (xii) external publications (e.g., [53], [54]). OSS (xiii) project property licenses (whether to assert collective ownership, transfer copyrights, insure "copyleft," or some other reciprocal agreement) are documents that also help to define what software or related project content are protected resources that can subsequently be shared, examined, modified, and redistributed.

Finally, (xiv) open software architecture diagrams, (xv) intra-application functionality realized via scripting languages like Perl and PhP, and the ability to either (xvi) incorporate externally developed software modules or "plug-ins", or (xvii) integrate software modules from other OSSD efforts, are all resources that are used informally, where or when needed according to the interests or actions of project participants.

All of the software informalisms are found or accessed from (xix) project related Web sites or portals. These Web environments are where most OSS software informalisms can be found, accessed, studied, modified, and redistributed [9].

---

[10] Each exhibit appears as a screenshot of a Web browsing session. It includes contextual information seen in a more complete display view, as is common in virtual ethnographic studies (cf. [10], [13], [40]).

Exhibit 1. A project digest that summarizes multiple messages including those hyperlinked (indicated by highlighted and underlined text fonts) to their originating online sources. Source: http://www.kerneltraffic.org/GNUe/latest.html, July 2006.

A Web presence helps make visible the project's information infrastructure and the array of information resources that populate it. These include OSSD multi-project Web sites (e.g., SourgeForge.net, Savanah.org, Freshment.org, Tigris.org, Apache.org, Mozilla.org), community software Web sites (PhP-Nuke.org), and project-specific Web sites (e.g., www.GNUenterprise.org), as well as (xx) embedded project source code Webs (directories), (xxi) project repositories (CVS [53]), and (xxii) software bug reports and (xxiii) issue tracking data base like Bugzilla ([55], http://www.bugzilla.org/). Last, giving the growing global interest in online social networking, it not surprising to find increased attention to documenting various kinds of social gatherings and meetings using (xxiv) social media Web sites (e.g, YouTube,

Flickr, MySpace, etc.) where OSS developers, users, and interested others come together to discuss, debate, or work on OSS projects, and to use these online media to record, and publish photographs/videos that establish group identity and affiliation with different OSS projects.

Together, these two dozen types of software informalisms constitute a substantial yet continually evolving web of informal, semi-structured, or processable information resources that capture, organize, and distribute knowledge that embody the requirements for an OSSD project. This web results from the hyperlinking and cross-referencing that interrelate the contents of different informalisms together. Subsequently, these OSS informalisms are produced, used, consumed, or reused within and across OSSD projects. They also serve to act as both a distributed virtual repository of OSS project assets, as well as the continually adapted distributed knowledge base through which project participants evolve what they know about the software systems they develop and use (cf.[15]).

Overall, it appears that none of these software informalisms would defy an effort to formalize them in some mathematical logic or analytically rigorous notation. Nonetheless, in the three of the five software communities examined in this study, there is no perceived requirement for such formalization (except for higher education computing and military computing), as the basis to improve the quality, usability, or cost-effectiveness of the OSS systems. If formalization of these documentary software informalisms has demonstrable benefit to members of these communities, beyond what they already realize from current practices, these benefits have yet to be articulated in the discourse that pervades each community. However, in contrast, the higher education and military communities do traditionally employ and develop formal requirements specification documents in order to coordinate and guide development of their respective "community source" software projects. Thus, we examine and compare these requirements development practices across all five communities so as to surface similarities, differences, and their consequences.

4   OSS Processes for Developing Requirements

In contrast to the world of classic software engineering, OSSD communities do not seem to readily adopt or practice modern software engineering or requirements engineering processes. Perhaps this is no surprise. If the history of software engineering were to reveal that one of the driving forces for capture and formalize software requirements was to support the needs of procurement and acquisition officials (i.e., not actual users of the resulting software) who want to be sure they know what some future software system is suppose to do once delivered, then requirements (documents) serve as the basis for software development contracts, delivery, and payment schedules. Software requirements are traditionally understood to serve a role in the development of proposed systems prior to their development [cf. 1], rather than for software systems that continuously emerge from networks of socio-technical interactions across a diverse ecosystem of users, developers, and other extant software systems [10], [11], [21]. However, OSS communities do develop software that is extremely valuable, generally reliable, often trustworthy, and readily used within its

associated user community. So, what processes or practices are being used to develop the requirements for OSS systems?

We have found many types of software requirements activities being employed within or across the five communities. However, what we have found in OSSD projects is different from common prescriptions for requirements engineering processes that seem to embraced in varying degrees by the higher education and military community source projects. The following subsections present six kinds of OSS requirements activities and associated artifacts that are compared with those traditional to software requirements engineering.

Exhibit 2. A sample of an asserted requirement to use the kdelibs platform libraries. Source: http://sourceforge.net/tracker/index.php?func=detail&aid=1851183&group_id=165310&atid=835080, June 2008.

4.1 Informal Post-Hoc Assertion of OSS Requirements vs. Requirements Elicitation

It appears that OSS requirements are articulated in a number of ways that are ultimately expressed, represented, or depicted on the Web. On closer examination, requirements for OSS can appear or be implied within an email message or within a discussion thread that is captured and/or posted on a Web site for open review, elaboration, refutation, or refinement. Consider the following example found on the Web site for the Avogardo molecular editor tool (http://avogadro.openmolecules.net) in the bioinformatics community. This example displayed in Exhibit 2 reveals the specification "We should use platform libraries when present. So for KDE, if the kdelibs are present, we should use them." As noted earlier, KDE is an Internet infrastructure community project.

These capabilities (appearing near the bottom of Exhibit 2) highlight implied requirements for the need or desirability of full integration of the Avogadro editor with the KDE functional command dialog system. These requirements are simply asserted without reference to other documents, standards, or end-user focus groups--they are requirements because some developers wanted these capabilities.

Perhaps it is more useful to define OSS requirements as *asserted system capabilities*. These capabilities are *post hoc* requirements characterizing a functional capability that has already been implemented, either in the system at hand, or by reference to some other related system that already exists. Based on observations and analyses presented here and elsewhere [6], [7], [8], [9], [10], [22], [45], it appears that concerned OSS developers assert and justify software system capabilities they support through their provision of the required coding effort to make these capabilities operational, or to modification some existing capability which may be perceived as limited or sometimes deficient. Senior members or core developers in the community then vote or agree through discussion to include the asserted capability into the system's feature set [56], or at least, not to object to their inclusion. The historical record of their discourse and negotiation may be there, within the email or discussion forum archive, to document who required what, where, when, why, and how. However, once asserted, there is generally no further effort apparent to document, formalize, or substantiate such a capability as a system requirement. Asserted capabilities then become taken-for-granted requirements that are can be labeled or treated as *obvious* to those familiar with the system's development.

Another example reveals a different kind required OSSD capability. This case displayed in Exhibit 3, finds a requirements "mission" document that conveys a non-functional requirement for both community development and community software development in the bottom third of the exhibit. This can be read as a non-functional requirement for the system's developers to embrace community software development as the process to develop and evolve the ArgoUML system, rather than through a process that relies on the use of system models represented as UML diagrams.

Perhaps community software development, and by extension, community development, are recognized as socio-technical capabilities that are important to the development and success of this system. Regular practice of such capabilities may also be a method for improving system quality and reliability that can be compared to functional capabilities of existing software engineering tools and techniques that seem to

primarily focus on technical or formal analysis of software development artifacts as the primary way to improve quality and reliability.

The next example reveals yet another kind of elicitation found in the Internet/Web infrastructure community. In Exhibit 4, we see an overview of the MONO project. Here we see multiple statements for would-be software component/class owners to sign-up and commit to developing the required ideas, run-time, (object service) classes, and projects (cf. [45]). These are non-functional requirements for people to volunteer to participate in community software development, in a manner perhaps compatible with that portrayed in Exhibit 3.

Exhibit 3. An OSS mission statement encouraging both the development of software for the community and development of the community. Source: http://www.tigris.org, June 2008.

Exhibit 4. A non-functional requirement identifying a need for volunteers to become owners for yet to be developed software components whose functional requirements are still somewhat open and yet to be determined. Source: http://www.mono-project.com/Todo, June 2008.

The systems in Exhibit 3 must also be considered early in their overall development or maturity, because it calls for functional capabilities that are needed to help make the desired software functionality sufficiently complete for future usage. However, these yet "Todo" software implementation tasks signal to prospective OSS developers, who may want to join a project, as to what kinds of new software functionalities are desired, and thus telegraph a possible pathway for how to become a key

contributor within a large, established OSSD project [45] by developing a proposed software system component or function that some core developer desires.

Thus, in understanding how the capabilities and requirements of OSS systems are elicited, we find evidence for elicitation of volunteers to come forward to participate in community software development by proposing new software development projects, but only those that are compatible with the OSS engineering mission for the Tigris.org community.

We also observe the assertion of requirements that simply appear to exist without question or without trace to a point of origination, rather than somehow being elicited from stakeholders, customers, or prospective end-users of OSS systems. As previously noted, we have not yet found evidence or data to indicate the occurrence or documentation of a requirements elicitation effort arising in a traditional OSSD project. However, finding such evidence would not invalidate the other observations; instead, it would point to a need to broaden the scope of how software requirements are captured or recorded. For example, community source projects found in the higher education community seek to span OSSD practices with traditional software engineering practices, which results in hybrid software development and software requirements practices that do not seem to fully realize the practices (or benefits) of OSS engineering projects like those found at Tigris.org. Early experiences such a hybrid scheme suggest the successful software production may not directly follow [57].

4.2 Requirements Reading, Sense-Making, and Accountability vs. Requirements Analysis

Software requirements analysis helps identify what problems a software system is suppose to address and why, while requirements specifications identify a mapping of user problems to system based solutions. In OSSD, how does requirements analysis occur, and where and how are requirements specifications described? Though requirements analysis and specification are interrelated activities, rather than distinct stages, we first consider examining how OSS requirements are analyzed. In Exhibit 5 from the networked game community for the computer game *Unreal Tournament* (aka, UT3), it seems that game mod developers are encouraged to produce multi-version, continuously improving game mods, so that they can subsequently be recognized as professional game developers. Thus, OSS developers learn that achieving enhanced social status requires development of new software functions (mods) that improve across versions.

In seeking to analyze what is needed to more capably develop UT3 game mods, a game developer may seek additional information from other sources to determine how best to satisfy the challenge of developing a viable game mod. This in turn may lead one to discover and review secondary information sources, such as that shown in Exhibit 6. This exhibit points to still other Web-based information sources revealing both technical and social challenges that must be addressed to successfully develop a viable game mod.

Exhibit 5. An asserted capability (in the center) that invites would-be OSS game developers to make UT3 game mods, including improved versions, of whatever kind they require among the various types of available extensions so they may "get professional status," and possibly win money or other contest prizes. Source: http://www.ut3modding.com/, June 2008.

The notion that requirements for OSS system are, in practice, analyzed via the reading of technical accounts as narratives, together with making sense of how such readings are reconciled with one's prior knowledge, is not unique to the game modding software community. These same activities can and do occur in the other three communities. If one reviews the functional and non-functional requirements appearing in Exhibits 1-6, it is possible to observe that none of the descriptions appearing in these exhibits is self-contained. Instead, each requires the reader (e.g., a developer within the

Exhibit 6. Understanding and analyzing what you need to know and do in order to develop a game mod. Source: http://wiki.beyondunreal.com/wiki/Making_Mods, May 2006.

community) to closely or casually read what is described, make sense of it, consult other materials or one's expertise, and trust that the description's author(s) are reliable and accountable in some manner for the OSS requirements that has been described [38], [50]. Analyzing OSS requirements entails little/no automated analysis, formal reasoning, or visual animation of software requirements specifications prior to the development of proposed software functionality (cf. [1], [5]). Instead, emphasis focuses on understanding what has already been accomplished in existing, operational system functionality, as well as what others have written and debated about it in different, project-specific informalisms. Subsequently, participants in these communities are able to understand what the functional and non-functional requirements are in ways that are sufficient to lead to the ongoing development of various kinds of OSS systems.

4.3 Continually Emerging Webs of Software Discourse vs. Requirements Specification and Modeling
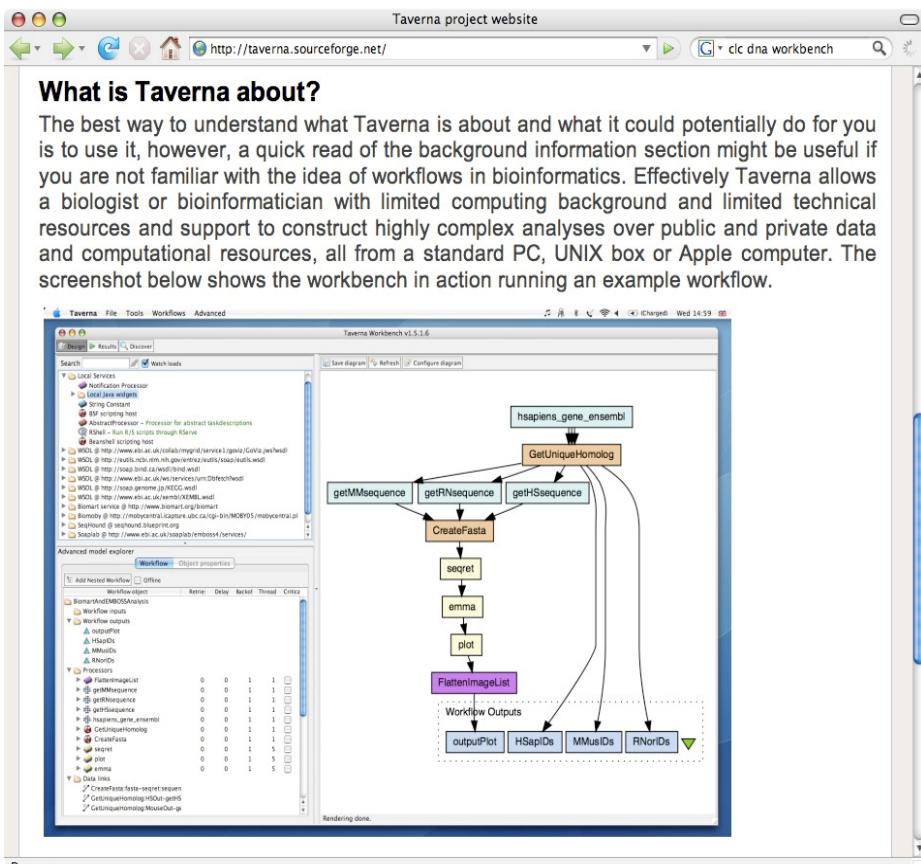
If the requirements for OSS systems are asserted after code-based implementation rather than elicited prior to development of proposed system functionality, how are these OSS requirements specified or modeled? In traditional software development projects, the specification of requirements may be a deliverable required by contract. In most OSSD

projects, there are no such contractual obligations, and there are no requirements specification documents. In examining data from the five communities, of which Exhibits 1-6 are instances, it is becoming increasingly apparent that OSS capabilities can emerge both from the experiences of community participants using the software, as well as through iterative discussion and debate rendered in email and discussion forums. These communication messages in turn give rise to the development of narrative descriptions that more succinctly specify and condense into a web of discourse about the functional and non-functional requirements of an OSS system. This discourse is rendered in multiple, dispersed descriptions that can be found in email and discussion forum archives, on Web pages that populate community Web sites, and in other informal software descriptions that are posted, hyperlinked, or passively referenced through the assumed common knowledge that community participants expect their cohorts to possess. In this way, participating OSS developers and users collectively develop a deep, situated understanding of the capabilities they have realized and how unrealized needs must be argued for, negotiated, and otherwise be found to be obvious to the developers who see it in their self-interest to get them implemented.

In Exhibit 7 from the bioinformatics community, we see passing reference to the implied socio-technical requirement for bioinformatics scientists to program and orchestrate an e-science workflow to perform their research computing tasks. Such workflows are needed to realize a multi-step computational process that can be satisfied through an e-science tool/framework like Taverna (cf. [25], [58]). To comprehend and recognize what the requirements for bioinformatics workflows are in order to determine how to realize some bioinformatics data analysis or *in silico* experiment, community members who develop OSS for such applications will often be bioinformatics scientists (e.g., graduate students or researchers with Ph.D. degrees), and rarely would be simply a competent software engineering professional. Consequently, the bioinformatics scientists that develop software in this community do not need to recapitulate any software system requirement of the problem domain (e.g., microbiology). Instead, community members are already assumed to have mastery over such topics prior to software development, rather than encountering problems in their understanding of microbiology arising from technical problems in developing, operation, or functional enhancement of bioinformatics software. Subsequently, discussion and discourse focuses on how to use and extend the e-science workflow software in order to accomplish the scientific research to be realized through a computational workflow specification. Thus, a web of discourse can emerge about the functional requirement for specifying computational workflows that can be supported and documented by the software capabilities of an OSS workflow modeling tool like Traverna, rather than for specifying the functionality of the tool.

Thus, spanning the five communities and the seven exhibits, we begin to observe that the requirements for OSS are specified in webs of discourse that reference or link:

- email, bboard discussion threads, online chat transcripts or project digests,
- system mission statements,
- ideas about system functionality and the non-functional need for volunteer developers to implement the functionality,

Exhibit 7. A description with embedded screenshot example of the Taverna tool framework for bioinformatics scientists suggesting why and how to develop workflows for computational processes needed to perform a complex data analysis or *in silico* research experiment [41]. Source http://taverna.sourceforce.net June 2008.

- promotional encouragement to specify and develop whatever functionality you need, which might also help you get a new job, and

- scholarly scientific research tools and publications that underscore how the requirements of bioinformatics software though complex, are understood without elaboration, since they rely on prior scientific knowledge and tradition of open scientific research.

Each of these modes of discourse, as well as their Web-based specification and dissemination, is a continually emerging source of OSS requirements from new contributions, new contributors or participants, new ideas, new career opportunities, and new research publications.

4.4   Condensing Discourse That Hardens and Concentrates System Functionality and Community Development vs. Requirements Validation

Software requirements are validated with respect to the software's implementation. The implemented system can be observed to demonstrate, exhibit, or be tested in operation to validate that its functional behavior conforms to its functional requirements. How are the software implementations to be validated against their requirements OSS requirements when they are not recorded in a formal Software Requirements Specifications document, nor are these requirements typically cast in a mathematical logic, algebraic, or state transition-based notational scheme?

In each of the five communities, it appears that the requirements for OSS are co-mingled with design, implementation, and testing descriptions and software artifacts, as well as with user manuals and usage artifacts (e.g., input data, program invocation scripts). Similarly, the requirements are spread across different kinds of artifacts including Web pages, sites, hypertext links, source code directories, threaded email transcripts, and more. In each community, requirements are routinely described, asserted, or implied informally. Yet it is possible to observe in threaded email discussions that community participants are able to comprehend and condense wide-ranging software requirements into succinct descriptions using lean media that pushes the context for their creation into the background.

Consider the next example found on the Web site for the KDE system (http://www.kde.org/), within the Internet/Web Infrastructure community. This example displayed in Exhibit 8 reveals asserted capabilities for the Qt3 subsystem within KDE, as well as displaying and documenting the part of the online discourse that justifies and explains the capabilities of the Qt3 subsystem in a manner that concentrates attention to processing features that the contributors find rationalizes the Qt3 requirements.

Goguen [50] suggests the metaphor of "concentrating and hardening of requirements" as a way to characterize how software requirements evolve into forms that are perceived as suitable for validation. His characterization seems to quite closely match what can be observed in the development of requirements for OSS. We find that requirements validation is a by-product, rather than an explicit goal, of how OSS requirements are constituted, described, discussed, cross-referenced, and hyperlinked to other informal descriptions of system and its implementations.

4.5 Global Access to OSS Webs vs. Communicating Requirements

One distinguishing feature of OSS associated with each of the five communities is that their requirements, informal as they are, are organized and typically stored in a persistent form that is globally accessible. This is true of community Web sites, site contents and hyperlinkage, source code directories, threaded email and other online discussion forums, descriptions of known bugs and desired system enhancements, records of multiple system versions, and more. Persistence, hypertext-style organization and linkage, and global access to OSS descriptions appear as conditions that do not receive much attention within the classic requirements engineering approaches, with few exceptions [51]. Yet, each of these conditions helps in the communication of OSS requirements. These conditions also contribute to the ability of community participants or outsiders looking in to trace the development and evolution of software requirements both within the software development descriptions, as well as across

community participants. This enables observers or developers to navigationally trace, for example, a web of different issues, positions, arguments, policy statements, and design rationales that support (e.g., see Exhibit 8) or challenge the viability of emerging software requirements (cf. [59], [60]).

Exhibit 8. Asserted requirements and condensed justifications producing a hardened rationale for the KDE software subsystem Qt3 expressed through an online discourse. Source: http://dot.kde.org/996206041/, July 2001.

Each of the five communities also communicates community-oriented requirements. These non-functional requirements may seem similar to those for enterprise modeling [5], [61]. However, there are some differences, though they may be minor. First, each community is interested in sustaining and growing the community as a development enterprise (cf. [15]). Second, each community is interested in sustaining and growing the community's OSS artifacts, descriptions, and representations. Third, each community is interested in updating and evolving the community's information sharing Web sites. In recognition of these community requirements, it is not surprising to observe the emergence of commercial efforts (e.g., SourceForge and CollabNet) that offer community support systems that are intended to address these requirements, such as is used in projects like those in Tigris.org, or even the Avogadro project in the Bioinformatics community see (Exhibits 2 and 3).

4.6 Identifying a Common Foundation for the Development of OSS Requirements

Based on the data and analysis presented above, it is possible to begin to identify what items, practices, or capabilities may better characterize how requirements for OSS are developed and articulated. This centers around the preceding OSS requirements processes that enable the emergent creation, usage, and evolution of informal software descriptions as the vehicle for developing OSS requirements.

5   Understanding OSS Requirements

First, there is no single correct, right, or best way/method for constructing software system requirements. The requirements engineering approach long advocated by the software engineering and software requirements community does not account for the practice nor results of OSS system, project, or community requirements. OSS requirements (and subsequent system designs) are different. Thus, given the apparent success of sustained exponential growth for certain OSS systems [62], [63], and for the world-wide deployment of OSSD practices, it is safe to say that the ongoing development of OSS systems points to the continuous development, articulation, adaptation, and reinvention of their requirements (cf. [22]) as capabilities that emerge through socio-technical interactions between people, discursive artifacts, and the systems they use, rather than as needs to be captured before the proposed system comes into use.

Second, the traditional virtues of high-quality software system requirements, namely, their consistency, completeness, traceability, and internal correctness are not so valued in OSSD projects. OSSD projects focus attention and practice to other virtues that emphasize community development and participation, as well as other socio-technical concerns. Thus, as with the prior observation, OSS system requirements are different, and therefore may represent an alternative paradigm for how to develop robust systems that are open to both their developers and users. Nonetheless, there are many examples of the use of tools and techniques for articulating OSS requirements as well as for tracing or monitoring their development (cf. [61]).

Third, OSS developers are generally also end-users of the systems they develop. Thus, there is no "us-them" distinction regarding the roles of developers and end-users, as is commonly assumed in traditional system development practices. Because the developers are also end-users, communication gaps or misunderstandings often found between developers and end-users are typically minimized.

Fourth, OSS requirements tend to be distributed across space, time, people, and the artifacts that interlink them. OSS requirements are thus decentralized—that is, *decentralized requirements* that co-exist and co-evolve within different artifacts, online conversations, and repositories, as well as within the continually emerging interactions and collective actions of OSSD project participants and surrounding project social world. To be clear, decentralized requirements are not the same as the (centralized) requirements for decentralized systems or system development efforts. Traditional software engineering and system development projects assume that their requirements can be elicited, captured, analyzed, and managed as centrally controlled resources (or documentation artifacts) within a centralized administrative authority that adheres to contractual requirements and employs a centralized requirements artifact repository—that is, *centralized requirements*. Once again, OSS projects represent an alternative paradigm to that long advocated by software engineering and software requirements engineering community.

Last, given that OSS developers are frequently the source for the requirements they realize in hindsight (i.e., what they have successfully implemented and released denote what was required) rather than in foresight, perhaps it is better to characterize such software system requirements as instead "software system capabilities" (and not software development practices associated with capability maturity models). She or he who codes determines what the requirements will be based on what they have coded—the open source code frequently appears before there is some online discourse that specifies how and why it was done. OSS developers may simply tell others what was done, whether or not they discussed and debated it beforehand. They are generally under no contractual obligation to report and document software functionality prior to its coding and implementation. Subsequently, OSS capabilities embody requirements that have been found retrospectively to be both implementable and sustainable across releases. *Software capabilities specification*—specifying what the existing OSS system does—may therefore become a new engineering practice and methodology that can be investigated, modeled, supported, and refined. This in turn may then lead to principles for how best to specify software system capabilities.

6 Conclusions

The paper reports on a study that investigates, compares, and describes how the requirements engineering processes occurs in OSSD projects found in different communities. A number of conclusions can be drawn from the findings presented.

First, this study sought to discover and describe the practices and artifacts that characterize how the requirements for developing OSS systems. Perhaps the processes and artifacts that were described were obvious to the reader. This might be true for those scholars and students of software requirements engineering who have already participated in OSS projects, though advocates who have do not report on the

processes described here [37], [38], [39], [54]. For the majority of students who have not participated, it is disappointing to not find such descriptions, processes, or artifacts within the classic or contemporary literature on requirements engineering [1], [2], [3], [5]. In contrast, this study sought to develop a baseline characterization of the how the requirements process for OSS occurs and the artifacts (and other mechanisms). Given such a baseline of the "as-is" process for OSS requirements engineering, it now becomes possible to juxtapose one or more "to-be" prescriptive models for the requirements engineering process, then begin to address what steps are needed to transform the as-is into the to-be [17]. Such a position provides a basis for further studies which seek to examine how to redesign OSS practices into those closer to advocated by classic or contemporary scholars of software requirements engineering. This would enable students or scholars of software requirements engineering, for example, to determine whether or not OSSD would benefit from more rigorous requirements elicitation, analysis, and management, and if so, how.

Second, this study reports on the centrality and importance of software informalisms to the development of OSS systems, projects, and communities. This result might be construed as an advocacy of the 'informal' over the 'formal' in how software system requirements are or should be developed and validated, though it is not so intended. Instead, attention to software informalisms used in OSS projects, without the need to coerce or transform them into more mathematically formal notations, raises the issue of what kinds of engineering virtues should be articulated to evaluate the quality, reliability, or feasibility of OSS system requirements so expressed. For example, traditional software requirements engineering advocates the need to assess requirements in terms of virtues like consistency, completeness, traceability, and correctness [1], [2], [3], [5]. From the study presented here, it appears that OSS requirements artifacts might be assessed in terms of virtues like encouragement of community building; freedom of expression and multiplicity of expression; readability and ease of navigation; and implicit versus explicit structures for organizing, storing and sharing OSS requirements. "Low" measures of such virtues might potentially point to increased likelihood of a failure to develop a sustainable OSS system. Subsequently, improving the quality of such virtues for OSS requirements may benefit from tools that encourage community development; social interaction and communicative expression; software reading and comprehension; community hypertext portals and Web-based repositories. Nonetheless, resolving such issues is an appropriate subject for further study.

Overall, OSSD practices are giving rise to a new view of how complex software systems can be constructed, deployed, and evolved. OSSD does not adhere to the traditional engineering rationality found in the legacy of software engineering life cycle models or prescriptive standards. The development OSS system requirements is inherently and undeniably a complex web of socio-technical processes, development situations, and dynamically emerging development contexts [20], [21], [41], [50], [64]. In this way, the requirements for OSS systems continually emerge through a web of community narratives. These extended narratives embody discourse that is captured in persistent, globally accessible, OSS informalisms that serve as an organizational memory [65], hypertextual issue-based information system [7, 34], and a networked community environment for information sharing, communication, and social interaction [21], [44], [66], [67]. Consequently, ethnographic methods are

needed to elicit, analyze, validate, and communicate what these narratives are, what form they take, what practices and processes give them their form, and what research methods and principles are employed to examine them [5], [10], [13], [40], [41], [50], [64]. This report thus contributes a new study of this kind.

Acknowledgements

References

1. Cheng, B.H.C., Atlee, J.M.: Research Directions in Requirements Engineering. In: Future of Software Engineering (FOSE 2007), pp. 285–303. IEEE Computer Society, Los Alamitos (2007)
2. Davis, A.M.: Software Requirements: Analysis and Specification. Prentice-Hall, Englewood Cliffs (1990)
3. Jackson, M.: Software Requirements & Specifications: Practice, Principles, and Prejudices. Addison-Wesley Pub. Co., Boston (1995)
4. Kushner, D.: Masters of Doom: How Two Guys Created an Empire and Transformed Pop Culture. Random House, New York (2003)
5. Nuseibeh, R., Easterbrook, S.: Requirements Engineering: A Roadmap. In: Finkelstein, A. (ed.) The Future of Software Engineering. ACM Press, New York (2000)
6. Elliott, M., Scacchi, W.: Free Software Development: Cooperation and Conflict in A Virtual Organizational Culture. In: Koch, S. (ed.) Free/Open Source Software Development, pp. 152–172. IGI Publishing, Hershey (2005)
7. Elliott, M., Ackerman, M.S., Scacchi, W.: Knowledge Work Artifacts: Kernel Cousins for Free/Open Source Software Development. In: Proc. ACM Conf. Support Group Work (Group 2007), Sanibel Island, FL, pp. 177–186 (November 2007)
8. Jensen, C., Scacchi, W.: Process Modeling Across the Web Information Infrastructure. Software Process–Improvement and Practice 10(3), 255–272 (2005)
9. Scacchi, W.: Understanding the Requirements for Developing Open Source Software Systems. IEE Proceedings–Software 149(1), 24–39 (2002)
10. Scacchi, W.: Free/Open Source Software Development: Recent Research Results and Methods. In: Zelkowitz, M.V. (ed.) Advances in Computers, vol. 69, pp. 243–295 (2007)
11. Scacchi, W.: Socio-Technical Interaction Networks in Free/Open Source Software Development Processes. In: Acuña, S.T., Juristo, N. (eds.) Software Process Modeling, pp. 1–27. Springer Science+Business Media Inc., New York (2005)
12. Scacchi, W., Feller, J., Fitzgerald, B., Hissam, S., Lakhani, K.: Understanding Free/Open Source Software Development Processes. Software Process–Improvement and Practice 11(2), 95–105 (2006)

13. Scacchi, W., Jensen, C., Noll, J., Elliott, M.: Multi-Modal Modeling, Analysis and Validation of Open Source Software Development Processes. Intern. J. Internet Technology and Web Engineering 1(3), 49–63 (2006)
14. Mi, P., Scacchi, W.: A Knowledge-based Environment for Modeling and Simulating Software Engineering Processes. IEEE Transactions on Knowledge and Data Engineering 2(3), 283–294 (1990)
15. Noll, J., Scacchi, W.: Supporting Software Development in Virtual Enterprises. J. Digital Information 1(4) (February 1999),
    http://jodi.ecs.soton.ac.uk/Articles/v01/i04/Noll/
16. Noll, J., Scacchi, W.: Specifying Process-Oriented Hypertext for Organizational Computing. J. Network and Computer Applications 24(1), 39–61 (2001)
17. Scacchi, W.: Understanding Software Process Redesign using Modeling, Analysis and Simulation. Software Process–Improvement and Practice 5(2/3), 183–195 (2000)
18. Zelkowitz, M.V., Wallace, D.: Experimental Models for Validating Technology. Computer 31(5), 23–31 (1998)
19. Klein, H., Myers, M.D.: A Set of Principles for Conducting and Evaluating Intrepretive Field Studies in Information Systems. MIS Quarterly 23(1), 67–94 (1999)
20. Ackerman, M.S., Atkinson, C.J.: Socio-Technical and Soft Approaches to Information Requirements Elicitation in the Post-Methodology Era. Requirements Engineering 5, 67–73 (2000)
21. Truex, D., Baskerville, R., Klein, H.: Growing Systems in an Emergent Organization. Communications ACM 42(8), 117–123 (1999)
22. Scacchi, W.: Free/Open Source Software Development Practices in the Computer Game Community. IEEE Software 21(1), 59–67 (2004)
23. Cleveland, C.: The Past, Present, and Future of PC Mod Development. Game Developer, 46–49 (February 2001)
24. Cagney, G., Amiri, S., Prewararadena, T., Lindo, M., Emili, A.: Silico proteome analysis to facilitate proteomic experiments using mass spectrometry. Proteome Science 1(5) (2003), doi:10.1186/1477-5956-1-5
25. Oinn, T., Addis, M., Ferris, J., Marvin, D., Senger, M., Greenwood, M., Carver, T., Glover, K., Pocock, M.R., Wipat, A., Li, P.: Taverna: A tool for the composition and enactment of bioinformatics workflows. Bioinformatics J. 20(17), 3045–3054 (2004)
26. Scacchi, W.: Understanding the Development of Free E-Commerce/E-Business Software: A Resource-Based View. In: Sowe, S.K., Stamelos, I., Samoladas, I. (eds.) Emerging Free and Open Source Software Practices, pp. 170–190. IGI Publishing, Hershey (2007)
27. Wheeler, B.: Open Source 2010: Reflections on 2007, EDUCAUSE, pp. 49–67 (January/February 2007)
28. Bollinger, T.: Use of Free and Open-Source Software (FOSS) in the U.S. Department of Defense, The MITRE Corporation (January 2, 2001),
    http://www.terrybollinger.com/dodfoss/dodfoss_html/index.html
29. Guertin, N.: Naval Open Architecture: Open Architecture and Open Source in DOD, Open Source - Open Standards - Open Architecture, Association for Enterprise Integration Symposium, Arlington VA (March 14, 2007)
30. Justice, N.: Open Source Software Challenge: Delivering Warfighter Value, Open Source - Open Standards - Open Architecture, Association for Enterprise Integration Symposium, Arlington VA (March 14, 2007)
31. Riechers, C.: The Role of Open Technology in Improving USAF Software Acquisition, Open Source - Open Standards - Open Architecture, Association for Enterprise Integration Symposium, Arlington VA (March 14, 2007)

32. Scacchi, W., Alspaugh, T.: Emerging Issues in the Acquisition of Open Source Software within the U.S. Department of Defense. In: Proc. 5th Annual Acquisition Research Symposium, Naval Postgraduate School, Monterey, CA (2008)
33. Starrett, E.: Software Acquisition in the Army. Crosstalk: The Journal of Defense Software Engineering, 4–8 (May 2007)
34. Weathersby, J.M.: Open Source Software and the Long Road to Sustainability within the U.S. DoD IT System. The DoD Software Tech. News 10(2), 20–23 (2007)
35. Wheeler, D.A.: Open Source Software (OSS) in U.S. Government Acquisitions. The DoD Software Tech. News 10(2), 7–13 (2007)
36. McDowell, P., Darken, R., Sullivan, J., Johnson, E.: Delta3D: A Complete Open Source Game and Simulation Engine for Building Military Training Systems. J. Defense Modeling and Simulation: Applications, Methodology, Technology 3(3), 143–154 (2006)
37. DiBona, C., Ockman, S., Stone, M.: Open Sources: Voices from the Open Source Revolution. O'Reilly Press, Sebastopol (1999)
38. Pavlicek, R.: Embracing Insanity: Open Source Software Development. SAMS Publishing, Indianapolis (2000)
39. Raymond, E.: The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary. O'Reilly and Associates, Sebastopol (2001)
40. Hine, C.: Virtual Ethnography. SAGE Publishers, London (2000)
41. Kling, R., Scacchi, W.: The Web of Computing: Computer technology as social organization. In: Yovits, M. (ed.) Advances in Computers, vol. 21, pp. 3–90. Academic Press, New York (1982)
42. Howison, J., Conklin, M., Crowston, K.: Flossmole: A collaborative repository for floss research, data, and analysis. Intern. J. Information Technology and Web Engineering 1(3), 17–26 (2006)
43. Madey, G., Freeh, V., Tynan, R.: Modeling the F/OSS Community: A Quantitative Investigation. In: Koch, S. (ed.) Free/Open Source Software Development, pp. 203–221. Idea Group Publishing, Hershey (2005)
44. Elliott, M., Scacchi, W.: Mobilization of Software Developers: The Free Software Movement. Information, Technology and People 21(1), 4–33 (2008)
45. Jensen, C., Scacchi, W.: Role Migration and Advancement Processes in OSSD Projects: A Comparative Case Study. In: Proc. 29th. Intern. Conf. Software Engineering, Minneapolis, MN, pp. 364–374. ACM Press, New York (2007)
46. Yamaguchi, Y., Yokozawa, M., Shinohara, T., Ishida, T.: Collaboration with Lean Media: How Open-Source Software Succeeds. In: Proceedings of the Conference on Computer Supported Cooperative Work (CSCW 2000), Philadelphia, PA, pp. 329–338. ACM Press, New York (2000)
47. Kwansik, B., Crowston, K.: Introduction to the special issue: Genres of digital documents. Information, Technology and People 18(2) (2005)
48. Spinuzzi, C.: Tracing Genres through Organizations: A Sociocultural Approach to Information Design. MIT Press, Cambridge (2003)
49. Lanzara, G.F., Morner, M.: Artifacts rule! How organizing happens in open software projects. In: Czarniawska, B., Hernes, T. (eds.) Actor Network Theory and Organizing. Copenhagen Business School Press, Copenhagen (2005)
50. Goguen, J.A.: Formality and Informality in Requirements Engineering (Keynote Address). In: Proc. 4th. Intern. Conf. Requirements Engineering, pp. 102–108. IEEE Computer Society, Los Alamitos (1996)

51. Cybulski, J.L., Reed, K.: Computer-Assisted Analysis and Refinement of Informal Software Requirements Documents. In: Proceedings Asia-Pacific Software Engineering Conference (APSEC 1998), Taipei, Taiwan, R.O.C., pp. 128–135 (December 1998)
52. Kotonya, G., Sommerville, I.: Requirements Engineering: Processes and Techniques. John Wiley and Sons, Inc., New York (1998)
53. Fogel, K.: Open Source Development with CVS. Coriolis Press, Scottsdale (1999)
54. Fogel, K.: Producing Open Source Software: How to Run a Successful Free Software Project. O'Reilly Press, Sebastopol (2005)
55. Ripoche, G., Gasser, L.: Scalable Automatic Extraction of Process Models for Understanding F/OSS Bug Repair. In: Proc. 16th Intern. Conf. Software Engineering & its Applications (ICSSEA 2003), Paris, France (December 2003)
56. Fielding, R.T.: Shared Leadership in the Apache Project. Communications of the ACM 42(4), 42–43 (1999)
57. Rosenberg, S.: Dreaming in Code: Two Dozen Programmers, Three years, 4732 Bugs, and One Quest for Transcendent Software. Crown Publishers, New York (2007)
58. Howison, J., Wiggins, A., Crowston, K.: eResearch workflows for studying free and open source software development. In: Russo, B., Damiani, E., Hissam, S., Lundell, B., Succi, G. (eds.) IFIP International Federation for Information Processing, Milan, IT. Open Source Development, Communities, and Quality, vol. 275 (2008)
59. Conklin, J., Begeman, M.L.: gIBIS: A Hypertext Tool for Effective Policy Discussion. ACM Transactions Office Information Systems 6(4), 303–331 (1988)
60. Lee, J.: SIBYL: a tool for managing group design rationale. In: Proc. Conf. Computer-Supported Cooperative Work (CSCW 1990), Los Angeles, CA, pp. 79–92. ACM Press, New York (1990)
61. Robinson, W.: A Requirements Monitoring Framework for Enterprise Systems. Requirements Engineering 11(1), 17–41 (2006)
62. Deshpande, A., Riehle, D.: The Total Growth of Open Source Software. In: Russo, B., Damiani, E., Hissam, S., Lundell, B., Succi, G. (eds.) IFIP International Federation for Information Processing, Open Source Development, Communities, and Quality, Milan, IT, vol. 275 (2008)
63. Scacchi, W.: Understanding Free/Open Source Software Evolution. In: Madhavji, N.H., Ramil, J.F., Perry, D. (eds.) Software Evolution and Feedback: Theory and Practice, pp. 181–206. John Wiley and Sons Inc., New York (2006)
64. Viller, S., Sommerville, I.: Ethnographically informed analysis for software engineers. Int. J. Human-Computer Studies 53, 169–196 (2000)
65. Ackerman, M.S., Halverson, C.A.: Reexamining Organizational Memory. Communications of the ACM 43(1), 59–64 (2000)
66. Kim, A.J.: Community-Building on the Web: Secret Strategies for Successful Online Communities. Peachpit Press (2000)
67. Smith, M., Kollock, P. (eds.): Communities in Cyberspace. Routledge, London (1999)

Author Index