

Andreas Jedlitschka
Outi Salo (Eds.)

LNCS 5089

Product-Focused Software Process Improvement

9th International Conference, PROFES 2008
Monte Porzio Catone, Italy, June 2008
Proceedings

 Springer

Commenced Publication in 1973

Founding and Former Series Editors:

Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

David Hutchison

Lancaster University, UK

Takeo Kanade

Carnegie Mellon University, Pittsburgh, PA, USA

Josef Kittler

University of Surrey, Guildford, UK

Jon M. Kleinberg

Cornell University, Ithaca, NY, USA

Alfred Kobsa

University of California, Irvine, CA, USA

Friedemann Mattern

ETH Zurich, Switzerland

John C. Mitchell

Stanford University, CA, USA

Moni Naor

Weizmann Institute of Science, Rehovot, Israel

Oscar Nierstrasz

University of Bern, Switzerland

C. Pandu Rangan

Indian Institute of Technology, Madras, India

Bernhard Steffen

University of Dortmund, Germany

Madhu Sudan

Massachusetts Institute of Technology, MA, USA

Demetri Terzopoulos

University of California, Los Angeles, CA, USA

Doug Tygar

University of California, Berkeley, CA, USA

Gerhard Weikum

Max-Planck Institute of Computer Science, Saarbruecken, Germany

Andreas Jedlitschka Outi Salo (Eds.)

Product-Focused Software Process Improvement

9th International Conference, PROFES 2008
Monte Porzio Catone, Italy, June 23-25, 2008
Proceedings

Volume Editors

Andreas Jedlitschka
Fraunhofer Institute for Experimental Software Engineering
Fraunhofer Platz 1, 67663 Kaiserslautern, Germany
E-mail: andreas.jedlitschka@iese.fraunhofer.de

Outi Salo
VTT Technical Research Centre of Finland
Kaitoväylä 1, 90570 Oulu, Finland
E-mail: Outi.Salo@vtt.fi

Library of Congress Control Number: 2008929491

CR Subject Classification (1998): D.2, K.6, K.4.2, J.1

LNCS Sublibrary: SL 2 – Programming and Software Engineering

ISSN 0302-9743
ISBN-10 3-540-69564-8 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-69564-6 Springer Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.

Springer is a part of Springer Science+Business Media

springer.com

© Springer-Verlag Berlin Heidelberg 2008
Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12320066 06/3180 5 4 3 2 1 0

Preface

On behalf of the PROFES Organizing Committee, we are proud to present to you the proceedings of the 9th International Conference on Product-Focused Software Process Improvement (PROFES 2008) held in Frascati - Monteporzio Catone, Rome, Italy.

Since 1999, PROFES has established itself as one of the recognized international process improvement conferences. The main theme of PROFES is professional software process improvement (SPI) motivated by product and service quality needs. Focussing on a product to be developed, PROFES 2008 addressed both quality engineering and management topics including processes, methods, techniques, tools, organizations, and enabling SPI. Both solutions found in practice and the relevant research results from academia were presented.

Domains such as the automotive and mobile applications industry are growing rapidly, resulting in a strong need for professional development and improvement. Nowadays, the majority of embedded software is developed in collaboration, and distribution of embedded software development continues to increase. Thus, PROFES 2008 addressed different development modes, roles in the value chain, stakeholders' viewpoints, collaborative development, as well as economic and quality aspects. Agile development was included again as one of the themes.

Since the beginning of the series of PROFES conferences, the purpose has been to bring to light the most recent findings and novel results in the area of process improvement, and to stimulate discussion among researchers, experienced professionals, and technology providers from around the world.

The technical program was selected by a committee of leading experts in software process improvement, software process modeling, and empirical software engineering research. This year, 61 papers from 23 nations were submitted, with each paper receiving at least three reviewers. After thorough evaluation, the Program Committee selected 31 technical full papers. The topics addressed in these papers indicate that SPI is still a vibrant research discipline, but is also of high interest for the industry; many papers report on case studies or SPI-related experience gained in industry.

The technical program consisted of the tracks quality and measurement, cost estimation, capability and maturity models, lessons learned and best practices, software process improvement, systems and software quality, and agile software development.

We were proud to have three keynote speakers, Antonia Bertolini, Kurt Schneider, and Horst Degen-Hientz. Interesting tutorials and workshops were co-located with PROFES 2008.

We are thankful for the opportunity to have served as Program Co-chairs for this conference. The Program Committee members and reviewers provided excellent support in reviewing the papers. We are also grateful to the authors, presenters, and Session Chairs for their time and effort in making PROFES 2008 a success. The General Chair, Frank Bomarius, and the Steering Committee provided excellent guidance. We wish to thank Fraunhofer IESE, the VTT Technical Research Centre of Finland, and University of Rome Tor Vergata for supporting the conference. We are also grateful to the authors for high-quality papers, the Program Committee for their hard work in

reviewing the papers, and the Organizing Committee for making the event possible. In addition, we sincerely thank Frank Bomarius for his work as a General Chair of PROFES 2008. Last, but not least, many thanks to Giovanni Cantone and his team at University of Rome Tor Vergata for the local organization of this conference and the maintenance of the PROFES 2008 website, and Sonnhild Namingha and Isabelle Schlitzer at Fraunhofer IESE for her support in copyediting this volume.

June 2008

Andreas Jedlitschka
Outi Salo

Organization

General Chair

Frank Bomarius, Fraunhofer IESE and University of Applied Sciences Kaiserslautern,
Germany

Program Co-chairs

Andreas Jedlitschka, Fraunhofer IESE, Germany
Outi Salo, VTT Technical Research Centre, Finland

Tutorial and Workshop Chair

Darja Šmite, Rigas Informācijas Tehnoloģijas Instituts, Latvia

Organization Chair

Giovanni Cantone, Università degli Studi di Roma Tor Vergata, Italy

Local Organization Committee

Università degli Studi di Roma Tor Vergata, Italy

Anna Lomartire, Centro di Calcolo e Documentazione (CCD)
Gianfranco Pesce, Centro di Calcolo e Documentazione (CCD)
Davide Falessi, Dipartimento di Informatica, Sistemi e Produzione
Maurizio Saltali, Dipartimento di Informatica, Sistemi e Produzione
Alessandro Sarcia, Dipartimento di Informatica, Sistemi e Produzione

PR Chair

S. Alessandro Sarcia, Università degli Studi di Roma Tor Vergata, Italy

Publicity Co-chairs

Benelux	Ko Doorns, Philips
Canada	Dietmar Pfahl, University of Calgary
Central Europe	Frank Seelisch, Fraunhofer IESE

VIII Organization

Finland	Minna Isomursu, VTT
Japan	Shuji Morisaki, NAIST
Scandinavia	Tore Dybå, SINTEF
South America	Christiane Gresse von Wangenheim, Universidade do Vale do Itajaí
USA	Raimund L. Feldmann, FC-MD, USA

Program Committee

Zeiad A. Abdelnabi, Garyounis University - IT College, Libya
Silvia Abrahão, Universidad Politécnica de Valencia, Spain
Muhammad Ali Babar, Lero, University of Limerick, Ireland
Bente Anda, Simula Research Laboratory, Norway
Maria Teresa Baldassarre, University of Bari, Italy
Andreas Birk, SWPM - Software.Process.Management, Germany
Danilo Caivano, University of Bari, Italy
Gerardo Canfora, University of Sannio, Italy
Jeff Carver, Mississippi State, USA
Marcus Ciolkowski, Fraunhofer IESE, Germany
Reidar Conradi, Norwegian University of Science and Technology, Norway
Beniamino Di Martino, Second University of Naples, Italy
Torgeir Dingsøy, SINTEF, Norway
Tore Dybå, SINTEF, Norway
Davide Falessi, University of Rome "Tor Vergata", Italy
Raimund Feldmann, Fraunhofer Center Maryland, USA
Jens Heidrich, Fraunhofer Institute for Experimental Software Engineering, Germany
Martin Höst, Lund University, Sweden
Frank Houdek, Daimler AG, Germany
Hajimu Iida, NAIST, Japan
Katsuro Inoue, Osaka University, Japan
Janne Järvinen, F-Secure, Finland
Erik Johansson, Ericsson Mobile Platforms, Sweden
Natalia Juristo, Universidad Politécnica de Madrid, Spain
Kari Kansala, NOKIA, Finland
Pasi Kuvaja, University of Oulu, Finland
Marek Leszak, Alcatel-Lucent, Germany
Lech Madeyski, Wroclaw University of Technology, Poland
Annukka Mäntyniemi, VTT Technical Research Centre of Finland, Finland
Annukka Mäntyniemi, Nokia, Finland
Kenichi Matsumoto, Nara Institute of Science and Technology, Japan
Makoto Matsushita, Osaka University, Japan
Nils Brede Moe, SINTEF ICT, Norway
Maurizio Morisio, Politecnico di Torino, Italy
Mark Mueller, Robert Bosch GmbH, Germany
Jürgen Münch, Fraunhofer IESE, Germany
Haruka Nakao, Japan Manned Space Systems Corporation, Japan
Risto Nevalainen, FiSMA ry, Finland

Mahmood Niazi, Keele University, UK
Paolo Panaroni, INTECS, Italy
Dietmar Pfahl, University of Calgary, Canada
Minna Pikkarainen, VTT, Finland
Teade Punter, Embedded Systems Institute (ESI), The Netherlands
Austen Rainer, University of Hertfordshire, UK
Karl Reed, La Trobe University, Australia
Daniel Rodríguez, University of Alcalá, Spain
Kurt Schneider, Leibniz Universität Hannover, Germany
Carolyn Seaman, UMBC and Fraunhofer Center Maryland, USA
Darja Smite, University of Latvia, Latvia
Michael Stupperich, Daimler AG, Germany
Guilherme Travassos, COPPE/UF RJ, Brazil
Markku Tukiainen, University of Joensuu, Finland
Mark van den Brand, Eindhoven University of Technology, The Netherlands
Rini van Solingen, LogicaCMG and Delft University of Technology, The Netherlands
Sira Vegas, Universidad Politecnica de Madrid, Spain
Matias Vierimaa, VTT, Finland
Hironori Washizaki, National Institute of Informatics, Japan
Claes Wohlin, Blekinge Institute of Technology, Sweden
Bernard Wong, University of Technology, Sydney, Australia

External Reviewers

Ramón Garcia-Martinez, Buenos Aires Institute of Technology, Argentina
Anna Grimán Padua, Simón Bolívar University, Venezuela
Martín Solari, ORT University, Uruguay
Adam Trendowicz, Fraunhofer IESE, Germany

Table of Contents

Keynote Addresses

Software Testing Forever: Old and New Processes and Techniques for Validating Today's Applications	1
<i>Antonia Bertolino</i>	
Culture of Error Management "Why Admit an Error When No One Will Find Out?"	2
<i>Horst Degen-Hientz</i>	
Supporting Experience and Information Flow in Software Projects	3
<i>Kurt Schneider</i>	

Quality and Measurement I

Goal-Oriented Setup and Usage of Custom-Tailored Software Cockpits	4
<i>Jens Heidrich and Jürgen Münch</i>	
MIS-PyME Software Measurement Maturity Model-Supporting the Definition of Software Measurement Programs	19
<i>María Díaz-Ley, Félix García, and Mario Piattini</i>	
Predicting Software Metrics at Design Time	34
<i>Wolfgang Holz, Rahul Premraj, Thomas Zimmermann, and Andreas Zeller</i>	
A Metrics Suite for Measuring Quality Characteristics of JavaBeans Components	45
<i>Hironori Washizaki, Hiroki Hiraguchi, and Yoshiaki Fukazawa</i>	

Cost Estimation

Software Cost Estimation Inhibitors - A Case Study	61
<i>Ana Magazinovic, Joakim Pernstål, and Peter Öhman</i>	
Impact of Base Functional Component Types on Software Functional Size Based Effort Estimation	75
<i>Luigi Buglione and Cigdem Gencel</i>	
Managing Uncertainty in ERP Project Estimation Practice: An Industrial Case Study	90
<i>Maya Daneva</i>	

The Effect of Entity Generalization on Software Functional Sizing:
 A Case Study 105
*Oktay Turetken, Onur Demirors, Cigdem Gencel,
 Ozden Ozcan Top, and Baris Ozkan*

Capability and Maturity Models

Towards a Capability Model for the Software Release Planning
 Process—Based on a Multiple Industrial Case Study 117
Markus Lindgren, Rikard Land, Christer Norström, and Anders Wall

From CMMI to SPICE – Experiences on How to Survive a SPICE
 Assessment Having Already Implemented CMMI..... 133
Fabio Bella, Klaus Hörmann, and Bhaskar Vanamali

A Model for Requirements Change Management: Implementation of
 CMMI Level 2 Specific Practice 143
*Mahmood Niazi, Charles Hickman, Rashid Ahmad, and
 Muhammad Ali Babar*

Systems and Software Quality

Experience Report on the Effect of Software Development
 Characteristics on Change Distribution 158
*Anita Gupta, Reidar Conradi, Forrest Shull, Daniela Cruzes,
 Christopher Ackermann, Harald Rønneberg, and Einar Landre*

Virtual Prototypes in Developing Mobile Software Applications and
 Devices 174
*Kari Liukkunen, Matti Eteläperä, Markku Oivo,
 Juha-Pekka Soininen, and Mika Pellikka*

Comparing Assessment Methodologies for Free/Open Source Software:
 OpenBRR and QSOS..... 189
Jean-Christophe Deprez and Simon Alexandre

Quality and Measurement II

Predicting Software Fault Proneness Model Using Neural Network 204
Yogesh Singh, Arvinder Kaur, and Ruchika Malhotra

Automating the Measurement of Functional Size of Conceptual Models
 in an MDA Environment 215
Beatriz Marín, Oscar Pastor, and Giovanni Giachetti

How Does a Measurement Programme Evolve in Software
 Organizations? 230
Lasse Harjumaa, Jouni Markkula, and Markku Oivo

A Fault Prediction Model with Limited Fault Data to Improve Test Process	244
<i>Cagatay Catal and Banu Diri</i>	

Software Process Improvement

Big Improvements with Small Changes: Improving the Processes of a Small Software Company	258
<i>Anu Valtanen and Jarmo J. Ahonen</i>	

Software Process Improvement Methodologies for Small and Medium Enterprises	273
<i>Deepti Mishra and Alok Mishra</i>	

An Empirical Study on Software Engineering Knowledge/Experience Packages	289
<i>Pasquale Ardimento and Marta Cimitile</i>	

Customized Predictive Models for Process Improvement Projects	304
<i>Thomas Birkhölzer, Christoph Dickmann, Harald Klein, Jürgen Vaupel, Stefan Ast, and Ludger Meyer</i>	

Lessons Learned and Best Practices I

Improving Customer Support Processes: A Case Study	317
<i>Marko Jänntti and Niko Pykkänen</i>	

Influential Factors on Incident Management: Lessons Learned from a Large Sample of Products in Operation	330
<i>João Caldeira and Fernando Brito e Abreu</i>	

Pitfalls in Remote Team Coordination: Lessons Learned from a Case Study	345
<i>Darja Šmite, Nils Brede Moe, and Richard Torkar</i>	

Agile Software Development

A Model to Identify Refactoring Effort during Maintenance by Mining Source Code Repositories	360
<i>Raimund Moser, Witold Pedrycz, Alberto Sillitti, and Giancarlo Succi</i>	

The Application of ISO 9001 to Agile Software Development	371
<i>Tor Stålhane and Geir Kjetil Hanssen</i>	

Study of the Evolution of an Agile Project Featuring a Web Application
Using Software Metrics 386
*Giulio Concas, Marco Di Francesco, Michele Marchesi,
Roberta Quaresima, and Sandro Pinna*

Lessons Learned and Best Practices II

Identifying and Understanding Architectural Risks in Software
Evolution: An Empirical Study 400
*Odd Petter Nord Slyngstad, Jingyue Li, Reidar Conradi, and
M. Ali Babar*

A Hands-On Approach for Teaching Systematic Review 415
*Maria Teresa Baldassarre, Nicola Boffoli, Danilo Caivano, and
Giuseppe Visaggio*

An Empirical Study Identifying High Perceived Value Practices of
CMMI Level 2 427
Mahmood Niazi, Muhammad Ali Babar, and Suhaimi Ibrahim

Workshops

2nd International Workshop on Measurement-Based Cockpits
for Distributed Software and Systems Engineering Projects
(SOFTPIT 2008) 442
*Marcus Ciolkowski, Jens Heidrich, Marco Kuhrmann, and
Jürgen Münch*

10th International Workshop on: Learning Software Organizations
-Methods, Tools, and Experiences- 443
Raimund L. Feldmann and Martin Wessner

Implementing Product Line Engineering in Industry: Feedback from
the Field to Research 444
Davide Falessi and Dirk Muthig

What to Learn from Different Standards and Measurement Approaches?
Is a Pragmatic Integrative Approach Possible? 445
Fabio Bella and Horst Degen-Hientz

Author Index 447

Software Testing Forever: Old and New Processes and Techniques for Validating Today's Applications

Antonia Bertolino

Istituto di Scienza e Tecnologie dell'Informazione
Consiglio Nazionale delle Ricerche
via Moruzzi, 1, 56124 Pisa, Italy
antonia.bertolino@isti.cnr.it

Software testing is a very complex activity deserving a first-class role in software development. Testing related activities encompass the entire development process and may consume a large part of the effort required for producing software. In this talk, I will first organize into a coherent framework the many topics and tasks forming the software testing discipline, pointing at relevant open issues [1]. Then, among the outlined challenges, I will focus on some hot ones posed by the testing of modern complex and highly dynamic systems [2]. What is assured is that software testers do not risk to remain without their job, and testing researchers are not at short of puzzles. Software testing is and will forever be a fundamental activity of software engineering: notwithstanding the revolutionary advances in the way it is built and employed (or perhaps exactly because of), the software will always need to be eventually tried and monitored. In the years, software testing has evolved from an “art” to a discipline, but test practice largely remains a trial-and-error methodology. We will never find a test approach that is guaranteed to deliver a “perfect” product, whichever is the effort we employ. However, what we can and must pursue is to transform testing from “trial-and-error” to a systematic, cost-effective and predictable engineering practice.

Keywords: Software testing research challenges, Testing and monitoring of dynamic systems, Testing for functional and non-functional properties.

References

1. Bertolino, A.: Software Testing Research: Achievements, Challenges, Dreams. In: 2007 Future of Software Engineering, at ICSE 2007, Minneapolis, USA, May 20 - 26, pp. 85–103 (2007)
2. The Plastic Consortium, Deliverable D4.1: Test Framework Specification and Architecture, <http://www.ist-plastic.org/>

Culture of Error Management

“Why Admit an Error When No One Will Find Out?”

Horst Degen-Hientz*

KUGLER MAAG CIE GmbH Germany
horst.degen-hientz@kuglermaag.com
www.kuglermaag.com

What has a Stradivari and Linux in common? It is the error culture-driven process that created it. A culture of restless strives for innovation and quality enabling continuous learning. We systematically get trained by being punished as child when doing mistakes and often need a life long cumbersome process to undo this conditioning. In western world many organization behave just like as that: errors are socially not acceptable. This seems to be universal applicable as Kaizen and the “zero-defect-culture” can teach us. It is not a society intrinsic attitude - as one can observe from the Toyota way - which took years to establish an organizational error management culture. Studies in Europe show too that organizational error management are a means to boost companies’ performance and goals achievement. Hence, what can we learn from Stradivari and Linux? It is the way to organize error management and innovation. This is key to open source projects and the raising inner source projects as observable in companies like Google.

* CTO and Partner at KUGLER MAAG CIE.

Supporting Experience and Information Flow in Software Projects

Kurt Schneider

Leibniz Universität Hannover
Institut für Praktische Informatik FG Software Engineering
Welfengarten 1, 30167 Hannover, Germany
kurt.schneider@inf.uni-hannover.de

Several large companies have conducted initiatives for systematic learning from experience in software engineering. In the international Software Experience Center (SEC), for example, five companies exchanged experiences and collaborated in building experience exchange mechanisms to be used within each company. Many insights were gained and lessons were learned over the years, among them: (1) Written and documented experiences are the exception rather than the rule. (2) Although not documented in detail or controlled by a process, experience needs guidance and support in order to reach the designated person or group. The “flow” of experience must be kept in mind. (3) Experience is a delicate material, and any avoidable effort or threshold to participate in systematic experience exploitation may endanger stakeholder participation and success. (4) Tools can effectively be built to support orderly flow of experience, but they must be optimized for cognitive support of their users. These lessons learned from supporting experience exploitation can be applied to software projects more generally: Requirements, rationale, and other information flowing through a software project resemble experience with respect to the above-mentioned characteristics: They are often communicated orally rather than in a document. There are diverse processes and practices designed to channel information flow. Early and vague requirements must be handled with care, and tools need to be optimized to reduce cognitive barriers and thresholds, or they will not be accepted. A focus on information and experience flow takes the above-mentioned lessons into account. Information flows within one project, while experience often cuts across several projects. Requirements of one project are useful only in that same project. Experience in designing a product, however, may be reused in subsequent projects. Information and experience flows need to be modelled explicitly. A simple notation is proposed to capture just the essence of flowing information. What may seem like a subtle shift from processes to flows offers a new perspective: Based on those models, dedicated techniques and tools can be developed for analysing and for improving the flows. A wide range of current trends in software engineering can benefit from a better understanding of – and support for – appropriate information flow: Interfaces to the subcontractors, distributed and collaborative teams, Wiki webs, and a variety of new communication channels in global software engineering call for a focus on information flow.

Goal-Oriented Setup and Usage of Custom-Tailored Software Cockpits

Jens Heidrich and Jürgen Münch

Fraunhofer IESE, Fraunhofer Platz 1, 67663 Kaiserslautern, Germany
{jens.heidrich, juergen.muench}@iese.fraunhofer.de

Abstract. Software Cockpits, also known as Software Project Control Centers, support the management and controlling of software and system development projects and provide means for quantitative measurement-based project control. Currently, many companies are developing simple control dashboards that are mainly based on Spreadsheet applications. Alternatively, they use solutions providing a fixed set of project control functionality that cannot be sufficiently customized to their specific needs and goals. Specula is a systematic approach for defining reusable, customizable control components and instantiate them according to different organizational goals and characteristics based on the Quality Improvement Paradigm (QIP) and GQM. This article gives an overview of the Specula approach, including the basic conceptual model, goal-oriented measurement, and the composition of control components based on explicitly stated measurement goals. Related approaches are discussed and the use of Specula as part of industrial case studies is described.

Keywords: Software Project Control Center, QIP, GQM.

1 Introduction

The complexity of software development projects continues to increase. One major reason is the ever-increasing complexity of functional as well as non-functional software requirements (e.g., reliability or time constraints for safety-critical systems). The more complex the requirements, the more people are usually involved in meeting them, which further increases the complexity of controlling and coordinating the project. This, in turn, makes it even harder to develop the system according to plan (i.e., matching time and budget constraints). Project control issues are very hard to handle. Many software development organizations still lack support for obtaining intellectual control over their software development projects and for determining the performance of their processes and the quality of the produced products. Systematic support for detecting and reacting to critical project states in order to achieve planned goals is often missing [15].

Companies have started to introduce so-called software cockpits, also known as Software Project Control Centers (SPCC) [15] or Project Management Offices (PMO) [16], for systematic quality assurance and management support. A software cockpit is comparable to an aircraft cockpit, which centrally integrates all relevant information

for monitoring and controlling purposes. A project manager can use it to get an overview of the project state and a quality assurance manager can use it to check the quality of the software product. In addition to these primary users of an SPCC, basically any role of a project may profit from making direct or indirect use of the SPCC functionality. For instance, a developer can use the SPCC to keep track of code quality or to trace quality issues. The benefit provided by an SPCC for a certain project role depends on the functionality and services offered. However, the needs with respect to project control differ between different organizations, projects, and roles. They depend on organizational goals (business goals), process maturity, the experience of the project team, and many other factors. For instance, for multi-disciplinary, distributed software development, measurement data has to be collected from different sources (locations) and formats. In this case, integration of data is crucial for getting a consistent picture of the project state.

In general, an important success factor in the software engineering domain is that these solutions are customized to the specific goals, organizational characteristics and needs, as well as the concrete project environment. *Specula* (*lat.* watch tower) is an approach for composing project control functionality out of reusable control components [7], [8]. It was mainly developed at the Fraunhofer Institute for Experimental Software Engineering (IESE) and makes use of the Quality Improvement Paradigm (QIP) for integrating project control activities into a continuous improvement cycle. Furthermore, the GQM approach [2] is used for explicitly specifying measurement goals for project control.

Section 2 of the article presents related work in the field of software project control centers and key performance indicators for project control. Section 3 introduces the *Specula* approach, describes the underlying conceptual model and its relationship to goal-oriented measurement, and finally presents the basic steps of the methodology for composing control components (encapsulated, packaged techniques for project control) based on explicitly defined measurement goals. Section 4 presents first empirical evaluation results based on industrial case studies conducted. The article concludes with a brief summary and discussion of future work.

2 Related Work

An overview of the state of the art in Software Project Control Centers can be found in [15]. The scope was defined as generic approaches for online data interpretation and visualization on the basis of past experience. However, project dashboards were not included in this overview. In practice, many companies develop their own dashboards (mainly based on Spreadsheet applications) or use dashboard solutions that provide a fixed set of predefined functions for project control (e.g., deal with product quality only or solely focus on project costs) and are very specific to the company for which they were developed. Most of the existing, rather generic, approaches for control centers offer only partial solutions. Especially purpose- and role-oriented usages based on a flexible set of techniques and methods are not comprehensively supported. For instance, SME (Software Management Environment) [10] offers a number of role-oriented views on analyzed data, but has a fixed, built-in set of control indicators and corresponding visualizations. The SME successor

WebME (Web Measurement Environment) [19] has a scripting language for customizing the interpretation and visualization process, but does not provide a generic set of applicable controlling functions. Unlike Provence [13] and PAMPA [18], approaches like Amadeus [17] and Ginger2 [20] offer a set of purpose-oriented controlling functions with a certain flexibility, but lack a role-oriented approach to data interpretation and visualization.

The indicators used to control a development project depend on the project's goals and the organizational environment. There is no default set of indicators that is always used in all development projects in the same manner. According to [14], a "good" indicator has to (a) support analysis of the intended information need, (b) support the type of analysis needed, (c) provide the appropriate level of detail, (d) indicate a possible management action, and (e) provide timely information for making decisions and taking action. The concrete indicators that are chosen should be derived in a systematic way from the project goals [12], making use of, for instance, the Goal Question Metric (GQM) approach. Some examples from indicators used in practice can be found in [1]. With respect to controlling project cost, the Earned Value approach provides a set of commonly used indicators and interpretation rules. With respect to product quality, there exists even an ISO standard [11]. However, the concrete usage of the proposed measures depends upon the individual organization. Moreover, there is no unique classification for project control indicators. One quite popular classification of general project management areas is given by the Project Management Body of Knowledge (PMBok) [16]. The PMBoK distinguishes between nine areas, including project time, cost, and quality management.

The ideas behind GQM and the Quality Improvement Paradigm (QIP) [2] are well-proven concepts that are widely applied in practice today. An approach based on GQM and QIP to create and maintain enhanced measurement plans, addressing data interpretation and visualization informally, is presented in [5]. Moreover, related work in this field is presented.

3 The Specula Approach

Specula is a state-of-the-art approach for project control. It interprets and visualizes collected measurement data in a goal-oriented way in order to effectively detect plan deviations. The control functionality provided by Specula depends on the underlying goals with respect to project control. If these goals are explicitly defined, the corresponding functionality is composed out of packaged, freely configurable control components. Specula provides four basic components: (1) a logical architecture for implementing software cockpits [15], (2) a conceptual model formally describing the interfaces between data collection, data interpretation, and data visualization [9], (3) an implementation of the conceptual model, including a construction kit of control components [4], and (4) a methodology of how to select control components according to explicitly stated goals and customize the SPCC functionality [9].

The methodology is based on the Quality Improvement Paradigm (QIP) and makes use of the GQM approach [2] for specifying measurement goals. QIP is used to implement a project control feedback cycle and make use of experiences and knowledge gathered in order to reuse and customize control components. GQM is used to drive

the selection process of finding the right control components according to defined goals. Large parts of the approach are supported by a corresponding prototype tool, called Specula Project Support Environment (PSE), which is currently also being used as part of industrial case studies (see Section 4 and [4]). Specula basically addresses the following roles that make use of the provided functionality:

- *Primary Users*: Project manager, quality assurance manager, and controller who mainly use an SPCC to control different aspects of the software development project and initiate countermeasures in case of deviations and risks.
- *Secondary Users*: Developers and technical staff who use an SPCC to enter measurement data as well as to detect root causes for deviations and risks.
- *Administrators*: Administrators who have to install and maintain an SPCC.
- *Measurement Experts*: Experts who define measurement goals, support derivation of control components, and help to customize and effectively use the SPCC.

Section 3.1 gives a brief overview of the conceptual model upon which Specula is built. Section 3.2 addresses the connection of the conceptual model to goal-oriented measurement, and Section 3.3 provides a brief overview of all steps necessary to apply the Specula approach as a whole.

3.1 Cockpit Concepts

The conceptual model of the Specula approach formalizes the process of collecting, interpreting, and visualizing measurement data for software project control. The derived structure for operationally controlling a development project is called a *Visualization Catena (VC)* [7], which defines components for automatically and manually collecting measurement data, processing and interpreting these data, and finally visualizing the processed and interpreted data. The processing and interpretation of collected measurement data is usually related to a special measurement purpose, like analyzing effort deviations, or guiding a project manager. A set of techniques and methods (from the repository of control components) is used by the VC for covering the specified measurement purpose. The visualization and presentation of the processed and collected measurement data is related to roles of the project that profit from using the data. The VC creates a set of custom-made controlling views, which presents the data according to the interests of the specified role, such as a high-level controlling view for a project manager, and a detailed view of found defects for a quality assurance manager. The whole visualization catena has to be adapted in accordance with the context characteristics and organizational environment of the software development project currently being controlled.

Fig. 1 gives an overview of all VC components and their corresponding types. Specula distinguishes between the following five components on the type level from which a concrete VC is instantiated for a certain project:

(T1) Data types describe the structure of incoming data and data that is further processed by the VC. For instance, a time series (a sequence of time stamp and corresponding value pairs) or a project plan (a hierarchical set of activities having a start and end date and an effort baseline) could be logical data types that could either be directly read-in by the system or be the output of a data processing function.

(T2) Data access object packages describe the different ways concrete data types may be accessed. For instance, an XML package contains data access objects for reading data (having a certain data type) from an XML file, writing data to an XML file, or changing the contents of an XML file. A special package may be used, for instance, to automatically connect to an effort tracking system or bug tracking data base. A data access object contains data type-specific parameters in order to access the data repositories.

(T3) Web forms describe a concrete way of managing measurement data manually, involving user interaction. A web form manages a concrete data type. For instance, new data may be added, existing data may be changed or completely removed. A web form also refers to other data types that are needed as input. For instance, in order to enter effort data manually, one needs the concrete activities of the project for which the effort is tracked. Web forms are needed if the data cannot be automatically retrieved from an external data source.

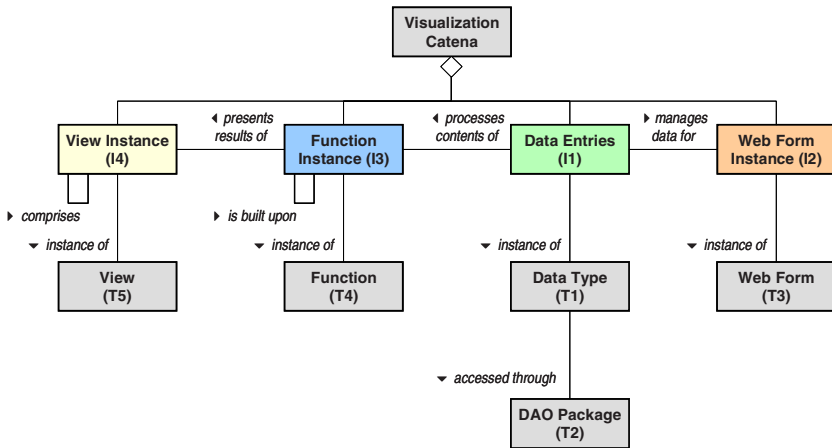


Fig. 1. Overview of the elements of the conceptual model. A view instance presents the results of a data processing function, which in turn processes the contents of data entries for which data is provided by a web form instance.

(T4) Functions represent a packaged control technique or method, which is used to process incoming data (like Earned Value Analysis, Milestone Trend Analysis, or Tolerance Range Checking). A function needs different data types as input, produces data of certain data types as output, and may be adapted to a concrete context through a set of parameters.

(T5) Views represent a certain way of presenting data, like drawing a two-dimensional diagram or just a table with a certain number of rows and columns. A view visualizes different data types and may refer to other views in order to create a hierarchy of views. The latter may, for instance, be used to create a view for a certain project role consisting of a set of sub-views.

In addition, the following components are distinguished on the instances level:

(I1) Data entries instantiate data types and represent the concrete content of measurement data that are processed by the SPCC. We basically distinguish between external and internal data. External data must be read-in or imported from an external location, or manually entered into the system. Each external data object has to be specified explicitly by a data entry containing, for instance, the start and end time and the interval at which the data should be collected. In addition, the data access object package that should be used to access the external data has to be specified. Internal data are the outcome of functions. They are implicitly specified by the function producing the corresponding data type as output and therefore need no explicit specification and representation as data entry. External as well as internal data may be used as input for instances of functions or views if their corresponding data types are compatible.

(I2) Web form instances provide web-based forms for manually managing measurement data for data entries. All mandatory input data type slots of the instantiated web form have to be filled with concrete data entries and all mandatory parameters have to be set accordingly.

(I3) Function instances apply the instantiated function to a certain set of data entries filling the mandatory input slots of the function. A function instance processes (external and internal) data and produces output data, which could be further processed by other function instances or visualized by view instances. All mandatory function parameters have to be set accordingly.

(I4) Finally, view instances apply the instantiated view to a certain set of data entries filling the corresponding mandatory data type slots of the view. A view instance may refer to other view instances in order to build up a hierarchy of views.

Each component of a VC and its corresponding type contains explicitly specified checks that may be used to test whether the specification is complete and consistent, whether data are read-in correctly, whether function instances can be computed accurately, and whether view instances can be created successfully. A visualization catena consists of a set of data entries, each having exactly one active data access object for accessing incoming data, a set of web form instances for managing the defined data entries, a set of function instances for processing externally collected and internally processed data, and finally, a set of view instances for visualizing the processing results. A formal specification of all components may be found in [6].

3.2 Mapping Cockpit Concepts to GQM

For a goal-oriented selection of control components, a structured approach is needed that describes how to systematically derive control components from project goals and characteristics. GQM provides a template for defining measurement goals, systematically derives questions that help to make statements about the goals, and finally derives metrics in order to help answer the stated questions. In order to complete such a measurement plan for a concrete project, each metric can be further described by a data collection specification (DCS) basically making statements on who or which tool has to collect the measurement data at which point in time of the project from which

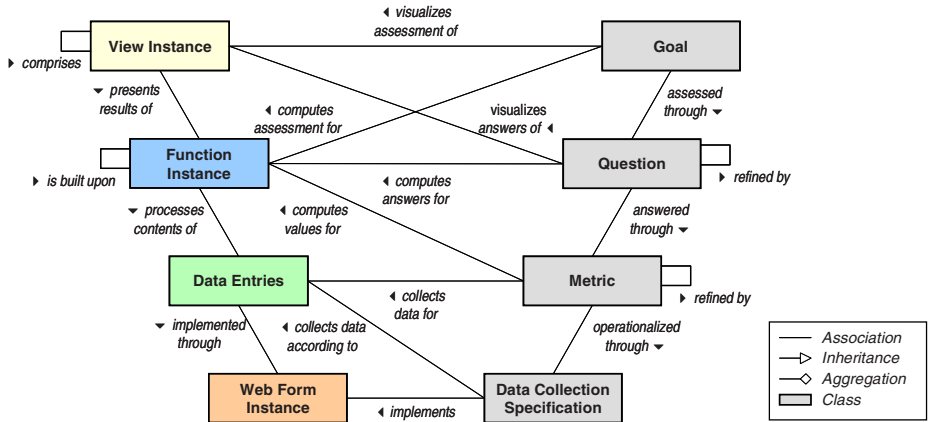


Fig. 2. Mapping the conceptual model to the GQM paradigm. On the left side, one can see the components of the visualization catena. On the right side, one can see the structure of a GQM model and a corresponding data collection specification.

data source. In [8], usage scenarios on how to derive a GQM plan from a control goal and how to define a VC that is consistent with the defined goals are described.

Fig. 2 presents an overview of all relationships between a GQM plan, its DCS, and a visualization catena (cf. [9]):

- Data entries collect measurement data for GQM metrics according to the DCS. If the data has to be collected manually, a web form instance is used to implement the DCS in addition. For instance, if the DCS states that the start and end date of an activity shall be collected from an MS Project file, a corresponding data entry is defined and a web form instance implements importing the project plan from the file.
- Function instances compute metric values if a metric has to be computed from other metrics. For instance, if a cost performance index is computed for an Earned Value Analysis, the budgeted costs of work performed and the actual costs of work performed are needed. A function instance could also compute answers for GQM questions by taking into account all metrics assigned to the question and applying an interpretation model to all metric values. In analogy, a function instance could assess the attainment of a GQM goal by assessing the answers of all assigned questions using an interpretation model.
- View instances visualize the answers to GQM questions. A chart is produced or tables are displayed illustrating the metric results of the corresponding questions and the interpretation model used to answer the question. For instance, the cost performance and schedule performance index could be visualized as a line chart in which good and bad index values are marked accordingly. A view instance could also visualize the assessment of the GQM goal.

3.3 Composing Control Components

Specula is largely based on the Quality Improvement Paradigm (QIP). The basic phases and steps are as follows:

Phase I: Characterize Control Environment: First, project stakeholders characterize the environment in which project control shall be applied in order to set up a corresponding measurement program that is able to provide a basis for satisfying all needs.

- *Describe the project context.* Important characteristics for setting up project control mechanisms have to be defined.
- *Discuss the overall organization.* Organizational characteristics have to be clarified. This includes roles and responsibilities, potential stakeholders, like managers of the organization, project managers, quality assurance manager, developers, and team organization.

Phase II: Set Control Goals: Then, measurement goals for project control are defined and metrics are derived determining what kind of data to collect.

- *Elicit control goals.* The Specula approach makes use of GQM in order to define measurement goals in a structured way. GQM already provides a systematic approach for defining measurement goals, systematically derives questions that help to make statements about the goals, and finally derives metrics in order to help answer the stated questions.
- *Clarify relations to higher-level goals.* The relation to higher-level goals should be modeled. For this purpose, all measurement goals are connected to higher-level software and business goals using the GQM⁺Strategies[®] approach [3].
- *Derive indicators.* Based on the measurement goals defined for project control, questions and metrics have to be derived using GQM.
- *Define GQM model.* A GQM model is created containing the project-specific measurement goals, corresponding questions that make statements about achieving goals, and metrics that support answering the questions.

Phase III: Goal-oriented Composition: Next, a visualization catena is composed based on the defined goals in order to provide online feedback on the basis of the data collected during project execution. More details about this process can be found in [9].

- *Derive measurement plan.* A comprehensive measurement plan has to be derived based on the GQM model, including a data collection specification.
- *Define interpretation models.* Interpretation models are used to basically aggregate measurement data in order to answer a GQM question or make a statement about achieving a GQM goal.
- *Derive data entries and web form instances.* Next, matching data types are identified based on the metric definition, the object to be measured and the quality attribute. For each simple metric (which is not computed from other metrics), instantiate the data type and create a corresponding data entry. The data collection specification is used to determine the start time, end time, and interval when the data should be collected. If the metric has to be collected manually, a web form is identified based on the data source and the instantiated web form is attached to the data entry.
- *Derive function instances for complex metrics.* For each complex metric (which is computed from other metrics), a function is identified that is able to compute the metric based on the metric definition, the object to be measured, and the quality

attribute. The identified functions are instantiated by first filling all input data slots with data entries or results of other function instances. Then, the function instances are parameterized according to the metric definition.

- *Derive function instances for GQM questions.* If an interpretation model is described in the GQM plan that defines how to formally answer a question, a function implementing this model is identified based on the object and quality attribute addressed in order to compute the answers to the question. The functions are instantiated by filling all input data slots with data entries or results of other function instances assigned to the question. The function instances are parameterized according to the interpretation model.
- *Derive view instances for GQM questions.* The answers to the question are visualized by identifying a set of views based on the kind of answers to the question and the data visualization specifications of the measurement plan (if any). The identified views are instantiated by filling all input data slots with data entries or results of function instances assigned to the question. The view instances are parameterized according to the data presented (e.g., title and axis description, size, and color).
- *Derive function instances for GQM goals.* If an interpretation model is described in the GQM plan that defines how to formally assess goal attainment, a function implementing this model is identified and instantiated based on the object and quality focus addressed in order to attain the measurement goal.
- *Derive view instances for GQM goals.* Goal attainment is visualized by identifying and instantiating a set of views based on the kind of assessment of the goal and the data visualization specifications of the measurement plan (if any).
- *Check consistency and completeness.* After defining the whole visualization catena for controlling the project, the consistency and completeness of the mapping process are checked.
- *Configure SPCC.* If the visualization catena is defined and checked, it has to be transferred to a corresponding tool (Specula tool prototype).
- *Provide training.* Training is provided for all SPCC users in order to guarantee the effective usage of the SPCC.

Phase IV: Execute Project Control Mechanisms: Once the visualization catena is specified, a set of role-oriented views are generated by the SPCC for controlling the project based on the specified visualization catena. If a plan deviation or project risk is detected, its root cause must be determined and the control mechanisms have to be adapted accordingly.

- *Perform data collection.* The SPCC users have to perform data collection activities according to the measurement plan defined.
- *Use control views for GQM questions.* The SPCC users have to use the view instances offered to get answers for the GQM questions of their GQM model.
- *Use control views for GQM goals.* The SPCC users have to use the view instances offered to get a general answer with respect to achieving a certain goal of the GQM models.
- *Check SPCC functionality.* The SPCC users should check the correct functionality of the Project Control Center regularly.

Phase V: Analyze Results: After project completion, the resulting visualization catena has to be analyzed with respect to plan deviations and project risks detected in-time, too late, or not detected at all. The causes for plan deviations and risks that have been detected too late or not all have to be determined.

- *Analyze plan deviations and project risks.* The complete lists of plan deviations and project risks have to be analyzed after the end of the project.
- *Analyze measurement plan.* For all deviations and risks that were not detected at all, the measurement plan has to be analyzed with respect to missing goals or other missing parts of the GQM models.
- *Analyze interpretation models.* For all deviations and risks that were not detected at all or that were detected too late, the interpretation models have to be checked to see whether they work as intended or whether metrics or answers to questions need to be interpreted in a different way.
- *Analyze visualization catena.* For all deviations and risks that were detected too late, the components of the visualization catena that helped in detecting them have to be analyzed to see whether they can be improved to support earlier detection in future projects.

Phase VI: Package Results: The analysis results of the visualization catena that was applied may be used as a basis for defining and improving control activities for future projects (e.g., selecting the right control techniques and data visualizations, choosing the right parameters for controlling the project).

4 Empirical Evaluation and Usage Example

The evaluation of the Specula approach is currently being conducted in the context of several industrial case studies as part of the Soft-Pit research project funded by the German Federal Ministry of Education and Research (<http://www.soft-pit.de>). The project focuses on getting experience and methodological support for operationally introducing control centers into companies and projects. The project includes performing several industrial case studies with German companies from different domains, in which the developed control center and its deployment are evaluated. The project is mainly organized into three iterations focusing on different controlling aspects. An application of Specula in the first iteration showed the principal applicability of the VC concept in an industrial environment. Results can be found in [4]. The second iteration focused on three aspects: (a) perceived usefulness and ease of use of the approach, (b) found plan deviations and project risks, and (c) costs for setting up and applying an SPCC. Those aspects were evaluated in four industrial case studies, in which the Specula prototype tool was used to control the software development project. The system was perceived as useful and easy to use. However, the degree of usefulness depended on the group of users: the benefits for secondary users were limited. Usefulness also varied across different organizations; this may be related to the different control mechanisms used before introducing an SPCC. Preliminary results show that following a structured process for setting up an SPCC also does result in a significantly improved detection rate of plan deviations and project risks. The

costs for setting up and applying an SPCC were around 10% of the overall development effort for a medium-sized project (10 team members). In the following, the basic steps of the method are illustrated using data from a practical course conducted at the University of Kaiserslautern in which the Specula project control approach was applied.

Phase I: Characterize Control Environment: The aim was to develop mobile services for creating a virtual office of the future. There were 17 team members. The project manager and quality assurance manager should use an SPCC to control different aspects of the project. In addition, an administrator (not part of the project team) was provided who was familiar with the SPCC tool.

Phase II: Set Control Goals: A measurement expert conducted structured interviews with the project manager and quality assurance manager in order to retrieve the measurement goals with respect to project control that are to be achieved:

- Analyze the project plan for the purpose of monitoring the consistency of the plan from the point of view of the project manager.
- Analyze the project plan for the purpose of comparing the actual effort with the planned effort from the point of view of the project manager.
- Analyze the project plan for the purpose of monitoring schedule adherence from the point of view of the project manager.
- Analyze the project plan for the purpose of monitoring effort tracking regularity from the point of view of the project manager.
- Analyze the source code for the purpose of monitoring the quality from the point of view of the quality assurance manager.
- Analyze the defect detection activities for the purpose of monitoring their efficiency from the point of view of the quality assurance manager.

Phase III: Goal-oriented Composition: A visualization catena was created for the GQM goals above. For example, if the goal is to evaluate the effort plan with respect to plan deviation, the corresponding control components can be selected as follows. Fig. 3 presents the GQM model for this goal on the left side and the corresponding excerpt of the resulting VC on the right side. The one and only question asked was about absolute effort deviation per activity. A complex metric defined the deviation as the amount that an actual effort value is above an effort baseline. Three simple metrics were consequently defined and operationalized by corresponding data collection specifications. The baseline should be extracted from a project plan stored in an MS project file, so a corresponding web form collecting project plan information and data types representing the project activities and the effort baseline were instantiated. The actual effort data should be extracted from the company-wide effort tracking system including effort per person and activity. A data type was instantiated that accesses the tracking system using a corresponding data access object. A function was applied to aggregate the effort data for each activity across all persons. In order to compute the complex metric “effort plan deviation”, a tolerance range checking function was applied that computes the deviation accordingly. Finally, a view was instantiated in order to graphically display the results of the assigned function instances and data entries. Fig. 4 presents the complete visualization catena that was derived for all goals defined as outputted by the Specula prototype tool (instantiation of the concepts shown in Fig. 1). As can be seen, the logical dependency of components is quite high,

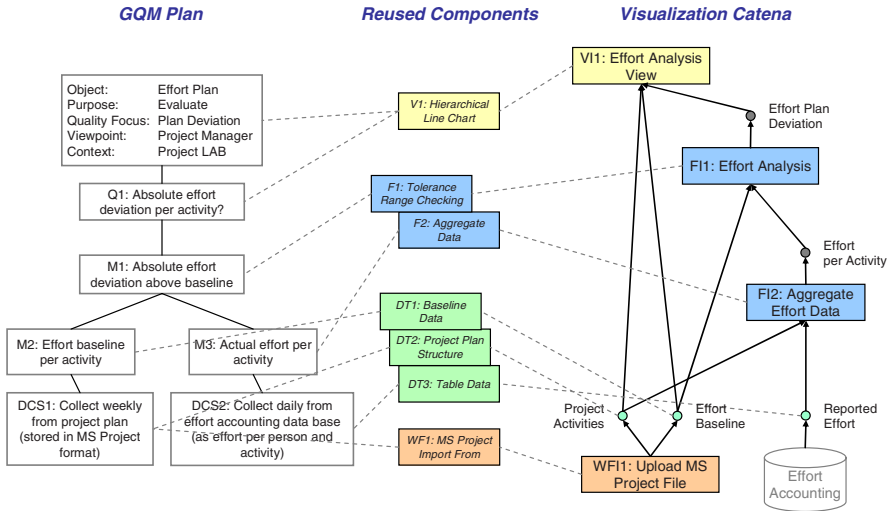


Fig. 3. Composing the VC from reusable components. The left side shows the GQM plan to be implemented by an SPCC. According to the information specified in the GQM plan, components are identified from a reuse repository and instantiated in order to create a visualization catena.

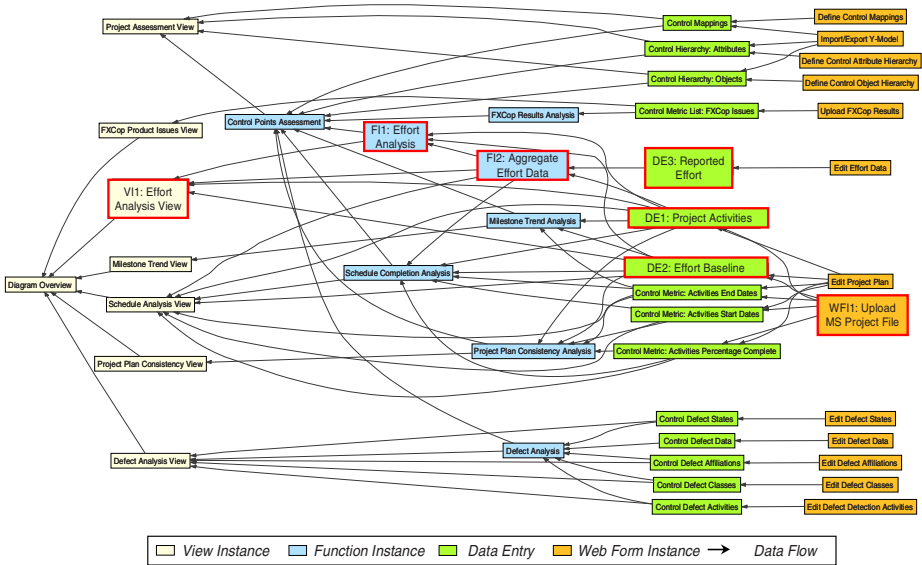


Fig. 4. Example visualization catena. One can see all input and output data of all control components used for constructing the VC. 13 web form instances provide input for 15 data entries, which are processed by 8 function instances, and visualized by 8 view instances.

even for a limited number of control components. The excerpts of the VC discussed above are highlighted accordingly.

Phase IV: Execute Project Control Mechanisms: Fig. 5 presents a visualization of the effort controlling view generated by the Specula prototype tool. During the execution of the project, the team members entered their effort data using the corresponding Specula web form. The project manager regularly updated the project plan using MS Project and imported the plan into the SPCC. The quality assurance manager used a static code analysis tool to analyze code quality and imported a corresponding report into the SPCC.

Phase V: Analyze Results: General deviations from the effort baseline were detected including, but not limited to, that the requirements phase took a lot more effort than planned. The project manager updated the project plan accordingly. In addition, if we assume that a negative milestone trend was not detected at all, an important milestone might have been missed.

Phase VI: Package Results: If we assume that the control component for detecting milestone trends used a wrong parameter setting, it will have to be adapted for future use in subsequent projects.

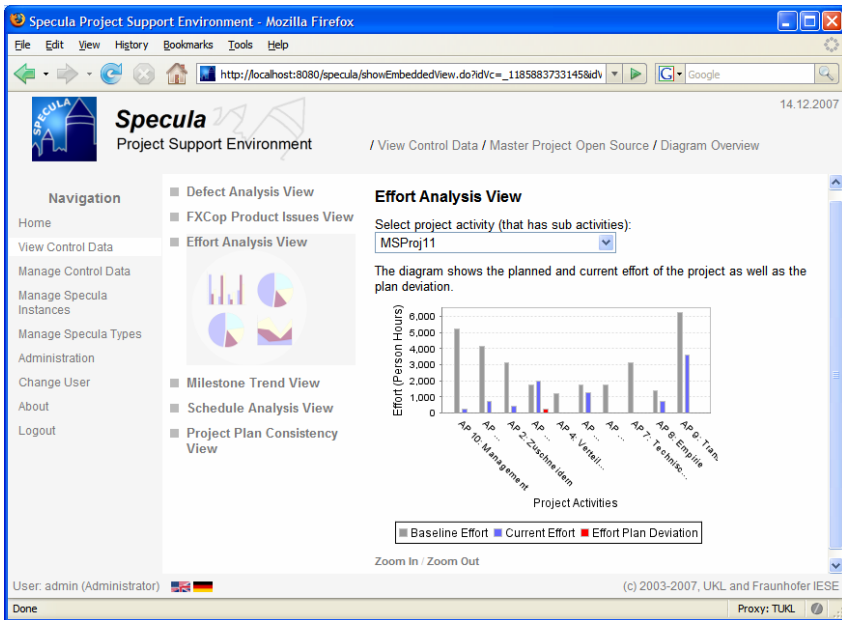


Fig. 5. User interface of the Specula prototype tool. On the left side, one can see the overall navigation bar. The menu close to the navigation bar displays all available views for controlling the project. On the right side, one can see the selected view for analyzing effort data.

5 Conclusion and Future Work

The article presented the Specula controlling approach for setting up a project control mechanism in a systematic and goal-oriented way, profiting from experiences gathered.

Reusable control components were defined and instantiated to illustrate how to define measurement-based project control mechanisms and instantiate them for the software development projects of a concrete organization. A high-level process was shown that provided guidance on how to select the right control components for data collection, interpretation, and visualization based on explicitly defined measurement goals. Moreover, a simple example was presented of how to apply generically defined control components. The Specula approach implements a dynamic approach for project control; that is, measures and indicators are not predetermined and fixed for all projects. They are dynamically derived from measurement goals at the beginning of a development project. Existing control components can be systematically reused across projects or defined newly from scratch. Data is provided in a purpose- and role-oriented way; that is, a certain role sees only measurement data visualizations that are needed to fulfill the specific purpose. Moreover, all project control activities are defined explicitly, are built upon reusable components, and are systematically performed throughout the whole project. A context-specific construction kit is provided, so that elements with a matching interface may be combined. The qualitative benefits of the approach include: being able to identify and reduce risks related to introducing software cockpits, being more efficient in setting up and adapting project controlling mechanisms, allowing for more transparent decision-making regarding project control, reducing the overhead of data collection, increasing data quality, and, finally, achieving projects that are easier to plan and to control.

Further development and evaluation of the approach will take place in the context of the Soft-Pit project. Future work will also concentrate on setting up a holistic control center that integrates more aspects of engineering-style software development (e.g., monitoring of process-product dependencies and linking results to higher-level goals). The starting point for setting up such a control center are usually high-level business goals, from which measurement programs and controlling instruments can be derived systematically. Thus, it would be possible to transparently monitor, assess, and optimize the effects of business strategies performed.

Acknowledgements

This work was supported in part by the German Federal Ministry of Education and Research (Soft-Pit Project, No. 01ISE07A). We would also like to thank Sonnhild Namingha from Fraunhofer IESE for reviewing a first version of this article.

References

1. Agresti, W., Card, D., Church, V.: *Manager's Handbook for Software Development*. SEL 84-101, NASA Goddard Space Flight Center. Greenbelt, Maryland (November 1990)
2. Basili, V.R., Caldiera, G., Rombach, D.: *The Experience Factory*. *Encyclopaedia of Software Engineering* 1, 469–476 (1994)
3. Basili, V.R., Heidrich, J., Lindvall, M., Münch, J., Regardie, M., Rombach, D., Seaman, C., Trendowicz, A.: *GQM+Strategies®: A Comprehensive Methodology for Aligning Business Strategies with Software Measurement*. In: Büren, G., Bundschuh, M., Dumke, R. (eds.) *MetriKon 2007, DASMA-Software-Metrik-Kongress*, Kaiserslautern, Germany, November 15-16, 2007, pp. 253–266 (2007)

4. Ciolkowski, M., Heidrich, J., Münch, J., Simon, F., Radicke, M.: Evaluating Software Project Control Centers in Industrial Environments. In: International Symposium on Empirical Software Engineering and Measurement, ESEM, Madrid, pp. 314–323 (2007)
5. Differding, C.: Adaptive measurement plans for software development. Fraunhofer IRB Verlag, PhD Theses in Experimental Software Engineering, 6 (2001) ISBN: 3-8167-5908-4
6. Heidrich, J.: Custom-made Visualization for Software Project Control. Technical Report 06/2003, Sonderforschungsbereich 501, University of Kaiserslautern (2003)
7. Heidrich, J., Münch, J.: Goal-oriented Data Visualization with Software Project Control Centers. In: Büren, G., Bundschuh, M., Dumke, R. (eds.) MetriKon 2005, DASMA-Software-Metrik-Kongress, Kaiserslautern, Germany, November 15-16, 2005, pp. 65–75 (2005)
8. Heidrich, J., Münch, J., Wickenkamp, A.: Usage Scenarios for Measurement-based Project Control. In: Dekkers, T. (ed.) Proceedings of the 3rd Software Measurement European Forum. Smef 2006, Rome, Italy, May 10-12, 2006, pp. 47–60 (2006)
9. Heidrich, J., Münch, J.: Cost-Efficient Customisation of Software Cockpits by Reusing Configurable Control Components. In: Dekkers, T. (ed.) Proceedings of the 4th Software Measurement European Forum. Smef 2007, Rome, Italy, May 9-11, 2007, pp. 19–32 (2007)
10. Hendrick, R., Kistler, D., Valett, J.: Software Management Environment (SME)— Concepts and Architecture (Revision 1). NASA Goddard Space Flight Center Code 551, Software Engineering Laboratory Series Report SEL-89-103, Greenbelt, MD, USA (1992)
11. ISO 9126: Software Engineering – Product Quality. Technical Report. ISO/IEC TR 9126. Geneva (2003)
12. Kitchenham, B.A.: Software Metrics. Blackwell, Oxford (1995)
13. Krishnamurthy, B., Barghouti, N.S.: Provence: A Process Visualization and Enactment Environment. In: Sommerville, I., Paul, M. (eds.) ESEC 1993. LNCS, vol. 717, pp. 451–465. Springer, Heidelberg (1993)
14. McGarry, J., Card, D., Jones, C., Layman, B., Clark, E., Dean, J., Hall, F.: Practical Software Measurement – Objective Information for Decision Makers, 1st edn. Addison-Wesley Professional, Reading (2001)
15. Münch, J., Heidrich, J.: Software Project Control Centers: Concepts and Approaches. Journal of Systems and Software 70(1), 3–19 (2003)
16. Project Management Institute: A Guide to the Project Management Body of Knowledge (PMBOK Guide) 2000 Edition. Project Management Institute, Four Campus Boulevard, Newtown Square, PA 19073-3299 USA (2000)
17. Selby, R.W., Porter, A.A., Schmidt, D.C., Berney, J.: Metric-Driven Analysis and Feedback Systems for Enabling Empirically Guided Software Development. In: Proceedings of the 13th International Conference on Software Engineering, pp. 288–298 (1991)
18. Simmons, D.B., Ellis, N.C., Fujihara, H., Kuo, W.: Software Measurement – A Visualization Toolkit for Project Control and Process Improvement. Prentice Hall Inc., New Jersey (1998)
19. Tesoriero, R., Zelkowitz, M.V.: The Web Measurement Environment (WebME): A Tool for Combining and Modeling Distributed Data. In: Proceedings of the 22nd Annual Software Engineering Workshop (SEW) (1997)
20. Torii, K., Matsumoto, K., Nakakoji, K., Takada, Y., Takada, S., Shima, K.: Ginger2: An Environment for Computer-Aided Empirical Software Engineering. IEEE Transactions on Software Engineering 25(4), 474–492 (1999)

MIS-PyME Software Measurement Maturity Model- Supporting the Definition of Software Measurement Programs

María Díaz-Ley¹, Félix García², and Mario Piattini²

¹ Sistemas Técnicos de Loterías del Estado (STL)
Gaming Systems Development Department 28234 Madrid, Spain
Maria.diaz@stl.es

² University of Castilla-La Mancha
Alarcos Research Group – Institute of Information Technologies & Systems
Dep. of Information Technologies & Systems – Escuela Superior de Informática
13071 Ciudad Real, Spain
{Felix.Garcia, Mario.Piattini}@uclm.es

Abstract. An important reason why measurement program implementation fails is that the maturity of companies as regards measurement has not been taken into account at its definition phase. Unfortunately, the major methods and frameworks supporting measurement programs –such as Goal Question Metric (GQM), Goal-Driven Software Measurement, GQ(IM), PSM and ISO/IEC 15939– do not explicitly address, this issue, which is especially important in small and medium settings, where low measurement maturity level is typical and there are more measurement implementation constraints. Additionally, these companies usually have poor measurement knowledge, limited resources and budget, which prevent measurement integration in the corporate culture. This restricts measurement support in these companies and increases the chances of failure. In this paper we will be looking at an adaptation of the software measurement maturity method developed by Daskalantonakis. The so-called “MIS-PyME maturity model” is focused on giving support towards measurement program definition and is integrated in MIS-PyME, a methodological framework for measurement suited to small and medium settings.

Keywords: Software measurement maturity model, measurement program definition, success factor, SMEs, MIS-PyME.

1 Introduction

A software measurement program is the result of an initiative meant to define and implement the whole process required to obtain and treat certain software information needs. A successful measurement program is such that becomes a good tool [1], i.e. it directly contributes to solving a part of the engineering problem at hand and generates value rather than data [2]. However, software measurement has proved to be a complex and difficult undertaking in the field of software, especially within the context of small and medium enterprises [3].

In literature one can find that many factors are involved in the successful implementation of measurement programs. As an example, Gopal et al. [4] identified and checked some success factors by analyzing their effects on the success of measurement programs. The success of a measurement program was measured using two variables: use of metrics in decision-making and improved organizational performance. The success factors selected were divided into two groups: organizational and technical factors.

Daskalantonakis also developed a good practice guide based on his experience at Motorola [5, 6]. He gives major importance to the integration of measurement programs with the rest of the software processes of an organization. In addition, he argues that the best people to analyse measurement results are the project managers and engineers involved in the measurement program, since they are experts in that particular field and understand perfectly the meaning of that data.

Fenton et Hall [7] identified fifteen success factors based on their experience, which are as follows: incremental implementation, well planned metrics framework, use of existing metrics materials, involvement of developers during implementation, measurement process transparent to developers, usefulness of metrics data, feedback to developers, ensuring that data is seen to have integrity, and that measurement data is used and seen to be used, securing commitment on the part of project managers, use of automated data collection tools, constantly improving the measurement program, internal metrics champions used to manage the program, use of external metrics gurus and provision of training from practitioners.

In [8] Pfleeger states that it is necessary to link the establishment of a measurement program to the maturity level of an organization. “Metrics are welcome only when they are clearly needed and easy to collect and understand.” As an example, a measurement immature organization should not intend to implement a predictive model. This may lead to results that are unexpectedly negative, positive but spurious, difficult to interpret, or difficult to build on in subsequent studies [9]. Also, measurement cannot exceed software process: if the development process does not define the types of tests, it is not possible to evaluate the efficiency of some tests as regards others.

In this paper we look at how this last success factor is integrated in MIS-PyME, a methodological framework for defining software measurement programs focused on small and medium enterprises (SMEs) or settings. We describe an adaptation of Daskalantonakis’[6] software measurement maturity method and the interface for integrating this model into MIS-PyME in order to support it for the purpose of defining measurement programs adapted to the measurement maturity of each company.

This paper is organized as follows: Section 2 brings this work into context by summarizing existing software measurement maturity models. Section 3 introduces MIS-PyME. Section 4 describes MIS-PyME measurement maturity module. Section 5 gives an example of a real-life application for this module and underlines its advantages, and Section 6 sums up the content of this paper and outlines future research.

2 Related Work

In this section the major measurement maturity methods and models found in literature are summarized. We start with Daskalantonakis’ [6] method for assessing an

organization's software measurement technology which is consistent with the SEI Software process assessment methodology [10]. This method is based on a number of assumptions which determine the focus of the Measurement Technology Assessment. From these assumptions, ten themes are derived according to which the company is characterized and evaluated:

1. Formalization of the development process
2. Formalization of the measurement process
3. Scope of measurement within the organization
4. Implementation support for formally capturing and analyzing knowledge
5. Measurement evolution within the organization
6. Measurement support for management control of software projects
7. Project improvement using measurement technology
8. Product improvement using measurement technology
9. Process improvement using measurement technology
10. Predictability of project, product, and process characteristics

For each theme, five evolutionary stages are defined that a software development organization may follow in order to reach the highest level of maturity for that particular theme. These five evolutionary stages correspond to the five levels of software process maturity as defined by SEI: initial, repeatable, defined, managed and optimized. Some questions have been classified by maturity level in order to perform the assessment.

Niessink and Vliet define a capability maturity model for measurement (M-CMM) as that which can be used to assess the measurement capability of software organizations and to identify ways to improve their measurement capability [11]. The model measures the measurement capability on a five ordinal scale which matches Daskalantonakis' maturity stages. However, Niessink and Vliet define a set of pre-established processes which are different for each level and have to be in place so that an organization can reside on that level. On the other hand, following Daskalantonakis' method, each theme has its own development path.

As regards measurement treatment in software capability maturity models, we must highlight CMM [10] and its successor, CMMI [12], which both include a key process called Measurement and Analysis. This process defines good practices to implement a measurement process in an organization and reach maturity level 2.

In MIS-PyME, the measurement maturity model is used as a support module to help define measurement programs which are adapted to the measurement maturity of the organization. It will not be initially used for organization evaluation purposes. The measurement maturity module will be used as a reference to seek detailed information about a number of measurement aspects, helping the user to decide whether it is convenient or not to implement an indicator for a particular maturity measurement aspect (e.g., can the organization implement the indicator for evaluation purposes?).

Based on this assumption, we found Daskalantonakis' [6] method to be the most suitable for our needs, since the themes defined for assessing maturity mostly match the measurement aspects we want to assess, and because each measurement aspect (theme) has an evolution path organized into different levels, thus allowing the user to adjust its definition depending on what can be achieved.

CMMI [12] deals with most of the measurement aspects. However, they are distributed across most of the key process areas: software project planning at level 2, integrated software management at level 3, quantitative process management at level 4, etc. [13] but it does not deal with this information in a separate module.

Niessink and van Vliet [11] developed their own model to try and evaluate an organization's measurement maturity, and we focus on encouraging the user to define a measurement program which matches the organization's measurement maturity. The key processes defined in this model do not look in sufficient detail at some important measurement capability issues, such as what the company can measure (product, process, project, etc), to what extent (some projects, the whole organization, etc.), their analysis capability (characterizing, evaluating, etc.). This model makes a broader evaluation of measurement processes and does not go into detail as much as would be necessary for users to define their measurement program.

The major models supporting software measurement program definition include: Goal Question Metric (GQM) [14], Goal-Driven Software Measurement GQ(I)M [15], PSM [16] and ISO/IEC 15939 [17]. None of them give explicit support to users in defining measurement programs suitable for their measurement maturity.

3 MIS-PyME

MIS-PyME (Marco metodológico para la definición de Indicadores de Software orientado a PyME) is a methodological framework focused on defining measurement programs based on software indicators in small and medium settings [18].

MIS-PyME framework is classified in three main modules: the methodology and roles (MIS-PyME methodology), the workproducts which give support to the methodology (MIS-PyME Measurement Goals Table, MIS-PyME Indicator Template and MIS-PyME Database) and the third module - the measurement maturity (MIS-PyME Measurement Maturity Model).

MIS-PyME Methodology is based on GQ(I)M [15, 19], but it is designed to define basic indicators which are commonly used and required in most small and medium software development settings. Like GQ(I)M, MIS-PyME is a top-down methodology since it develops a measurement program with the ultimate goal in mind, but restricts actual changes to software process improvement, and may be conditioned by the MIS-PyME table of measurement goals and the indicator templates provided. MIS-PyME work-products are as follows:

- MIS-PyME table of measurement goals: MIS-PyME framework proposes a set of structured measurement goals usually required to implement improvement activities related to software processes.
- MIS-PyME indicator templates: An indicator template is defined for each measurement goal. The indicator template will guide users and help them define indicators and measures for a specific measurement goal. An indicator template shows, among other things, the possibility of implementing the indicator as regards the measurement maturity of the company, the conditions required to successfully implement the indicator regarding previous indicators required, conditions which must be fulfilled in order to successfully implement the indicator and how to integrate this indicator into the software process. The typical questions

which the indicator tries to answer are proposed. Typical outcomes and their related analysis may also be described and show the user what the potential of an indicator is, etc.

- MIS-PyME database: Each MIS-PyME indicator template contains a set of examples of real indicators which have been defined in a successfully implemented measurement program.

One of the objectives of MIS-PyME is to define and implement measurement programs which are adapted to the measurement maturity of the setting. Companies should work in defining and implementing measurement programs which they can successfully implement, rather than trying to obtain the best measure when there are several obstacles that make a successful implementation impossible.

This paper aims to describe the third module which contains MIS-PyME measurement maturity model, and how this model is linked to the indicator templates which are intended as a guide for users.

4 MIS-PyME Measurement Maturity Model

As indicated in the second section, MIS-PyME measurement maturity (MIS-PyME-MM) model is based on Daskalantonakis' [6] method, but modified as follows:

- MIS-PyME model does not only take into account the development process, but also the quality and management processes. Additionally, it deals with the process from the point of view of capability, rather than formalization.
- The scope theme has been deeply specified by indicating what the company is able to measure at each capability level.
- Implementation support does not only take into account measurement support tools, but also the development and management tools required for the company to reach each measurement capability level.
- Some themes specified in Daskalantonakis' [6] method have been unified for the sake of simplicity: "scope", "measurement evolution" and "predictability" have been joined into one, and product, project and process improvement themes have been included in other themes.
- The theme known as "Formalization of the measurement process" has not been included in MIS-PyME measurement maturity model since it is mainly used to evaluate measurement process and not so much to support measurement program definition.
- An interface between MIS-PyME measurement maturity model and the rest of the MIS-PyME framework has been defined in order to give support to the measurement analyst.

MIS-PyME measurement maturity model, which is defined in table 2, will be mainly required during the indicator definition phase. When the measurement analyst defines the indicators, he will be supported by the corresponding MIS-PyME indicator template. This template will make recommendations for measurement maturity (amongst others) in terms of indicator implementation. These recommendations come from the interface of MIS-PyME measurement maturity module.

The interface of MIS-PyME measurement maturity module, which is shown in table 3, 4 and 5, establishes a relationship between the measurement maturity model and MIS-PyME Indicator templates. This interface helps users decide if their measurement maturity is enough for certain values of the indicator field by posing questions based on MIS-PyME measurement maturity model. Therefore, some indicator fields depend on the maturity of the company as regards measurement, especially these fields are those which determine the goal of the indicator and are the following:

Table 1. Indicator template fields which depend on measurement maturity

Indicator Field	MIS-PyME-MM theme	Description
Purpose	Software management, quality and development capability	Measurement process has to fit with the rest of the processes. Otherwise, the implementation of the measurement program will in all probability fail. For example, you cannot measure the effectiveness between test phases if test phases are not well differentiated.
	Measurement scope	There are certain kinds of measures which require a certain degree of measurement maturity and previous experience. As an example, you cannot make reliable predictions on a particular aspect when there has not been any previous, frequent and rigorous measurement of that aspect.
	Tools support	In order to implement some measurement programs, some tools are required such as databases, tools that make it possible to visualize an indicator control panel, etc.
	Measurement support for management issues	Measurement should be established in order to support process improvement goals, which also means management goals. If there is not any purpose in analyzing measurement in terms of decision making or corrective actions, the implementation of a measurement program is not recommended (for example, it is not advisable to implement a measurement program for project monitoring purposes). If the existing measurement data is not used to take simple corrective actions, it is not recommended to do so for other purposes such as optimization.
Entity	Measurement scope	If organizational information is needed based on measurement usually it is previously required to measure projects or products individually. Projects are the first entities to be measured; products comes second and processes third
Focus	Tool support	There are a number of measurements which cannot be performed if certain management or development tools are not in place.
	Processes capability	The aspect to be measured has to be established by the other development, management or quality processes.

- Purpose. This field specifies the intention of the indicator. MIS-PyME suggested values based on [20] which are as follows: characterizing, monitoring, evaluating, predicting and optimizing.
- Entity: This indicator specifies what is to be measured: the process (PROC), the project (PRJ) or the product (PROD).
- Focus: It specifies the aspect or attribute to be measured, a quality attribute (reliability, portability, usability, etc.), process performance (compliance, efficiency), user satisfaction, etc.

Table 1 shows the measurement maturity aspects on which each of the above fields depends.

5 MIS-PyME Measurement Maturity Model - Case Study

In this section we show how MIS-PyME measurement maturity model was applied in an experience which consisted in implementing a measurement program in a medium-sized setting. This experience has given us an idea about the usefulness and benefits of the proposed MIS-PyME maturity model for SMEs.

The measurement program was defined and implemented in the software development and maintenance department of Sistemas Técnicos de Loterías del Estado (STL), which is formed by 39 employees. This company was created by the Spanish Government and provides operational and IT development services for the national lottery.

In 2003, the quality department in STL encouraged an initiative to implement measurement programs in the development and maintenance department in STL but it was not well accepted and implementation was unsuccessful. The director of the development and maintenance department was nonetheless aware of the importance of measurement and was intent on mastering this. Most especially, his objective was to improve management and quality control through these measures. In July 2006, he defined two process improvement goals:

- PIG 1: Improving project and process monitoring and control. He particularly wished to improve the monitoring of the project's progress in comparison with the plan, controlling the tests phases and improving project planning.
- PIG 2: Improving development service and product quality. This goal focused on monitoring and evaluating the development service and product quality.

These process improvement goals comprise five sub-goals, and the indicators shown in figure 1.

We now show two outstanding indicators that were modified to make them be better adapted to the maturity of the company based on MIS-PyME measurement maturity model.

IND-PRJ-FiabImpl indicator aimed to evaluate the reliability of the product developed in order to take corrective actions if necessary. This indicator was necessary for the second process improvement goal (improving development service and product quality). The intention of this indicator is to "evaluate"; the focus is "reliability" and the entity is the "product".

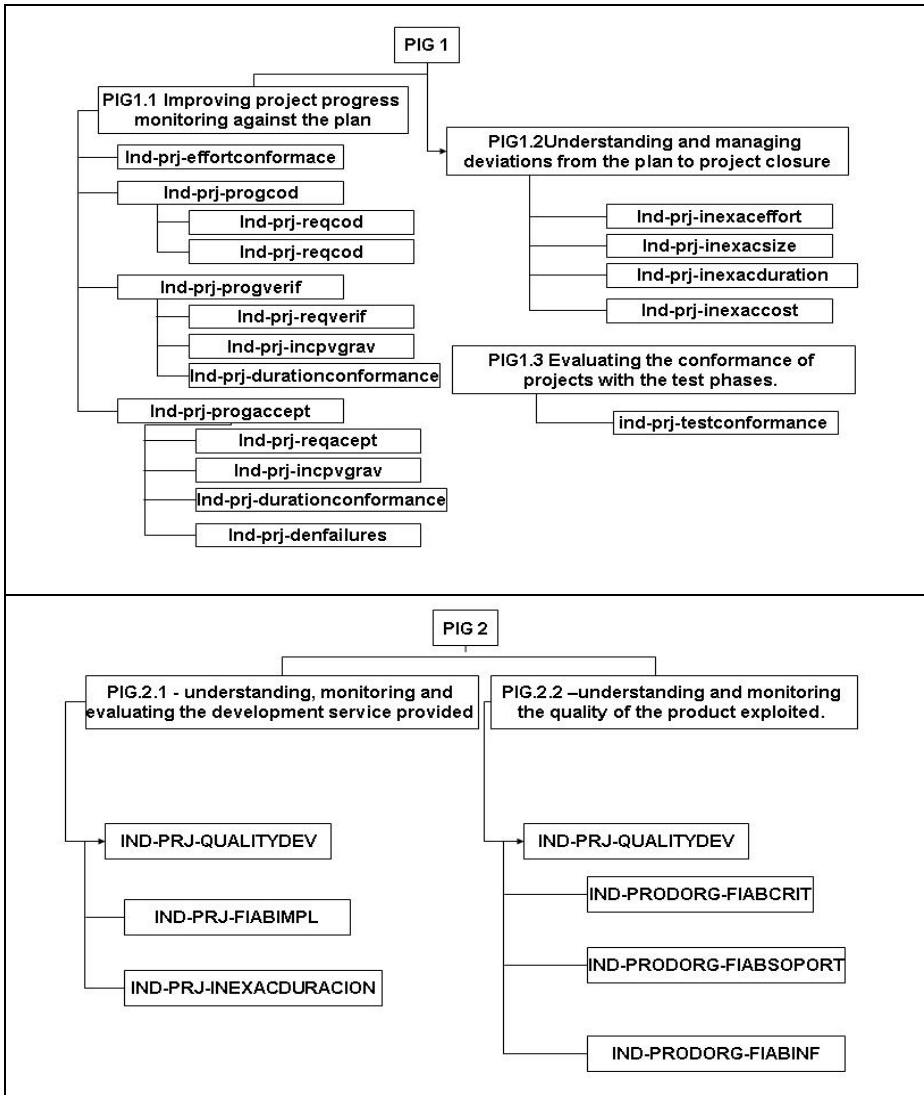


Fig. 1. Measurement Program Definition Implemented in STL

Initially, this indicator evaluated the reliability of the company based on a fix value meant as a threshold (a number of failures registered in production after the product had been installed). However, even if we had experience and we knew (more-less) the reliability of the products in production, and thanks to the suggestions included in this indicator template provided by MIS-PyME which are based on MIS-PyME measurement maturity, we realized that we were not mature enough to state what the reliability of the product would be based on the characteristics of the product developed with a fix goal. The measurement maturity model made us

reflect on this. Focusing on Table 3 and the purpose of “evaluating” the questions are: “do we rigorously, frequently and in an organized fashion measure the reliability of the product and other aspects that may have a relationship with the reliability of the product?” and “could we set reliable goals based on the available data?” Both answers were negative.

We therefore decided to evaluate indicators based on a range of values (good, normal, not too good, not acceptable). In this case we could answer affirmatively to the questions posed: we could define in a reliable way the ranges of reliability of the product developed, which would depend on the type of project: high, medium, low. As can be observed, we descend from level 4 to level 3 in terms of the measurement scope theme. Regarding the measurement support for management issues theme, the top manager was very interested in this indicator, which was included in the project close reports, and the intention was to monitor these data and take corrective actions in case of frequent negative results, therefore the answer of “¿Is measurement going to be used to take corrective actions?” is affirmative.

As regards the focus element of the indicator, the quality (see table 4), we fulfilled maturity measurement requirements. Our management process already provided a close project activity where project managers analyzed the reliability in production of the product developed in addition to other project issues. Regarding “tool support” theme, we had an incident database where failures in production were registered. Most of the people in the company used it and the process was quite well established.

The indicator Ind-PRJ-TestConformance was also modified for it to be better adapted to the maturity of the company. This indicator was defined so as to achieve the first process improvement goal: monitoring conformance with test phases. Initially, this indicator assessed conformance with test phases based on the failures detected during each test phase and compared with a threshold. We were not mature enough to define a threshold for each testing activity, but we were mature enough to define a percentage ratio threshold between test phases (e.g. more than 70% of the failures should be detected during integration test). In both cases, we went on with the fourth measurement maturity level but the second definition was easier and more reliable for us since we were experienced in analyzing the percentage of defects found. Project managers agreed to these modifications and stated that the previous definition of the indicator had not been accurate.

The examples set out in this section illustrate how important it is to define measurement programs which are adapted to the measurement maturity of each company. Even if it seems evident, it is quite easy to make the mistake of trying to define the best measures, even if we cannot implement them. In SMEs it is still easier to make this mistake since resources, budget and measurement culture are limited, and people who define measurement programs may be from inside the company and not too experienced. MIS-PyME indicator templates advise users as to what measurement maturity requirements they should fulfill in order to define the indicator. These advise come from, MIS-PyME measurement maturity model.

Table 2. MIS-PyME. Measurement maturity model based on that developed by Daskalantonakis[6]

Themes	Level 1	Level 2	Level 3	Level 4	Level 5
Software management, quality and development capability	Immature processes. Projects depend on experienced professionals. Project management focus.	Repeat tasks which have been mastered in the past. Project depends on experienced professionals. Project management focus.	Projects characterized and reasonably understood. Project and development system management focus.	Measuring over process and process control. Focus on controlling the process.	Optimized process. Focus on process improvement. Software process improvement, benefits are quantified.
Measurement scope	Carried out occasionally with experienced people or not at all.	Carried out in big projects and with experienced people. Measurement is based on phase-by-phase project tracking, actual against estimate (size, effort, schedule, etc.). Tracking the quality of products in production. There are some estimation mechanisms and some historical data.	Organization establishes standard processes and measurement models which are followed in projects and products. Cross-projects analyses are available. Data collected and analysis are more reliable and consistent. Planning and tracking is often performed at work-package level and still involves actual vs planned performance. Defect quality measures are collected over the developed products. Threshold techniques are used.	Measurement is used in most of the projects and products. Measurement models are also process focused and there is an understanding of the process performance. The measurement process is integrated into the development process. There is a broader view of quality, not just defects but usability, maintainability, flexibility, etc. There is a set of measures that represents a quantitative model of the overall life cycle process.	Well adapted measurement models. Measuring overall process improvement, improving business results and the capability of setting quantitative improvement goals. The organization implements a control panel to keep track of achievements. Improvement can be quantitatively proved.
Tools support	There are no tools to explicitly support	Measurement support tools focused on projects. There are some	Project and product focus measurement tools. There is a measurement database for storing his-	Measurement support tools focused on projects and products. There is an or-	There are organizational tools which automatically col-

Table 2. (continued)

	process measurement.	tools which support estimations. There are incident, cost and planning management tools.	torical data. There is a life cycle configuration management tool for each requirement, models for analysis, etc.	ganizational database where historical data is stored. Data in the database are more reliable and there are actions to prevent recording dirty data. Development of an advanced environment is used which automatically provides product measures.	lect data on the project, product and process and generate a control panel of indicators in order to provide analyses. They also generate automatic reports.
Measurement support for management issues	Management is not supported by measures	Basic project management. Milestones and commitment management. Measurement is used to take reactive decisions during project development, if there are deviations, etc.	The product developed is controlled by means of measures which are used to make decisions regarding the product. Data is used to estimate ranges and thresholds for the project and product. This allows taking corrective actions without the need of re-planning.	It is possible to predict the product, service and other attributes before the product is in production. Usual problems are controlled. It is possible to adapt processes and plans in order to achieve a certain quality degree or other kind of goal.	It is possible to predict and prevent problems. Technological needs and values are known thanks to measurement.

Table 3. MIS-PyME Measurement Maturity Interface as regards Purpose

Themes	Characterizing	Monitoring	Evaluating	Predicting	Optimizing
Software management, quality and development capability	Has the company defined the attributes which are to be measured, even informally?	(Level 3) Have the attributes which are to be measured been included in company processes? Are they correctly understood and used?	(Level 4) Is attribute evaluation included in the process?	(Level 4) Are processes stable enough to be performed rigorously and provide reliable data for the purpose of making estimations?	(Level 5) Is it possible to predict attributes in order to prevent problems and make suitable changes?

Table 3. (continued)

<p>Measurement scope</p>	<p>(Level 1) – No maturity suggestion.</p>	<p>(Level 2) Is any estimating mechanism required to monitor actual vs planned? Is it available? Is data from other projects available? Is the software management process stable enough to include measurement activities? (Level 3) Has the project been monitored phase-by-phase before starting monitoring work-packages?</p>	<p>(Level 4) Does this attribute (and any others related to it) undergo frequent, rigorous and generalized measurement which makes it possible to define a reliable evaluation goal adapted to the characteristics of the organization? (Level 3) Is there sufficient data and business knowledge to define ranges of good, normal, bad, etc. regarding this attribute and the characteristics of the organization?</p>	<p>(Level 4) Has the measurement process been established organizationally? Does this attribute (and any others related to it) undergo frequent, rigorous and generalized measurement which makes it possible to predict it? Is it possible to obtain reliable estimations based on the available data?</p>	<p>(Level 5) Does the organization have a performance control panel of organizational processes and improvement goals? Is the organization qualitatively able to determine if a goal has been achieved? Is it possible to define a reliable improvement plan?</p>
<p>Tools support</p>	<p>(Level 1) – No maturity suggestion.</p>	<p>(Level 2) Are there any tools which provide the required indicators to show project progress? If any attribute product is measured based on defects or failures, is there any incident management tool in the organization?</p>	<p>(Level 4) Is there an organization database to store historical data?</p>	<p>(Level 4) Is there an organization database to store historical data? And estimation mechanisms and tools?</p>	<p>(Level 4) Is there a management tool to dynamically and automatically obtain indicators, reports and estimations in order to make sophisticated analyses?</p>
<p>Measurement support for management issues</p>	<p>(Level 1) What is the intended use of measurement? Is it meant to contribute to decision making and to encourage improvement?</p>	<p>(Level 2) Is it intended to make reactive decisions based on the results of the measurement analysis regarding projects and products in production? (Level 3) – Is it intended to make decisions regarding the developed product based on</p>	<p>(Level 4) Is the organization's purpose to make improvements as required to achieve a goal? (Level 3) Is measurement going to be used to take corrective actions in advance without the need of re-planning?</p>	<p>(Level 4) Is prediction going to be used to improve project planning, avoid future problems, etc? Is prediction going to be used to adapt the planning in order to</p>	<p>(Level 5) What is the intended use of measurement? Is it meant to be used to make dynamic decisions in order to avoid problems and adapt the development, quality and management process? Is it intended</p>

Table 3. (continued)

	measurement results?	achieve a certain goal?	to encourage the improvement of controlled actions to achieve processes and business goals?
ment? Otherwise, what is its purpose?			

Table 4. MIS-PyME Measurement Maturity Interface as regards Focus

Themes	Quality (maintainability, reliability, portability, usability) (Level 2) Is reliability taken into account in process definition? (Level 4) Are maintainability, usability and portability taken into account in these processes? Is there any quantifiable agreement with the user as regards these aspects of the product?	Cost (effort, personnel) (Level 2) Are workers assigned an activity or a task during the project management process? Are any estimates performed for the effort that this will involve?	Project progress, calendar (Level 2) Are projects planned? Do processes specify any monitoring activities for project progress?	Process (compliance, effectiveness) (Level 4) Is the development process well defined? Are development phases well differentiated?	Client satisfaction (Level 2) Does the process define a close collaboration for the project activity where project managers reflect about the development of the project from the point of view of the client? Do they have enough resources and experience to develop this task?
Software management, quality and development capability					
Tools support	(Level 2) Is there any incident management tool in order to obtain data based on defects and failures? Is there any configuration management tool? (Level 4) Is there any development tool where code quality attributes can be obtained such as: cyclomatic complexity, module coupling, inheritance, etc.?	(Level 2) Is there any effort/task management tool which can be used in each project?	(Level 2) Is there any tool to support project planning? (Level 3) Is there any project management tool to control project progress by work-packages?	(Level 2): Is there any incident management tool which makes it possible to obtain data based on defects and failures? And any management tool for life cycle project information?	(Level 2) Is there any incident and change request management tool which makes it possible to obtain data based on defects and failures?

Table 5. MIS-PyME Measurement Maturity Interface as regards Entity and depending on Measurement Scope theme

Project	Product	Process
<p>(Level 3) Before monitoring work packages it is better to start with (Level 2) phase-by-phase monitoring.</p> <p>(Level 3) It is not possible to make cross-project analysis if a measurement model has not been established for the whole organization.</p>	<p>(Level 2-3) Measuring products attributes in production is usually easier, more reliable and important than measuring attributes in development, such as the reliability of products.</p> <p>(Level 3) Usually it is more urgent to proceed with the measurement of projects than with that of products in development, which are used for quality control. Therefore our suggestion is to start measuring projects first and then products in development. (Level 3-4) Usually it is easier and more important to measure products based on defects or failures than to measure effectiveness, reliability, and afterwards other attributes such as friendliness, complexity, maintainability, etc.</p>	<p>(Level 4) Usually, maturity and some experience with projects and products are required to measure the aspects of a process.</p>

6 Conclusions and Further Research

This paper highlights a factor which must be taken into account in order to successfully implement measurement programs, which is defining measurement programs adapted to the measurement maturity of each company.

The paper gives an outline of MIS-PyME measurement maturity model, which is an adaptation of the measurement maturity method developed by Daskalantonakis [6] and the interface defined to integrate this model into MIS-PyME framework. For illustration purposes, two examples are provided and a case study (software measurement program definition in a medium setting) gives an idea of the advantages MIS-PyME measurement maturity model brings with it.

This support module, a measurement maturity framework integrated in the measurement program model for the purpose of defining measurement programs adapted to the measurement maturity of each company, is especially important for SMEs, since usually these companies have poor measurement knowledge and limited resources and budget and people from inside the company, not too experienced in the field, may be those who define the measurement program.

Our future work will revolve around testing and improving MIS-PyME measurement maturity module.

Acknowledgment. We would like to thank the staff of Sistemas Técnicos de Loterías del Estado (STL) for their collaboration. This research has been sponsored by the COMPETISOFT (CYTED, 506AC0287), ESFINGE (Dirección General de Investigación del Ministerio de Educación y Ciencia, TIN2006-15175-C05-05) and INGENIO (Junta de Comunidades de Castilla-La Mancha, PAC08-0154-9262) projects.

References

1. Hughes, R.T.: Expert Judgment as an Estimating Method. *Information and Software Technology*, 67–75 (1996)
2. Niessink, F., Vliet, H.v.: Measurements Should Generate Value, Rather Than Data. In: *Proceedings of the Sixth International Software Metrics Symposium (METRICS 1999)*, Boca Raton (1999)
3. Gresse, C., Punter, T., Anacleto, A.: Software measurement for small and medium enterprises. In: *7th International Conference on Empirical Assessment in Software Engineering (EASE)*, Keele, UK (2003)
4. Gopal, A., et al.: Measurement Programs in Software Development: Determinants of Success. *IEEE Transactions on Software Engineering* 28(9), 863–875 (2002)
5. Daskalantonakis, M.K.: A Practical View of Software Measurement and Implementation Experiences Within Motorola. *IEEE Transactions on Software Engineering* 18(11), 998–1010 (1992)
6. Daskalantonakis, M.K., Yacobellis, R.H., Basili, V.R.: A Method for Assessing Software Measurement Technology. *Quality Engineering*, 27–40 (1990)
7. Hall, T., Fenton, N.: Implementing Effective Software Metrics Programs. *IEEE software* 14(2), 55–65 (1997)
8. Pfleeger, S.L.: Understanding and Improving Technology Transfer in Software Engineering. *Systems and Software* 47 (1999)
9. Briand, L.C., Morasca, S., Basili, V.R.: An Operational Process for Goal-Driven Definition of Measures. *IEEE Transactions on Software Engineering* 28, 1106–1125 (2002)
10. SEI, *The Capability Maturity Model: Guidelines for Improving the Software Process*, Software Engineering Institute (1995)
11. Niessink, F., Vliet, H.V.: *Towards Mature Measurement Programs*. Software Maintenance and Reengineering (1998)
12. CMMI Product Team: *CMMI for Systems Engineering/Software Engineering, Version 1.1 - Staged Representation (CMU/SEI-2002-TR-002, ADA339224)*, Software Engineering Institute, Carnegie Mellon University: Pittsburgh, PA (2002)
13. Weber, C., Layman, B.: Measurement Maturity and the CMM: How measurement Practices Evolve as Processes Mature. *Software Quality Practitioner* 4(3) (2002)
14. Solingen, R.v., Berghout, E.: *The Goal/Question/Metric Method - A practical guide for Quality Improvement of Software Development*. Mc Graw Hill (1999)
15. Park, R.E., Goethert, W.B., Florac, W.A.: *Goal-Driven Software Measurement-A Guidebook*. Carnegie Mellon University Pittsburgh: Software Engineering Institute (1996)
16. PSM: *Practical Software and Systems Measurement - A Foundation for Objective Project Management Version 4.0c*. Department of Defense and US Army (November 2000)
17. ISO/IEC 15939, *Software Engineering-Software Measurement Process*, ISO and IEC, Editors (2002)
18. Diaz-Ley, M., García, F., Piattini, M.: Software Measurement Programs in SMEs - Defining Software Indicators: A methodological framework. In: *PROFES 2007* (2007)
19. Goethert, W., Sivi, J.: Applications of the Indicator Template for Measurement and Analysis. *Software Engineering Measurement and Analysis Initiative* (September 2004)
20. Basili, V.R., Weiss, D.: A Methodology for Collecting Valid Software Engineering Data. *IEEE Transactions on Software Engineering* 10(11), 758–773 (1984)

Predicting Software Metrics at Design Time

Wolfgang Holz¹, Rahul Premraj¹, Thomas Zimmermann², and Andreas Zeller¹

¹ Saarland University, Germany

{holz, premraj, zeller}@st.cs.uni-sb.de

² University of Calgary, Canada

tz@acm

Abstract. How do problem domains impact software features? We mine software code bases to relate problem domains (characterized by imports) to code features such as complexity, size, or quality. The resulting predictors take the specific imports of a component and predict its size, complexity, and quality metrics. In an experiment involving 89 plug-ins of the ECLIPSE project, we found good prediction accuracy for most metrics. Since the predictors rely only on import relationships, and since these are available at design time, our approach allows for early estimation of crucial software metrics.

1 Introduction

Estimating the cost (or size) for a software project is still a huge challenge for project managers—in particular because the estimation typically is done at a stage where only few features of the final product are known. To date, several models have been proposed to improve estimation accuracy [1], but none have performed consistently well. Moreover, although a lot of emphasis is laid upon early estimation of development costs, the parameters used by many models are not known until much later in the development cycle [2]—that is, at a stage when prediction is both trivial and worthless.

In this work, we show how to reliably predict code metrics that serve as inputs (including software size) to prediction models very early on in the project by *learning from existing code*. We leverage the *problem domain of the software* to predict these metrics. The problem domain manifests itself in *import relationships*—that is, how individual components rely on each other’s services. In earlier work, it has been shown that a problem domain, as characterized by imports, impacts the likelihood of software defects [3] or vulnerabilities [4].

Our approach is sketched in Figure 1. We train a learner from pairs of imports and code metrics, as found in existing software. The resulting predictor takes the set of imports for a component (as available in the design phase) and predicts metrics for the component. Managers can then use the predicted metrics as a basis to make other decisions, such as: *What will this product cost to develop? How many people should I allocate to the project? Will this product have several defects to be fixed?*

This paper makes the following contributions:

1. We present a novel software size estimation method during the design phase of development.
2. Using the ECLIPSE code base, we show how imports can be used to predict *software size*, given as source lines of code (SLOC) [5].
3. Using the ECLIPSE code base, we show how to predict *software complexity*, as defined by the widely used object-oriented *ckjm* software metrics [6].

We expect that advance reliable knowledge of such product-specific metrics can be a boon to solving several management issues that constantly loom over all types of development projects at an early stage.

This paper is organized as follows. In Section 2, we discuss features and shortcomings of contemporary cost estimation models. The data used for our experimentation is elaborated upon in Section 3. Thereafter, we present our experimental setup in Section 4, which is followed by results and discussions in Section 5. Threats to validity are addressed in Section 6 and lastly, we conclude our work in Section 7.

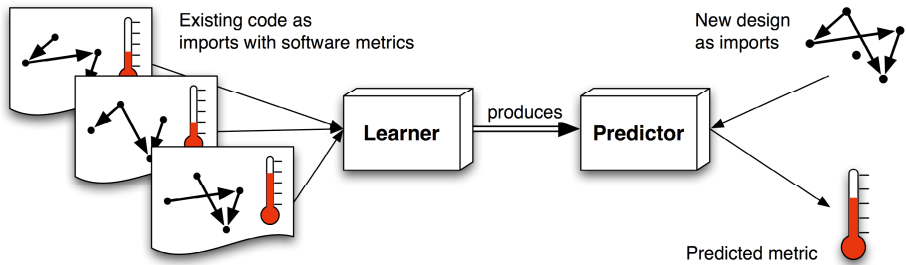


Fig. 1. Approach overview. By learning from the relationship between imports and metrics in existing code, we can predict metrics based on imports alone.

2 Background

As discussed above, cost estimation is vital to a successful outcome of a software project. But most contemporary estimation models depend upon characteristics of the software that are typically unknown at start. For example, many models take into account the relationship between software size and cost. Examples include algorithmic models such as COCOMO [7] and Putnam [8], regression models [9] and analogy-based models [10–13]. To use these models, first an estimate of the size of the project is needed. Again, size is unknown at start of the project and can only be estimated based on other characteristics of the software. Hence, basing cost estimates on an estimate of size adds to uncertainty of the estimates and fate of the project. This challenges the value of such models.

We propose a novel approach that, in contrast to others, focusses on estimating the size of a component as little knowledge as its design. This places managers at a unique position from where they can choose between several alternatives to optimize not only size, but also other metrics of the software that serve as its quality indicators. We present these metrics in more detail in the following section.

3 Data Collection

We used 89 core plug-ins from the ECLIPSE project as data source for our study. Core plug-ins are those that are installed by default in ECLIPSE. We used both source code and binaries to extract the data necessary to build prediction models. In this section, we describe the metrics extracted and the methods employed for their extraction. The metrics or features can be grouped into two categories; first, *input features*, i.e., the features that are already known to us, and second, *output features*, which we wish to predict.

3.1 Input Features

As mentioned above, we hypothesize that the domain of the software determines many of its metrics, for instance, defects—a quality indicator. Similar to Schröter et al. [3], we assume that the *import directives* in source code indicate the domain of the software.

Naturally, our first task is to establish the domains of the 89 ECLIPSE plug-ins, i.e., extract all import directives from the relevant code. At first, this task seems trivial because one can quickly glance through JAVA source code to find the import directives at the top of the file.

However, this task becomes very complex when one encounters a situation as illustrated in Figure 2. Here, the import directive in Label 1 contains reference to package `import java.sql.*` instead of classes. Later, in Label 2, objects of classes `Connection` and `Statement` belonging to the `java.sql` package have been instantiated.

It is crucial that such references to packages are resolved to class levels; else we run the risk of leading statistical learning models astray. To accomplish this, we used the Eclipse *ASTParser* [14] that transforms JAVA code into a tree form, where the code is represented as AST nodes (subclasses of *ASTNode*). Each element in JAVA code has an associated, specialised AST node that stores relevant information items. For example, a node *SimpleType* contains the *name*, *return type*, *parameters*, and like. Further information to examine the program structure more deeply is allowed by *bindings*, a provision in *ASTParser*. It is these bindings that resolve import packages into the relevant classes. Figure 2 demonstrates this where the two classes referred to in Label 2 get resolved by the *ASTParser* as `java.sql.Connection` (Label 3) and `java.sql.Statement` (Label 4) respectively.

Using the above method, we extracted 14,185 unique and resolved import statements from the 89 ECLIPSE plug-ins used in this study.

3.2 Output Features

As mentioned earlier, the knowledge of as many product-specific metrics early in the project's life cycle has several advantages. We demonstrate our model's capacity to predict such metrics on a set of commonly known and used in the software development community.

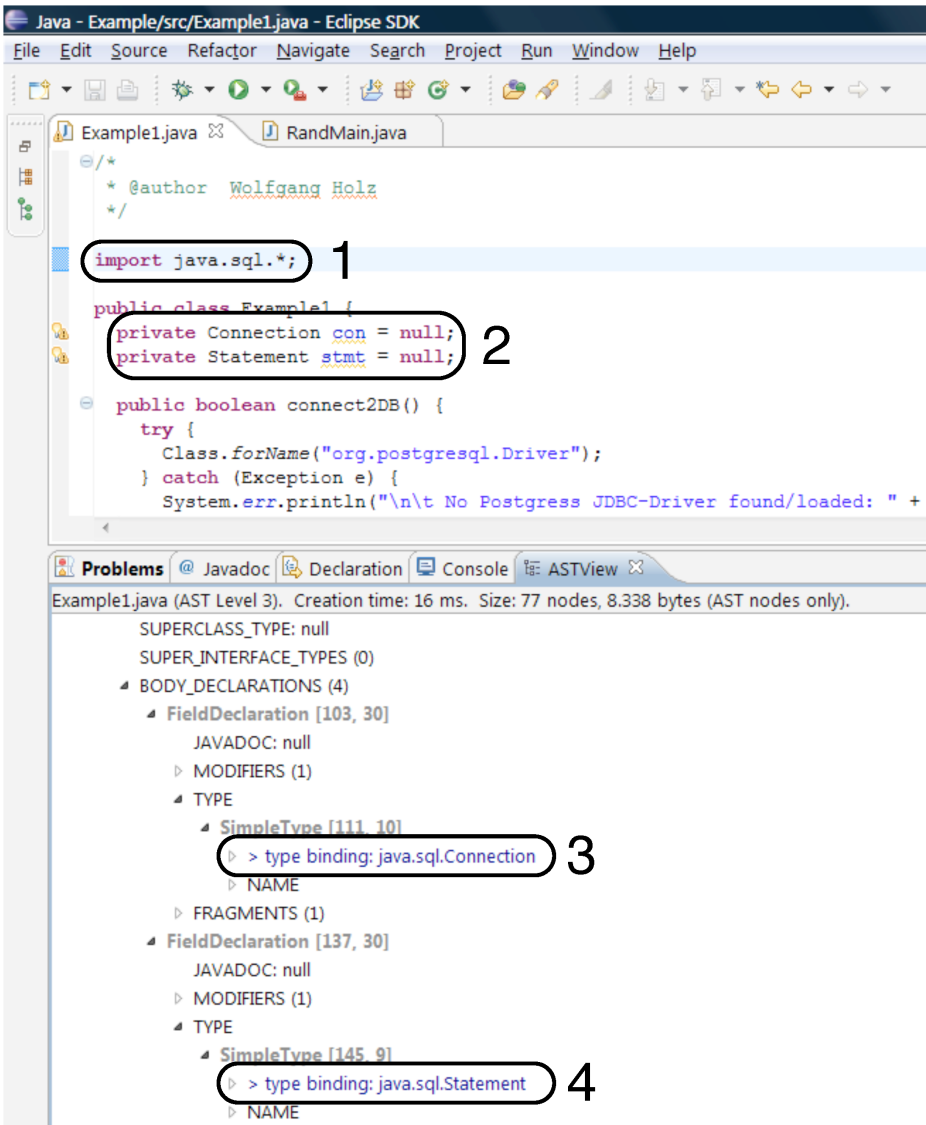


Fig. 2. An illustration of the use of the *ASTParser* to resolve import directives

Source Lines of Code (SLOC). The count of lines of code is the simplest measure of the system size. Early estimate of SLOC or a similar size measure can substantially influence management and execution of the project: development costs and duration of the project can be estimated, system requirements can be inferred, required team size can be appropriated, and like.

Many definitions for counting SLOC have been proposed. We implemented a tool to count SLOC abiding the guidelines laid by Wheeler [5]. As Wheeler recommends, we count the *physical lines of code*, which is defined as follows:

A physical SLOC is a line ending in a new line or end-of-file marker, and which contains at least one non-whitespace non-comment character.

Object-Oriented (OO) Metrics. Our second output feature is a set of OO metrics, referred to as *ckjm* metrics defined by Chidamber and Kemerer [6]. The *ckjm* tool computes six different metrics, summarised in Table 1. These metrics have previously been used to predict fault-proneness of classes [15], changes in short-cycled development projects using agile processes [16], system size [17, 18], and as software quality indicators [19–21].

Table 1. List of *ckjm* Metrics

Abbreviation	Metric
CA	Afferent Couplings
CBO	Coupling between Class Objects
CBOJDK*	Java specific CBO
DIT	Depth of Inheritance Tree
NOC	Number of Children
NPM	Number of Public Methods
LCOM	Lack of Cohesion in Methods
RFC	Response for a Class
WMC	Weighted Methods per Class

* In this metric, Java JDK classes (java.*, javax.*, and others) are included. We created a new metric because the use of JDK classes does not count toward a class's coupling because the classes are relatively stable in comparison to the rest of the project.

While the *ckjm* metrics have been shown to be useful predictors of a variety of software characteristics, a downside of their usage is that substantial parts of the code have to be written to reliably compute them. At this juncture, when code is reasonably mature, the value of such predictions is diminished, i.e., the new knowledge arrives too late in the product's life cycle. Our research alleviates this problem by predicting the *ckjm* metrics for classes at a very early stage of the life cycle. Endowed with predicted values of the *ckjm* metrics, project managers can make further predictions of software characteristics based on these values.

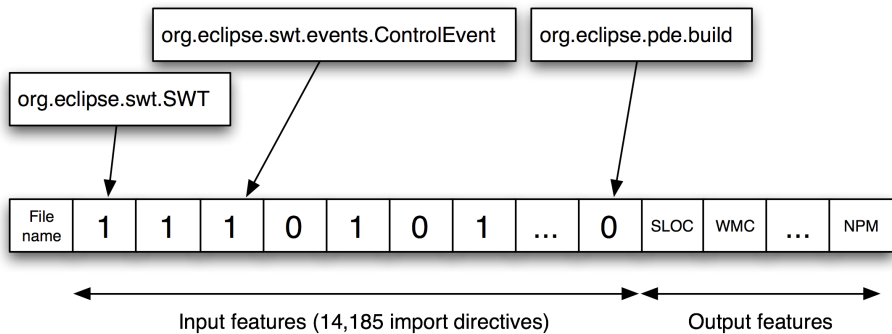
In Table 2, we present some summary statistics of the output features. The values suggest that most metrics are highly skewed. DIT is somewhat flat and most classes have no children (NOC), similar to the finding by Chidamber and Kemerer [6]. In fact, almost 84% of the classes had no children. Particularly noticeable is the fact that many metrics have extreme outliers, for example maximum value of LCOM is 329,563.

3.3 Data Format

After the data has been extracted, we have to shape it as feature vectors to be fed into statistical learning models. Each file is represented as a single row. The input features, that is, the imported classes are represented as dummies. This is illustrated in Figure 3

Table 2. Summary Statistics of Output Features

Metric	Min	Max	Median	Mean	Std. Dev
CA	0	588	2	5.40	5.23
CBO	0	212	9	13.86	16.15
CBOJDK	2	223	15	20.40	18.56
DIT	1	8	1	1.67	1.05
NOC	0	82	0	0.47	2.03
NPM	0	834	4	7.13	13.26
LCOM	0	329,563	9	164.10	3200.28
RFC	0	848	24	39.46	48.96
SLOC	3	7645	72	146.70	273.64
WMC	0	835	7	12.02	18.06

**Fig. 3.** Data format for experimentation

where each of the 14,185 import directives is represented as one column. To indicate that a file imports a class, the value of the respective cell is set to 1, while otherwise it is set to zero. The eleven output features (SLOC and *ckjm* metrics) are represented as columns too alongside the input features. As a result, we have a large matrix with 11,958 rows (files) and 14,196 columns (filename + input features + output features).

4 Experimental Setup

This section elaborates upon the experiments we performed. We begin with describing the prediction model, our training and test sets and lastly, the evaluation for the model performance.

4.1 Support Vector Machine

Support vector machine (SVM) is primarily a supervised classification algorithm that can learn to separate data into two classes by drawing a hyper-plane in-between them. The coordinates of the hyper-plane are determined by ensuring maximum distance between select boundary points of the two classes and the center of the margin. The

boundary points are called *support vectors*. The algorithm uses an implicit mapping of the input data to a high-dimensional feature space where the data points become linearly separable. This mapping is defined by a kernel function, i.e., a function returning the inner product between two points in the suitable feature space.

Recently, SVM has been upgraded to even perform *regression*. This is done by using a different kernel function—the ϵ -insensitive loss function. Basically, this function determines the regression coefficients by ensuring that the estimation errors lie below or equal to a threshold value, ϵ . For more information, we refer the reader to a tutorial on the topic [22].

Besides the kernel function, it is also possible to choose the SVM’s kernel. In a pilot study (predicting SLOC), we found that the *linear* kernel overwhelmingly outperforms other kernels including *polynomial*, *radial bias*, and *sigmoid* when using the evaluation criteria presented in Section 4.3. Hence, we chose to use the same kernel across all our experiments.

4.2 Procedure

The SVM regression model learns using training data instances. For this, we randomly sample 66.67% of the data described in Section 3 to create the training set, while the remaining instances of the data (33.33%) that comprise the test set. Once the model is trained on the training data, it is tested on the test data using only the input features. The output features for the test data are predicted by the model, which are then evaluated using the measures described in Section 4.3.

Additionally, to minimise sample bias, we generate 30 independent samples of training and testing sets, and perform our experiments on each of the 30 pairs.

4.3 Evaluation

We evaluate the results from the prediction model using *PredX*, a popular performance metric used in software engineering. We chose not to use other performance metrics such as *MMRE* because they have been shown to be biased [23]. *PredX* measures the percentage of predictions made that lie within $\pm x\%$ of the actual value. The larger the value of *PredX*, the higher is the prediction accuracy. Typically, x takes the values 25 and 50. We use the same values for our evaluation.

5 Results and Discussion

Figure 4 presents the results from our experiments. All metrics are presented in alphabetical order on the y -axis, while the *PredX* values are plotted on the x -axis. For each metric, we have plotted both, *Pred25* (as circles) and *Pred50* values (as triangles) from each of the thirty experimental runs. The plots are jittered [24], i.e., a small random variation has been introduced to ease observation of overlapping values on the x -axis.

We observe from the figure that SLOC is predicted with reasonable accuracy. *Pred25* values hover around 42% while *Pred50* values hover around 71%. Whereas, prediction results for CBO and CBOJDK are outstandingly good. The *Pred25* values

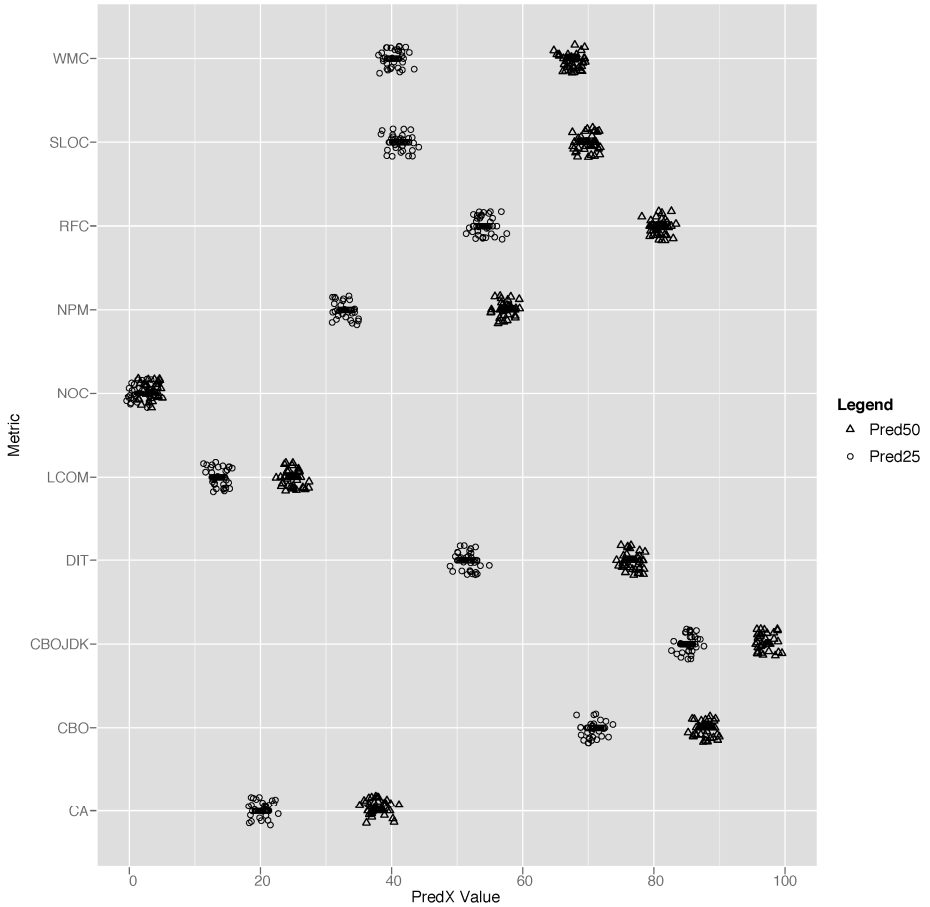


Fig. 4. Prediction accuracy for output metrics

for CBO hover around 72% and even higher for CBOJDK at 86%. Their *Pred50* values hover around 88% and 97% respectively. The model also predicts RFC and DIT values with reasonable accuracy. The values of *Pred25* for both these metrics hover around 51–54%. *Pred50* for DIT hover around 77%, while the same for RFC hovers around 83%.

The prediction accuracy for other metrics, i.e., CA, LCOM, NOC, and NPM is relatively lower. Nearly all *Pred25* and *Pred50* values for most of these metrics are lower than 50%. One metric that markedly stands out is number of children (NOC). This is primarily because of the distribution of the metric. Recall from Table 2 that the median value of NOC is zero and nearly 84% files have no children. This explains the poor results for NOC.

Overall, the prediction accuracy derived from our approach is good for most metrics. It is obvious that early and reliable estimation of SLOC places projects at a vantage point by contributing substantially to their likelihood of success. Our results for

SLOC demonstrate the value of our approach. Perhaps, these can be even topped by using more varied data and other prediction models.

Equally worthy is the approach's capability of predicting code-related metrics as early as during the design phase. Values of many of the metrics could be predicted with high accuracy, up to $Pred50 = 97\%$. The results warrant the use of our approach to facilitate many decisions pertaining to complexity, quality, and maintainability, and allow assessment of alternatives designs. If our results can be replicated in different environments, we anticipate the approach to be valuable support tool for practitioners.

6 Threats to Validity

Although we examined 89 ECLIPSE plug-ins that covered a wide spectrum of domains, from compilers to user-interfaces, we cannot claim with certainty that these plug-ins are representative of all kinds of software projects.

We also approximated the design of plug-ins by its import directives at release time. These relations may have undergone a series of changes from the initial design.

Lastly, we did not filter outliers from our data set. While doing so may improve the prediction accuracy of the models, we chose to preserve the outliers in the data since they make interesting cases to examine and realistically assess the power of our prediction models.

7 Conclusions and Consequences

When it comes to components, you are what you import. As soon as you know which components you will interact with, one can already predict the future size of the component or its complexity. This allows for early estimation and allocation of resources, reducing the risk of low quality or late delivery. Even if the present study limits itself to just one platform (i.e., ECLIPSE plug-ins), the technique can easily be replicated and evaluated on other code bases.

Our approach is easily generalisable to other metrics. Most interesting in this aspect is *cost*: If we know the actual development cost of a component, we can again relate this cost to its domain—and come up with a predictor that directly predicts development cost based on a given set of imports. Instead of development cost, we could also learn from and predict *maintenance costs* or *risk*. We are currently working to acquire appropriate data and look forward to apply our technique on it.

What is it that makes a specific domain impact software metrics? Obviously, the imports we are looking at are just a symptom of some underlying complexity—a complexity we can describe from experience, but which is hard to specify or measure a priori. Why is it that some domains require more code to achieve a particular goal? Is there a way to characterize the features that impact effort? Why do some domain result in more complex code? How do characteristics of imported components impact the features of the importers?

All these questions indicate that there is a lot of potential to not only come up with better predictors, but also to increase our general understanding of what makes software development easy, and what makes it hard. With the present work, we have

shown how to infer such knowledge for specific projects—and hopefully provided a starting point for further, more general, investigations.

In addition, we have made the data set used for this study publicly available for experimentation. The data set can be accessed from the PROMISE repository at

<http://promisedata.org/>

For more information about our research on the prediction of code features visit

<http://www.st.cs.uni-sb.de/softevo/>

Acknowledgments

Many thanks are due to the anonymous PROFES reviewers for their helpful suggestions on an earlier revision of this paper.

References

- [1] Jorgensen, M., Shepperd, M.J.: A systematic review of software development cost estimation studies. *IEEE Transactions on Software Engineering* 33(1), 33–53 (2007)
- [2] Delany, S.J.: The design of a case representation for early software development cost estimation. Master's thesis, Stafford University, U.K. (1998)
- [3] Schröter, A., Zimmermann, T., Zeller, A.: Predicting component failures at design time. In: *Proceedings of the 5th International Symposium on Empirical Software Engineering*, September 2006, pp. 18–27 (2006)
- [4] Neuhaus, S., Zimmermann, T., Holler, C., Zeller, A.: Predicting vulnerable software components. In: *Proceedings of the 14th ACM Conference on Computer and Communications Security* (October 2007)
- [5] Wheeler, D.A.: SLOCCount user's guide (Last accessed 23-11-2007), <http://www.dwheeler.com/sloccount/sloccount.html>
- [6] Chidamber, S.R., Kemerer, C.F.: A metrics suite for object oriented design. *IEEE Transactions on Software Engineering* 20(6), 476–493 (1994)
- [7] Boehm, B.: *Software Engineering Economics*. Prentice-Hall, Englewood Cliffs (1981)
- [8] Putnam, L.H., Myers, W.: *Measures for excellence: reliable software on time, within budget*. Yourdon Press, Englewood Cliffs (1991)
- [9] Mendes, E., Kitchenham, B.A.: Further comparison of cross-company and within-company effort estimation models for web applications. In: *IEEE METRICS*, pp. 348–357. IEEE Computer Society, Los Alamitos (2004)
- [10] Shepperd, M.J., Schofield, C.: Estimating software project effort using analogies. *IEEE Transactions on Software Engineering* 23(11), 736–743 (1997)
- [11] Kirsopp, C., Mendes, E., Premraj, R., Shepperd, M.J.: An empirical analysis of linear adaptation techniques for case-based prediction. In: Ashley, K.D., Bridge, D.G. (eds.) *ICCBR 2003*. LNCS, vol. 2689, pp. 231–245. Springer, Heidelberg (2003)
- [12] Mendes, E., Mosley, N., Counsell, S.: Exploring case-based reasoning for web hypermedia project cost estimation. *International Journal of Web Engineering and Technology* 2(1), 117–143 (2005)
- [13] Mendes, E.: A comparison of techniques for web effort estimation. In: *ESEM*, pp. 334–343. IEEE Computer Society, Los Alamitos (2007)

- [14] Marques, M.: Eclipse AST Parser (Last accessed 14-01-2008), <http://www.ibm.com/developerworks/opensource/library/os-ast/>
- [15] Basili, V., Briand, L., Melo, W.: A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on Software Engineering* 22(10), 751–761 (1996)
- [16] Alshayeb, M., Li, W.: An empirical validation of object-oriented metrics in two different iterative software processes. *IEEE Transactions of Software Engineering* 29(11), 1043–1049 (2003)
- [17] Ronchetti, M., Succi, G., Pedrycz, W., Russo, B.: Early estimation of software size in object-oriented environments: a case study in a CMM level 3 software firm. Technical report, Informatica e Telecomunicazioni, University of Trento (2004)
- [18] Aggarwal, K.K., Singh, Y., Kaur, A., Malhotra, R.: Empirical study of object-oriented metrics. *Journal of Object Technology* 5(8) (2006)
- [19] Subramanyam, R., Krishnan, M.: Empirical analysis of ck metrics for object-oriented design complexity: Implications for software defects. *IEEE Transactions on Software Engineering* 29(4), 297–310 (2003)
- [20] Andersson, M., Vestergren, P.: Object-oriented design quality metrics. Master's thesis, Uppsala University, Uppsala, Sweden (June 2004)
- [21] Thwin, M.M.T., Quah, T.S.: Application of neural networks for software quality prediction using object-oriented metrics. *Journal of Systems and Software* 76(2), 147–156 (2005)
- [22] Smola, A.J., Schölkopf, B.: A tutorial on support vector regression. *Statistics and Computing* 14, 199–222 (2004)
- [23] Foss, T., Stensrud, E., Kitchenham, B., Myrveit, I.: A simulation study of the model evaluation criterion MMRE. *IEEE Transactions on Software Engineering* 29(11), 985–995 (2003)
- [24] Chambers, J.M., Cleveland, W.S., Kleiner, B., Tukey, P.A.: *Graphical Methods for Data Analysis*. Wadsworth (1983)

A Metrics Suite for Measuring Quality Characteristics of JavaBeans Components

Hironori Washizaki, Hiroki Hiraguchi, and Yoshiaki Fukazawa

Waseda University, 3-4-1 Okubo, Shinjuku-ku, Tokyo, Japan
{washi, h_hira, fukazawa}@fuka.info.waseda.ac.jp

Abstract. In component-based software development, it is necessary to measure the quality of components before they are built into the system in order to ensure the high quality of the entire system. However, in application development with component reuse, it is difficult to use conventional metrics because the source codes of components cannot be obtained, and these metrics require analysis of source codes. Moreover, conventional techniques do not cover the whole of quality characteristics. In this paper, we propose a suite of metrics for measuring quality of JavaBeans components based on limited information that can be obtained from the outside of components without any source codes. Our suite consists of 21 metrics, which are associated with quality characteristics based on the ISO9126 quality model. Our suite utilizes the qualitative evaluation data available on WWW to empirically identify effective metrics, and to derive a reference value (threshold) for each metric. As a result of evaluation experiments, it is found our suite can be used to effectively identify black-box components with high quality. Moreover we confirmed that our suite can form a systematic framework for component quality metrics that includes conventional metrics and newly defined metrics.

1 Introduction

Component-based software development (CBD) has become widely accepted as a cost-effective approach to software development, as it emphasizes the design and construction of software systems using reusable components[1]. In this paper, we use object-oriented (OO) programming language for the implementation of components. CBD does not always have to be object-oriented; however, it has been indicated that using OO paradigm/language is a natural way to model and implement components[2].

Low-quality individual components will result in an overall software package of low quality. It is therefore important to have product metrics for measuring the quality of component units. A variety of product metrics have been proposed for components [3,4,5,6]; however, nobody has so far reported on the results of a comprehensive investigation of quality characteristics.

In this paper we propose a suite of metrics that provide a comprehensive mechanism for judging the quality characteristics of high-quality black-box components, chiefly from the viewpoint of users.

2 Component-Based Development and JavaBeans

A component is a replaceable/reusable software unit that provides a certain function. Components are generally implemented in an object-oriented programming language. Component-based development is a method for determining the software architecture (component architecture) that forms a development platform, reusing executable components or developing new components according to the architecture standard, and combining the resulting components to develop new software.

With the appearance of comprehensive development environments based on a visual component assembly metaphor and the popularization of environments for implementing web applications (JSP, ASP, etc.), client components have already become popular way of implementing items such as GUI components and general-purpose logic components[8]. Therefore this paper is concerned with JavaBeans components[10] as the subject of quality measurements.

2.1 JavaBeans Technology

JavaBeans is a component architecture for developing and using local components in the Java language. A JavaBeans component ("bean") is defined as a single class in the Java language that satisfies the two conditions listed below. Accordingly, a bean has constructors, fields and methods, which are the constituent elements of ordinary classes.

- It has an externally accessible default constructor that does not take any arguments, and can be instantiated simply by specifying the class name.
- It includes a `java.io.Serializable` interface and is capable of being serialized.

Figure 1 shows the UML class diagram of an example of a bean. In this example, the `Chart` class is a bean according to this definition. In JavaBeans, in addition to the above mentioned definition, it is recommended that the target class and associated classes conform to the following mechanism to make it easier for them to be handled by development environments and other beans:

- Properties: A property is a named characteristic whose value can be set and/or got (read) from outside the bean. In target classes that are handled as beans, a property is defined by implementing a setting method that allows the value of a characteristic to be externally set, and a getting method that allows the value of a characteristic to be externally read. Methods of both types are called property access methods. Property access methods are chiefly implemented according to the naming rules and the method typing. When the target class has a `getXyz()` method that returns a value of type `A` (or a `setXyz()` method that requires a argument value of type `A`), then it can be inferred that the class has a writable (or readable) property `xyz`. Most of a bean's properties tend to have a one-to-one correspondence to the fields implemented in the bean class[5]. In the example shown in Fig 1, the `Chart` class has the methods `setTitle` and `getTitle()` for setting and

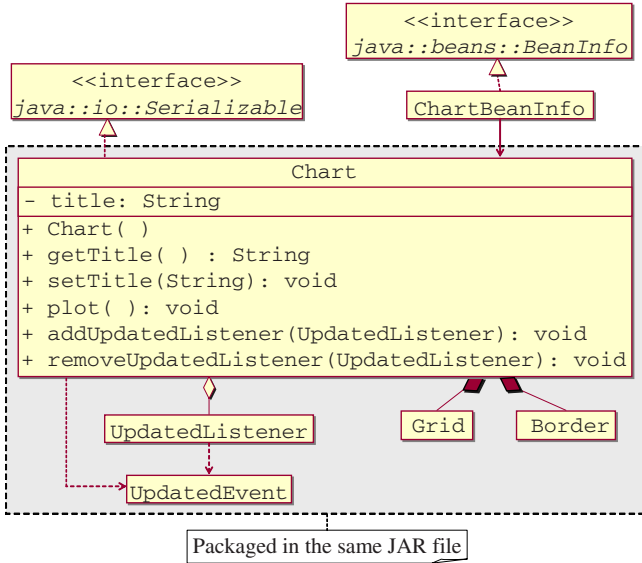


Fig. 1. Example of a bean and its associated classes (UML class diagram)

getting the value of a `title` field. Accordingly, the `Chart` bean has a `title` property whose value can be set and got.

- **Methods:** A method is a function that is provided for external interaction with a bean. In target classes that are handled as beans, they are defined by implementing **public** methods that can be called externally. In the example of Fig. 1, the `Chart` bean has a `plot()` method.
- **Events:** An event is a mechanism for externally announcing that certain circumstances have arisen inside a bean. The constituent elements of an event are an event source, and event listener, and an event object. In the example of Fig. 1, the `Chart` bean has an `Updated` event.

The above mentioned definitions and mechanisms do not guarantee that it will exist in an environment where the other classes and/or interfaces on which the bean depends are present when the bean is independently distributed. Therefore, in JavaBeans it is recommended that a JAR archive file is used to store all the Java classes and interfaces on which the bean depends in the same archive file for distribution and reuse. In the example of Fig. 1, the `Grid` and `Border` classes and event-related classes and interfaces on which the `Chart` bean depends must be distributed by storing them all together in a single JAR file.

2.2 JavaBeans Public Information

Components are not only reused within organizations to which the components' developers belong, but are also distributed in the form of an object code via the Internet and reused in other environments [9]. Therefore, users who want to

reuse components often cannot obtain source codes of the components except for object codes. To allow a bean to be reused as a black-box component while keeping all its internal details hidden, the following information can be obtained externally without having to analyze the source code.

- Basic bean information: An introspection mechanism [10] can be used to obtain information about the properties, events and methods of the above mentioned mechanism. This information is obtained either based on the naming rules, or by analyzing a BeanInfo object provided by the bean developer.
- Class constituent information: Information relating to the constructors, fields and methods of the bean as a class can be obtained by using a reflection mechanism.
- Archive file structure information: Information about the structure of the archive file containing the bean (information on the presence or absence of various resource files such as icons, and the constituent elements of other classes/interfaces on which the bean depends) can also be obtained externally without analyzing the source code.

This externally accessible public information is an essential judgment resource for measuring the quality characteristics of a bean.

3 Component Quality Metrics

To evaluate a component when it is reused, the component is assessed from a variety of viewpoints (e.g., maintainability, reusability, and so on) [11]. This necessitates the use of metrics that consistently deal with the overall quality provided by a component rather than a single metric that deals with a single quality characteristic of a component.

Since beans are implemented in Java, it is possible to apply the quality measurements of conventional object-oriented product metrics. However, most conventional metrics perform measurements on entire object-oriented systems consisting of class sets. On the other hand, since components are highly independent entities, it is difficult for these metrics to reflect the component characteristics even when applied to individual component units.

Also, conventional metrics often require analysis of the target source code. Components are sometimes distributed to and reused by third parties across a network, and since in this case they are black-box components whose source code cannot be seen by the user, it is impossible to use conventional white-box metrics [5]. Accordingly, for components whose source code is not exposed, we need measurements that can be applied in a black-box fashion.

In this paper, based on these issues, we use the following procedure to construct a suite of metrics that provide a component's user with useful materials for making judgments when a component is reused.

1. Comprehensive investigation of basic metrics
2. Selection of basic metrics based on qualitative assessment information
3. Construction of a suite of metrics

3.1 Comprehensive Investigation of Basic Metrics

All the information that can be measured from outside a bean is comprehensively investigated as basic metrics. The investigation results are shown in Table 1. Tables 1(a), (b) and (c) show the metrics relating to the bean's information, class structure information, and archive file structure information respectively.

In Table 1(a), "Default event present" expresses whether an initially selected event is pre-specified when a bean that provides multiple events is used in a development environment. Similarly, "Default property present" expresses whether an initially selected property is pre-specified.

In Table 1(b), RCO and RCS are the ratios of property getting methods and setting methods out of all the bean fields, and are used as metrics expressing the extent to which the properties of fields can be publicly got and set [5]. SCCp and SCCr are the ratios of methods that have no arguments or return values, and are used as metrics expressing the independence of the methods [5]. PAD and PAP are the ratios of `public/protected` methods and fields, and are used as metrics expressing the degree to which the methods and fields are encapsulated [12].

In Table 1(c), the notation "Overall M " represents the results of applying metric M under conditions where the constituent elements of all the classes contained in the archive file that includes the bean are assumed to exist within a single class. The number of root classes expresses the number of classes that are direct descendents of `java.lang.Object`. Also, "Overall bean field (method) ratio" expresses the ratio of fields and methods that a bean has in the sum total of fields (methods) in the entire classes.

3.2 Selection of Basic Metrics

Out of all the resulting basic metrics, we select those that are useful for judging the level of quality of the component. For this selection we use manually obtained component evaluation information published at `jars.com` [13]. The evaluation information at `jars.com` has already been used to set the evaluation standard values of a number of metrics [5,6]. At `jars.com`, in-house or independent group of Java capable and experienced individuals review each bean from the viewpoints of presentation, functionality and originality. Finally beans are rated into 8 levels as total of those different viewpoints. These 8 evaluation levels are normalized to the interval $[0, 1]$ (where 1 is best), and the resulting value is defined as the JARS score.

As our evaluation sample, we used all of the 164 beans that had been evaluated at `jars.com` as of March 2004. The publication of beans at `jars.com` means that they are reused in unspecified large numbers, so the JARS score is thought to reflect the height of the overall quality of the component taking the fact that the bean is reused into account. We therefore verified the correlation between the measured values of each bean's basic metrics and its JARS score.

As the verification method, we divided the components into a group with a JARS score of 1 (group A: 117 components) and a group with a JARS score of less than 1 (group B: 47 components), and we applied the basic metrics to all the beans belonging to each group. In cases where testing revealed a difference between the measured value distributions of each group, this basic metric

Table 1. Possible basic metrics relating to: (*A*: normality test result of "good" components group, *B*: that of "poor" group, *T*: difference test result of both distributions)

(a) bean itself		(c) archive file constituent		
Metric <i>M</i>	<i>A</i> <i>B</i> <i>T</i>	Metric <i>M</i>		<i>A</i> <i>B</i> <i>T</i>
BeanInfo present	n n n	Number of files		n n Y
Number of events	n n Y	Class file ratio		n Y n
Number of methods (of bean)	n n Y	Number of icons		n n Y
Number of properties	n n n	Number of classes		n n Y
Default event present	n n n	Number of root classes		n n n
Default property present	n n n	Average depth of class hierarchy (DIT)		Y Y Y
		Abstract class ratio		Y n n
		final class ratio		n n Y
		Interface ratio		n n n
		private class ratio		n n Y
		protected class ratio		n n Y
		public class ratio		n n n
		static member class ratio		n n n
		synchronized class ratio		n n n
		Overall number of fields		n n Y
		Average number of fields per class		n n n
		Overall RCO		n n Y
		Overall RCS		n n Y
		Overall abstract field ratio		n n n
		Overall final field ratio		n Y n
		Overall private field ratio		Y Y n
		Overall protected field ratio		n n n
		Overall public field ratio		n Y n
		Overall static field ratio		n Y n
		Overall transient field ratio		n n n
		Overall volatile field ratio		n n n
		Overall PAD		n n n
		Overall number of constructors		n n Y
		Average number of constructors per class		n Y n
		Overall constructor without arguments (default constructor) ratio		n Y n
		Overall average number of arguments per constructor		Y Y n
		Overall private constructor ratio		n n n
		Overall protected constructor ratio		n n Y
		Overall public constructor ratio		n n n
		Overall number of methods		n Y n
		Average number of methods per class		n n n
		Overall SCCp		n n Y
		Overall SCCr		n n n
		Overall average number of arguments per method		n Y n
		Overall abstract method ratio		n n n
		Overall final method ratio		n n n
		Overall native method ratio		n n n
		Overall private method ratio		Y n n
		Overall protected method ratio		n Y n
		Overall public method ratio		Y Y n
		Overall static method ratio		n Y n
		Overall strictfp method ratio		n n n
		Overall synchronized method ratio		n n n
		Overall PAP		Y Y n
		Overall bean field ratio		n n Y
		Overall bean method ratio		n n Y

was judged to affect the JARS score and was selected as a metric constituting the suite of metrics. Tests were performed for each metric M according to the following procedure.

1. With regard to the distribution of the measured value of M in each group, we tested for normality at a critical probability of 5%. The test results of group A and group B are respectively shown in columns A and B of Table 1. Y indicates that the results were normal, and n indicates that the results were not normal.
2. We tested the differences in the distributions of the measured values in both groups. When both groups were found to be normal, we used Welch's t-test [14] to check whether or not both groups had the same population mean. In other cases, we used the Mann-Whitney U-test [14] to check whether or not the median values of both population distributions were the same. These test results are shown in the T column of Table 1. Y indicates that the distributions were found to be different; i.e. there is a possibility to classify each bean into two groups by using the target metric.

3.3 Construction of Quality Metrics Suite

As a result of these tests, we found differences in the distributions of the measured values between the two groups for 29 metrics. Below, we will consider the association of these measurement test results with quality characteristics in the ISO9126 quality model [7].

- Number of events: Figure 2(a) shows a box-and-whisker plot of the measurement results. This box-and-whisker plot shows the range of the measured values together with the 25%/75% quantiles and the median value for group A (JARS score = 1; left side) and group B (JARS score < 1; right side). The measured values tended to be higher in group A. It seems that beans with a large number of events have sufficient necessary functions and a higher level of suitability.
- Number of methods: According to Fig. 3(a), there tended to be more methods in group A. Beans with a greater number of methods are able to fully perform the required operations and have a greater level of suitability.
- Number of fields: According to Fig. 3(b), the number of fields tends to be smaller in group A. This is thought to be because when using a bean in which the number of fields has been suppressed, the user is unaware of the unnecessary fields, resulting in greater understandability.
- Ratio of **protected** fields: No differences were observed in the distributions of measured values relating to fields with other types of visibility (private/public), so it appears that field visibility does not affect a bean's quality. This metric was therefore excluded from the suite.
- Ratio of **protected** methods: No differences were observed in the distributions of measured values relating to methods with other types of visibility (private/public), so it appears that method visibility does not affect a bean's quality. This metric was therefore excluded from the suite.

¹ Although several problems such as ambiguity have been indicated for the ISO9126 model [15] it can be a good starting point to explore related quality characteristics.

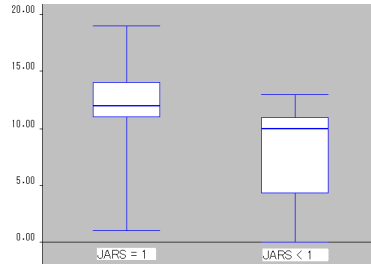


Fig. 2. Number of events (p-value of the null hypothesis=0.0007)

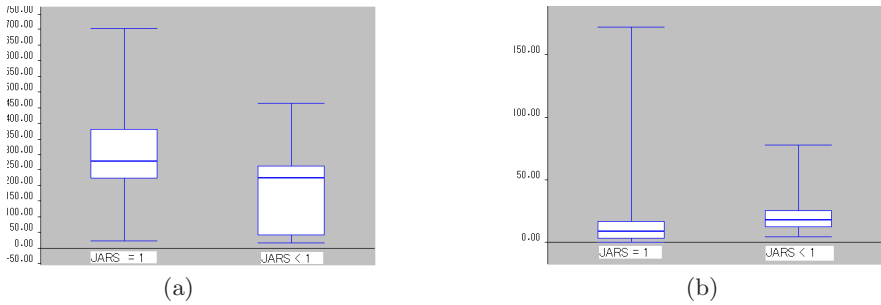


Fig. 3. (a) Number of methods (0.0068) (a) Number of fields (0.0008)

- RCO and RCS: According to Fig. 4(a) and (b), both of these measured values tended to be smaller in group A. By suppressing the number of properties that can be got/set, it is possible to reduce access as properties to more fields than are necessary, which is thought to result in a high level of maturity. Also, since the user is not bothered with unnecessary fields when using the bean, it is thought that the understandability is high.
- Overall RCO and overall RCS: According to Fig. 5(a) and (b), both of these measured values tended to be smaller in group A. Unlike the bean RCO/RCS values, the overall RCO/RCS values are thought to represent the internal maturity and stability of a bean.
- PAD: According to Fig. 6(a), this measured value tended to be smaller in group A. In a bean where this measured value is small, there are few fields that can be operated on without using property access methods, so it is thought that the maturity and changeability are high.
- Number of constructors: According to Fig. 6(b), this measured value tended to be larger in group A. When there is a large number of constructors, it is possible to select a suitable constructor when the class is instantiated, so it is thought that the suitability and testability are high.
- Default constructor ratio: This measured value tended to be smaller in group A. However, since all beans must by definition have a default constructor, this

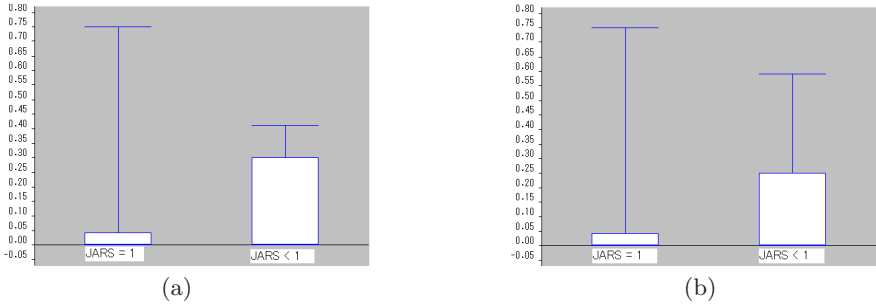


Fig. 4. (a) RCO (0.0671) (b) RCS (0.1096)

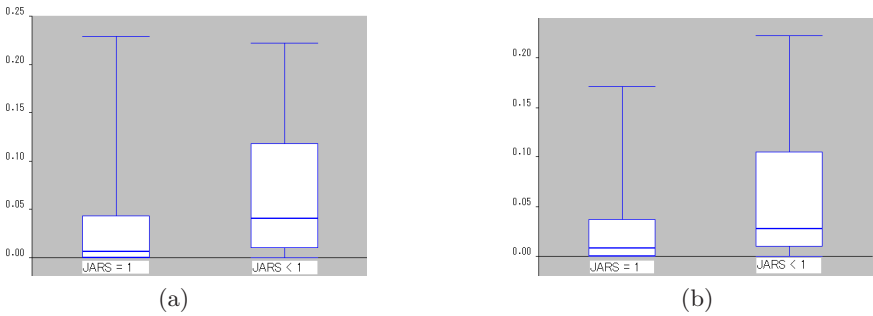


Fig. 5. (a) Overall RCO (0.0061) (b) Overall RCS (0.0071)

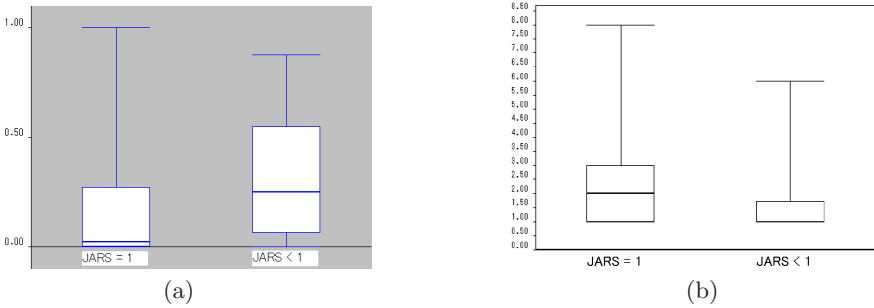


Fig. 6. (a) PAD (0.0042) (b) Number of constructors (0.0031)

metric exhibited the same tendency as the number of constructors. Accordingly, this metric is redundant and is excluded from the suite.

- Average number of arguments per constructor: This measured value tended to be larger in group A. However, since all beans must by definition have a void constructor, this metric exhibited the same tendency as the number of constructors. Accordingly, this metric is excluded from the suite.

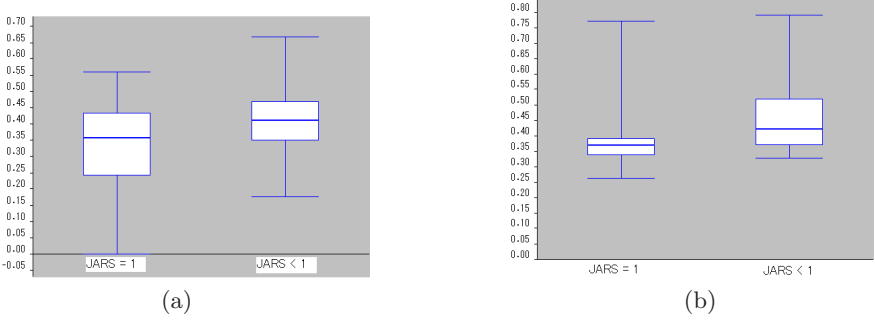


Fig. 7. (a) SCCp (0.0049) (b) Overall SCCp (0.0004)

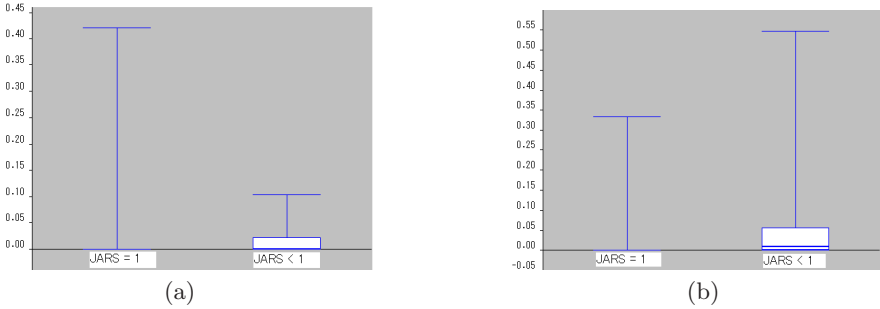


Fig. 8. (a) Static method ratio (0.0857) (b) Synchronized method ratio (0.0083)

- SCCp and overall SCCp: According to Fig. 7(a) and (b), the measured values for both of these metrics tended to be smaller in group A (where there is a higher proportion of methods with no arguments). It is thought that the understandability and analyzability are high because less information has to be prepared at the user side when the methods are used. Here, the overall SCCp differs from the SCCp of individual beans in that there is no redundancy because it relates to the handling of methods inside the bean.
- static method ratio: According to Fig. 8(a), the measured values tended to be smaller in group A. When there are few static methods, the possibility of being operated from various locations without instantiating a bean is reduced, so it is thought that that the analyzability is high.
- Synchronized method ratio: According to Fig. 8(b), this measured value tended to be smaller in group A. When there are few synchronized methods, it is thought that the target bean is set up so that it can be used either in multi-thread or single-thread environments, thus resulting in high analyzability.
- Number of files: According to Fig. 9(a), this measured value tended to be larger in group A. However, since the measured value of the number of files is more or less proportionally related to the number of classes, it is thought that the number

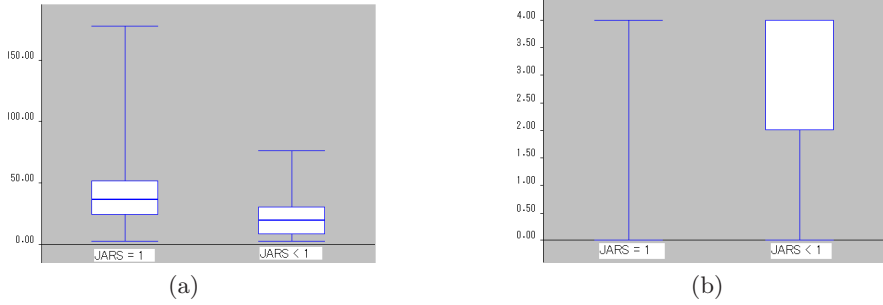


Fig. 9. (a) Number of files (0.0005) (b) Number of icons (0.0641)

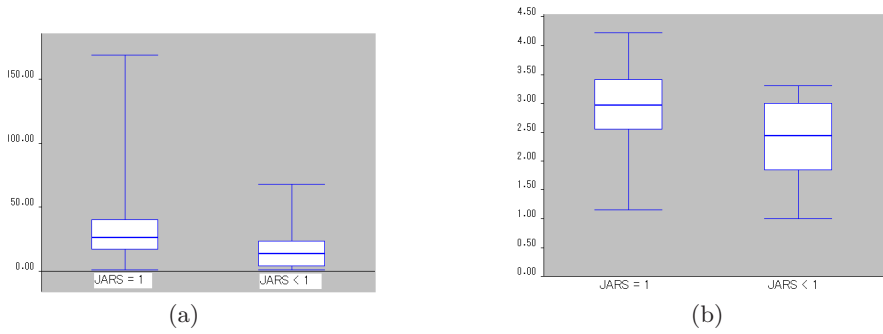


Fig. 10. (a) Number of classes (0.0009) (b) Average depth of class hierarchy (0.0009)

of classes is a more suitable indicator of the scale of a bean. Accordingly, the number of files is excluded from the suite.

- Number of icons: According to Fig. 9(b), this measured value tended to be larger in group A. Icons are information used to represent beans when they are selected in the development environment, and the magnitude of this measured value is thought to reflect the degree of operability.

- Number of classes: According to Fig. 10(a), the number of classes tended to be larger in group A. Looking at the results for other metrics, there is no difference between the distributions of group A and group B in terms of the average number of fields per class and the average number of methods per class, so it is thought that beans with a large number of classes in the archive are not dependent on class sets that are fragmented any more than is necessary, but that they purely express more functions. Therefore, it is thought that beans with more classes have higher suitability.

- Average depth of class hierarchy (DIT [16]): According to Fig. 10(b), this measured value tended to be larger in group A. In object-oriented design, the reuse of fields/methods and the embodiment of variable parts are realized by differential definitions based on inheritance. Therefore, it is thought that the analyzability

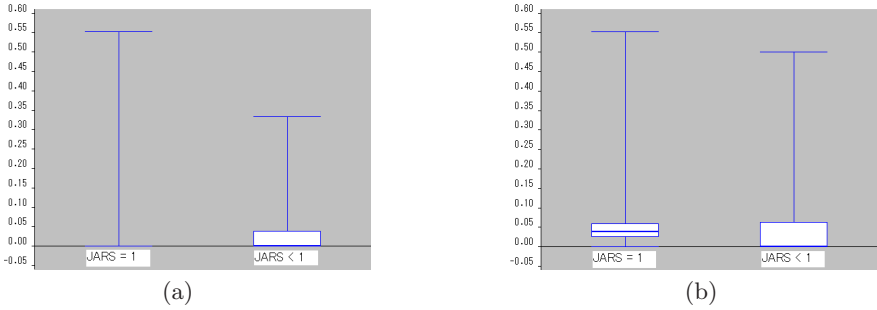


Fig. 11. (a) Final class ratio (0.0585) (b) Private class ratio (0.0074)

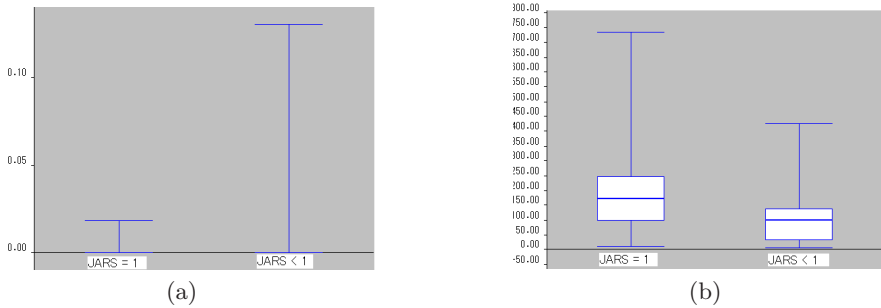


Fig. 12. (a) Protected class ratio (0.2509) (b) Overall number of fields (0.0016)

and changeability both increase with increasing depth of the inheritance hierarchy in the class set that constitutes the archive.

- `final/private/protected` class ratio: According to Fig. 11(a) and (b) and Fig. 12(a), all three of these metrics tended to be smaller in group A. Since these measured values are not encapsulated in a bean any more than is necessary, it is thought that the testability is high.

- Overall number of fields: According to Fig. 12(b), this measured value tended to be larger in group A. Since there is no difference between the two groups in terms of the distribution of the average number of fields per class, it is thought that this measured value increases as the number of classes increases, regardless of how high the quality is. Therefore, this metric is excluded from the suite because it represents the same characteristic as the number of classes.

- Overall number of constructors: This measured value tended to be larger in group A. Since there was no difference between the two groups in terms of the distribution of the average number of constructors per class, it is thought that this measured value increases as the number of classes increases, regardless of how high the quality is. Therefore, this metric is excluded from the suite because it represents the same characteristic as the number of classes.

- Overall `protected` constructor ratio: No differences were observed in the distributions of measured values relating to constructors with other types of visibility (private/public), so it appears that the visibility of constructors in the class set

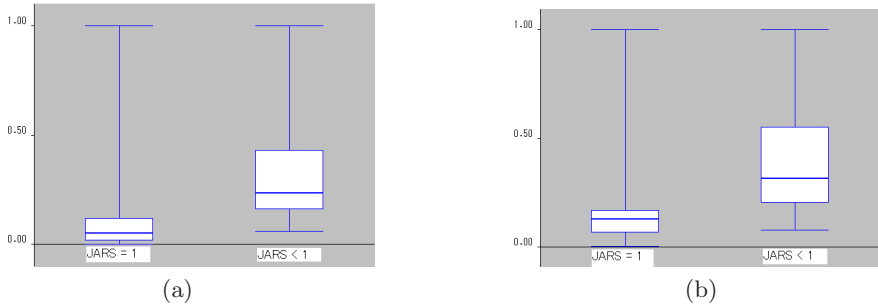


Fig. 13. (a) Overall bean field ratio (0.0000) (b) Overall bean method ratio (0.0000)

constituting an archive does not affect a bean’s quality. This metric is therefore excluded from the suite.

· Overall bean field ratio/bean method ratio: According to Fig. 13(a) and (b), this measured value tended to be smaller in group A for both of these metrics. In beans where these measured values are small, the realization of required functions is transferred to (fields/method in) other dependent class sets while suppressing information that is published externally, so it is thought that the maturity and analyzability are high.

Based on these findings, we selected 21 metrics to be incorporated in the quality metrics suite. According to our consideration of the results, Figure 14 shows a framework for component quality metrics (i.e. the suite of quality metrics) in which these metrics are associated with the quality characteristics mentioned in the ISO9126 quality model. In Fig. 14, metrics that are thought to be effective for measuring the quality of beans are connected by lines to the quality sub-characteristics that are closely related to these metrics in order to show their linked relationships. Of the 21 metrics, 14 metrics obtained results relating to maintainability.

Using this framework, it is possible to make detailed quality measurements focused on beans, and to select metrics that efficiently and comprehensively take account of the overall bean quality.

4 Verifying the Validity of the Metrics Suite

4.1 Standard Assessment Criteria

As a threshold value for deciding whether a target bean belongs in either group A or group B of the previous section, we obtained standard assessment criteria for each basic metric. Using these standard assessment criteria, we verified whether or not it is possible to judge the quality of a bean. If $\overline{X}_{M,a}$ and $\overline{X}_{M,b}$ are the average values of metric M in group A and group B respectively, then the standard assessment criterion E_M is defined as follows:

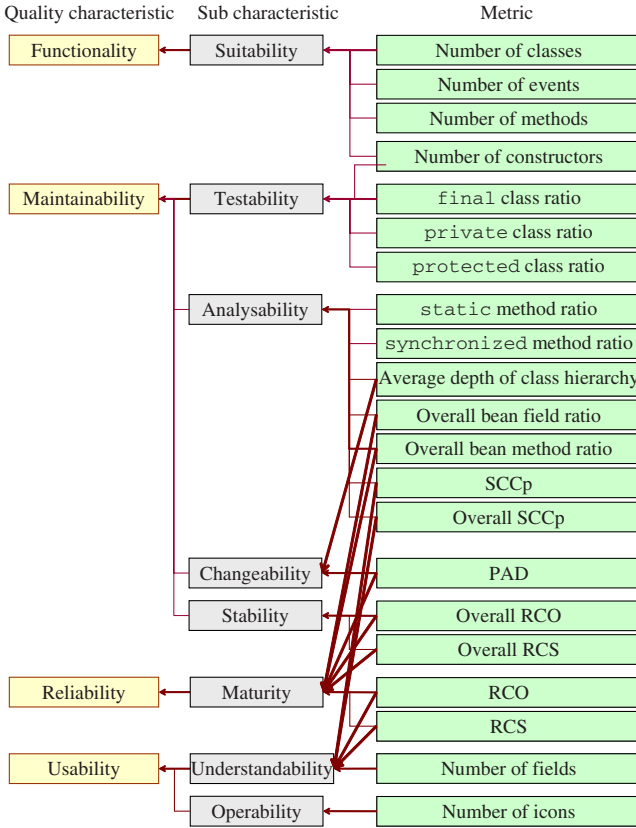


Fig. 14. A framework of component quality metrics

$$E_M = \begin{cases} \text{More than or equals to } \frac{\overline{X_{M,a}} + \overline{X_{M,b}}}{2} & (\text{if } \overline{X_{M,a}} > \overline{X_{M,b}}) \\ \text{Less than or equals to } \frac{\overline{X_{M,a}} + \overline{X_{M,b}}}{2} & (\text{Otherwise}) \end{cases}$$

When a measured value corresponds to a standard assessment criterion, the corresponding quality characteristics and/or sub-characteristics of the bean are high. For each of the 21 metrics constituting the proposed metrics suite, Table 2 lists the proportion of beans in group A that correspond to the standard assessment criterion (conformity R_A) and the proportion of beans in group B that do NOT correspond to the standard assessment criterion (conformity R_B). If both of R_A and R_B are close to 100%, the target standard assessment criterion is almost perfectly useful to classify each bean into two groups.

As both degrees of conformity become higher, it shows that the metric is more effective at correctly measuring the quality of the target bean and classifying it into the correct group. According to Table 2, the conformity values are both 50% or more for nine metrics such as SCCp, which shows that these nine metrics are particularly effective at quality measurements. Also, since the overall average

Table 2. Standard assessment criteria and conformity

Metric M	E_M	R_A	R_B
Number of events	≥ 11	77%	57%
Number of methods	≥ 248	60%	61%
Number of icons	≥ 4	99%	26%
Number of classes	≥ 31	45%	83%
Average depth of class hierarchy	≥ 2.6	69%	57%
Final class ratio	$\leq 3\%$	96%	26%
Private class ratio	$\leq 6\%$	79%	30%
Protected class ratio	$\leq 0.8\%$	98%	17%
Overall RCO	$\leq 4\%$	78%	48%
Overall RCS	$\leq 5\%$	78%	39%
Overall SCCp	$\leq 42\%$	91%	52%
Number of fields	≤ 18	76%	48%
RCO	$\leq 9\%$	80%	39%
RCS	$\leq 9\%$	76%	43%
PAD	$\leq 25\%$	74%	52%
Number of constructors	$\geq 2\%$	63%	78%
SCCp	$\geq 66\%$	51%	61%
Static method ratio	$\leq 1.5\%$	84%	35%
Synchronized method ratio	$\leq 4\%$	93%	35%
Overall bean field ratio	$\leq 22\%$	91%	57%
Overall bean method ratio	$\leq 27\%$	87%	52%
Average	–	74%	50%

values for both types of conformity are equal to or over 50%, it is highly likely that the quality of a bean can be suitably assessed by using the combination of multiple metrics constituting the proposed metrics suite.

4.2 Comparison with Conventional Metrics

Metrics suitable for beans in situations where the source code is unavailable include the usability metrics of Hirayama et al. [6], the testability metrics summarized by R. Binder [12], and the object-oriented metrics of Chidamber and Kemerer [16]. Of these conventional metrics, our proposed metrics suite includes all the metrics that can be applied to beans. The contribution of this paper is that it proposes a systematic framework for component quality metrics that includes these existing metrics and newly defined metrics, and that it has been verified using qualitative assessment information.

Metrics for measuring the complexity and reusability of beans have been proposed by Cho et al. [3], but these metrics included the need for analysis of the bean source code. Wang also proposes metrics for measuring the reusability of JavaBeans components [4]; however the metrics indicate the actual reuse rates of the reused component in a component library and cannot be used in a situation where sufficient time has not passed since the target component was developed.

In contrast, our metrics suite can be used in two situations where the source codes are unavailable and where the components were newly developed.

5 Conclusion and Future Work

We have proposed metrics for evaluating the overall quality of individual JavaBeans components in a black-box fashion, and we have empirically confirmed that they are effective based on a correlation with the resulting qualitative assessment information.

In the future, by carrying out manual verification trials, we plan to make a detailed verification of the effectiveness of these proposed metrics, and of the validity of the association between each metric and the quality characteristics. Several metrics that constitute the proposed metrics suite can also be applied to ordinary Java classes that are not beans. We also plan to investigate the possibility of applying them to other classes besides beans.

References

1. Szyperski, C.: *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, Reading (1999)
2. Hopkins, J.: *Component Primer*. *Communications of the ACM* 43(10) (2000)
3. Cho, E., Kim, M., Kim, S.: *Component Metrics to Measure Component Quality*. In: *Proc. 8th Asia-Pacific Software Engineering Conference* (2001)
4. Wand, A.J.A.: *Reuse Metrics and Assessment in Component-Based Development*. In: *Proc. 6th IASTED International Conference on Software Engineering and Applications* (2002)
5. Washizaki, H., et al.: *A Metrics Suite for Measuring Reusability of Software Components*. In: *Proc. 9th IEEE International Symposium on Software Metrics* (2003)
6. Hirayama, M., Sato, M.: *Usability evaluation of software components*. *IPSJ Journal* 45(6) (2004)
7. *ISO/IEC 9126 International Standard: Quality Characteristics and Guidelines for Their Use* (1991)
8. Suzuki, M., Maruyama, K., Aoki, T., Washizaki, H., Aoyama, M.: *A Research Study on Realization of Componentware Technology*, Research Institute of Software Engineering (2003)
9. Aoyama, M., et al.: *Software Commerce Broker over the Internet*. In: *Proc. 22nd IEEE Annual International Computer Software and Applications Conference* (1998)
10. Hamilton, G.: *JavaBeans 1.01 Specification*, Sun Microsystems (1997)
11. Sedigh-Ali, S., et al.: *Software Engineering Metrics for COTS-Based Systems*, *Computer*, vol. 34(5) (2001)
12. Binder, R.: *Design for Testability in Object-Oriented Systems*. *Communications of the ACM* 37(9) (1994)
13. JARS.COM: *Java Applet Rating Service*, <http://www.jars.com/>
14. Glass, G.V., Hopkins, K.D.: *Statistical Methods in Education and Psychology*. Allyn & Bacon, MA (1996)
15. Al-Kilidar, H., et al.: *The use and usefulness of the ISO/IEC 9126 quality standard*. In: *Proc. 4th International Symposium on Empirical Software Engineering* (2005)
16. Chidamber, S., Kemerer, C.: *A Metrics Suite for Object Oriented Design*. *IEEE Transactions on Software Engineering* 20(6) (1994)

Software Cost Estimation Inhibitors - A Case Study

Ana Magazinovic, Joakim Pernstål, and Peter Öhman

Chalmers, Computer Science and Engineering
SE – 421 96 Gothenburg, Sweden

{ana.magazinovic, pernstal, peter.ohman}@chalmers.se

Abstract. Software cost estimation errors are increasing in the automotive industry along with the number of software components in modern vehicles. As the software cost estimation is an important and problematic part of project planning there is a need of process improvement to decrease estimation errors. However, to improve the process of cost estimation there is a need to explore whether the perceived cost estimation problem is an estimation problem or if it is a management problem. This paper focuses on inhibitors in the process of software cost estimation in system development projects and reports the results of a study carried out at a Swedish automotive company in order to gain an understanding for the factors that affect the cost estimation process.

Keywords: Cost Estimation, Case Study, Empirical Software Engineering, Automotive Systems.

1 Introduction and Related Work

Cost estimation is an important part of the project planning process. The issue was addressed by Lederer and Prasad [1] who found that the average importance rating among the software managers and other professionals participating in their study was *highly important*. Most software organizations find the cost estimation activity to be a necessary part of project planning. According to Heemstra [2] 65% of the organizations estimate projects as a rule, and Lederer and Prasad [1] wrote that 87% of organizations' large projects are estimated.

Just as cost estimation is an important part of project planning, it is also considered a problem. In their paper [3] Moores and Edwards report that 91% of the companies in their study see cost estimation as a problem.

While the figures above are taken from studies concentrating on software industry the amount of software in embedded vehicle systems is increasing and the automotive industry is facing the same problems as the software industry. For the last 30 years the share of electronics and software has grown exponentially in vehicles since the major part of new innovations is realized with software and electronics. According to Grimm [4] and Broy [5] up to 40% of production costs are spent on electronic components and software in premium cars. Today's premium cars contain approximately 70 Electronic Control Units (ECU) controlling infotainment, climate and speed, and new communication networks are added continuously. Further, the new software based systems tend to become more complex which makes cost estimation even more difficult.

With small profit margins in the automotive industry on the one hand and cost overruns in the software industry [1, 6] on the other improved cost estimation process in software development projects is needed to increase cost control. The factors that affect the process of cost estimation need to be explored and understood before a cost estimation method can be proposed and adjusted to the company.

From a research point of view [7], over 50% of the efforts made in the area of Software Cost Estimation research is concentrated on examining the different methods used to estimate effort and cost of software development, such as expert judgment [8, 9], algorithmic models [10] and estimation by analogy [11]. Much of the work has been spent on developing own methods and on history based evolution. However, there are exceptions that point to the need of research concerning organizational issues such as problems created by management and politics. In Laderer and Prasad paper [6] one of the conclusions is that political and management control issues should be given more attention in cost estimation research, and Kitchenham writes in her book [12] that before improving the estimation process, there is a need to explore whether the problem is really an estimation problem and not a management problem.

The studies investigating the causes of cost estimation errors, such as the Phan et al. study involving software professionals from 191 organizations [13], the van Genuchten study involving project managers responsible for cost estimation in six different projects [14], the Lederer and Prasad study with 112 software professionals in different organizations [1, 6] and the Subramanian and Breslawski [15] study involving project managers representing 45 different projects, are mainly based on a set of predefined questions and are regrettably suffering from low response rates.

The purpose of this paper is to revisit the research question in order to explore whether responses would be consistent using qualitative methodology. The results will be compared to the results of the Lederer and Prasad study [1, 6], thus the research question is designed to respond to the second part of the research question presented in the Lederer and Prasad study [1, 6]: “What causes do actually predict inaccurate cost estimates?”:

RQ: What underlying factors affect the process of software cost estimation in system development projects?

To answer the research question an exploratory case study was designed and carried out at Volvo Car Corporation (VCC). Eleven interviews were conducted involving both functional and project managers and other professionals in the Electrical and Electronic Systems Engineering Unit at VCC.

2 Methodology

The qualitative study underlying this paper was conducted as an exploratory single case study as the research question is of exploratory nature and the study focuses on contemporary events, without any possibility to manipulate behavior directly, precisely and systematically.

According to Yin [16], six sources of evidence are most commonly used in case studies, namely documentation, archival records, interviews, direct observations, participant observations and physical events. The interviews were chosen as the primary source of evidence. Direct observation was provided by one of the researchers

who spent a one-week field trip at the company, and participant observation was added by the other researcher working at the company. However, the archival records and documentation (time spent on projects, costs etc) were found to be insufficient for triangulation of results due to unreliable time reports.

To increase the reliability of the results the investigator triangulation was performed. The study was performed by a team of two researchers, present during data collection and data analysis.

2.1 Case Selection

This study took place at the Electrical and Electronics Systems Engineering Unit at Volvo Car Corporation (VCC). Volvo Car Corporation was sold to the Ford Motor Company (FMC) by the Volvo Group in 1999 with FMC as a sole owner. Since then, VCC has produced 450 000 units per year, where safety has been made their trademark.

The selection of interviewees was made with the help of senior researchers with knowledge of the VCC organization and of industry partners working at the company. The selection criteria consisted of work experience, experience of the company, position at the company, and familiarity with external factors affecting the process of cost estimation such as contact with suppliers and the owner of VCC, FMC.

The matrix organization [17] of VCC needed to be considered due to budget division between the functional and the project parts of the organization. The budget for the functional part of the organization includes component costs; development costs are included in the project budget.

The interviews were conducted during the late spring of 2007. Ten of the interviewees had eight to 28 years of experience at VCC; one had worked at the company for 13 months. The majority of the interviewees had been working at the company since the time before FMC took over ownership. All had more than eight years of work experience.

The interviewees were chosen such that they would reflect the organization. Nine were part of the project organization, ranging from concept phase managers to development managers and managers on high and low levels. Two of the interviewees were chosen to represent the functional part of the organization.

2.2 Data Collection

Data were collected using semi-structured interviews as the main source of evidence. Some observational and participatory input was provided by the research team conducting the study. Both of the researchers were present during the interviews. One of the researchers had the role of interviewer and listener, the other wrote down the interview.

2.2.1 Collecting the Background Information

One of the researchers spent one week at the company in order to increase the understanding of the company structure, development processes, jargon and product. The second researcher had eight years of experience in the company, working in the manufacturing department.

2.2.2 Interviews

The interviews held were semi-structured to allow the two-way communication needed to perform this exploratory study. An interview guide was created and adapted to the jargon at the company. It was reviewed by senior researchers to assure the quality of the contents of the questions and by industry representatives in order to assure the understandability of the questions.

The interviewees received an introduction to the study in advance. Sample questions were included.

Six focus areas were developed and used during the interviews to answer the research question:

1. *Cost and time estimation process* deals with the purpose of finding out how the estimation work is done, whether any kind of methodology is used with any templates and process descriptions to be followed, whether the people involved in producing estimates find the methodology, the templates and process descriptions helpful, how the estimates are reported and to whom.
2. *Factors affecting the individual estimates* focuses on the work done by the individual and factors that affect this work. The focus was on finding out whether those who work with estimates discuss their estimates with anyone, how those people affect the estimators' decisions and whether there are other factors involved than calculations made by interviewees or others.
3. *People who have an influence on the estimates* focuses on the groups dealing with the estimates (if such groups exist). The purpose was to find out how the organizational structure affects the estimates, whether there are groups that produce/review the estimates or whether it is one-man job. If such groups exist the focus was to be on exploring the group culture, communication and the priority the estimates were given during their group meetings.
4. *Usage of the estimates* focuses on if and how the estimates are used, and if they are considered to be useful.
5. *Modification of the estimates* focuses on understanding whether and how the estimates are updated, whether there are formal occasions for doing this and, if so, how the estimators perceived those meetings.
6. *Efficiency of the estimation work* focuses on the time that the estimation work takes versus the usefulness of the final estimates, including their correctness.

The interviews were conducted by two researchers to increase the amount of data collected as suggested by Anda and Hove [18]. One had the role of interviewer and listener, and the other wrote down the interview. As issues related to economy are considered to be sensitive [18], the interviews were not recorded. The most sensitive questions were asked late in the interview. The focus was not on the exact figures, taking the edge off the most sensitive questions.

Directly after the interviews, the data collected were reviewed and the interviews were summarized in order to gain as much useful data as possible.

2.3 Data Analysis

To increase the validity of the results the data were analyzed separately by the two researchers. Each researcher made an independent analysis of the data collected. The

independent results were compared to inspect the validity of the results. The validity was found to be sufficient. A list was agreed upon that contained 14 issues.

The results were discussed and reviewed by senior researchers to verify the quality of the work done and its results. A group of industry representatives reviewed the list to verify that the issues were not misunderstood. All the steps were documented to ensure traceability.

3 Analysis

This chapter is outlined as follows. First the results of the underlying case study are presented. The issue abstraction level was chosen to correspond to the abstraction level of the issues reported in the compared study in order to make the second part of the analysis possible, namely the validation of the results found in the Lederer and Prasad study [6].

The issues found in the study underlying this paper not present in the Lederer and Prasad study [6] are also compared to other literature in the last part of the analysis.

3.1 Analysis of the VCC Results

The interviewees, who all belong to the Electrical and Electronics Systems Engineering Unit, state that the understanding of how software is developed is inadequate in the other parts of the company, leading to many of the issues mentioned below. Volvo Car Corporation (VCC) is a company that has, like the rest of automotive industry, relied mainly on mechanical components for many years. The development process at the company has not yet been fully adapted to the increasing development of software and electronic components.

The interviewees stated that they use *expert judgment* approach and to some extent also *analogy* to past projects. Both *top-down* and *bottom-up* [19] estimates are made in order to get the most realistic cost estimate possible when making estimates for new products.

The price per hour for a developer is fixed, regardless of which developer does the work. The price includes overhead costs and is calculated annually for all of VCC. The estimated costs are calculated by multiplying estimated effort (in hours) and the developer cost (per hour).

The list of issues identified contains 14 issues:

1. *There is an error in tracking the actual project costs leading to difficulties when comparing them to the estimated costs.* All the development time spent on a project is not reported by the developers, and is sometimes reported as being done on another project.
2. *The project organization does not exclusively own the relationship to the suppliers leading to conflict of interest.* The interviewees state that, because the project and functional parts of organization have separate budgets including development costs in the project budget and component costs in the functional budget, there is a lack of cooperation between them, making estimation work more difficult.
3. *The satisfaction with time spent on estimates is low* among the interviewees. Some of them find the time spent on estimates to be too low, and wish that more

time could be spent on estimates in order to make them more accurate and detailed. Others believe that the estimates are a waste of time, time that should be spent on technical tasks instead.

4. *Management estimation goals are not taken seriously.* The estimation errors that the interviewees state are considered acceptable do not correspond to management estimation accuracy goals.
5. *Dependencies of other projects are not taken into account when estimating costs.* The staff resources might be moved to other, more critical projects to meet the deadlines.
6. *Unclear requirements* make it difficult to know what is to be developed and estimated. Especially producing early estimates for new technology is found to be difficult.
7. *Changes in requirements are allowed late in the development process.* These are difficult to foresee, making estimation work more difficult.
8. *The padding is removed from the estimates when detected by management.* This is compensated by padding added at a higher level in the organization hierarchy.
9. *Unrealistic estimates are presented at the project milestones in order for the project to continue.* There is a belief that the development work must be done in any case and the estimates presented are sometimes too optimistic.
10. *The estimates are affected by the budget and management goals,* such as cost savings and efficiency demands, leading sometimes to too optimistic estimates.
11. *Different company cultures and estimation techniques lead to difficulties when communicating estimates to FMC representatives.* The cooperation between the companies is aggravated by different cultures in terms of development, estimation methodology, management and politics.
12. *There is a lack of competence in estimating costs for development done by suppliers.* The interviewees also express frustration over lack of competence when reviewing prices proposed by suppliers, as well as when estimating suppliers' abilities to finalize the project successfully. The company has a team that revises cost estimates for hardware components developed by suppliers. However, no such service is provided for software development, making it difficult to estimate the future costs of development work done by the suppliers as well as understand whether prices suggested by suppliers are reasonable or not.
13. *There is no common template for estimates.* Opinions about what template there is for the purpose of cost estimation differ, from no template at all to different kinds of documents that could be used to provide support while making estimates.
14. *There is a lack of estimation competence and the existing estimation competence is not used properly.* The interviewees state that there is in-house development of software components and that it would be preferable to use this competence when making estimates.

3.2 Validation of Lederer and Prasad Top Issues [6]

The Lederer and Prasad study [6] was conducted in the form of a questionnaire, sent out to 400 software professionals and answered by 112, resulting in 16 top factors correlated to estimation inaccuracy.

The issues found in the study underlying this paper were categorized in the same manner as in the Lederer and Prasad paper [6], focusing on four categories: Methodology issues that affect the tuning of the estimate, management issues dealing with project control, user communication issues and politic issues.

Table 1. User communication issues compared to issues found buy Lederer and Prasad [6]

VCC issues	Lederer and Prasad issues [6]
-	L&P 17. Users' lack of understanding of their own requirements.
VCC 7. Changes in requirements are allowed late in the development process.	L&P 14. Frequent request for changes by users
-	L&P 16. Users' lack of data processing understanding
VCC 6. Unclear requirements.	L&P 10. Poor or imprecise problem definition

Two of the Lederer and Prasad user communication issues [6], found in table 1, were validated by the issues found in this study, two could not be validated.

The interviewees participating in this study mention problems in estimation process due to *requests for change late in the development process* (VCC 7) and *unclear requirements* (VCC 7) increasing the uncertainty of the estimate.

Table 2. Methodology issues compared to issues found by Lederer and Prasad [6]

VCC issues	Lederer and Prasad issues [6]
VCC 13. There is no common template for estimates.	L&P 3. Lack of adequate methodology or guidelines for estimating
VCC 1. There is an error in tracking the project costs leading to difficulties while comparing them to the estimated costs.	L&P 12. Inability to tell where past estimates failed
-	L&P 5. Lack of setting and review of standard durations for use in estimating
VCC 14. There is a lack of estimation competence and the existing estimation competence is not used properly.	L&P 13. Insufficient analysis when developing the estimate
VCC 2. The project organization does not exclusively own the relationship to the suppliers leading to conflict of interest.	L&P 4. Lack of coordination of systems development, technical services, operations, data administration etc. functions during the development.

Among the methodology issues found by Lederer and Prasad [6] (table 2) four are validated by the issues found in this study. One of the issues could not be validated, namely L&P 5 *Lack of setting and review of standard durations for use in estimating*.

VCC1, *There is an error in tracking the project costs leading to difficulties while comparing them to the estimated costs*, leads to inability to learn from past mistakes. L&P 12, *Inability to tell where past estimates failed*, addresses this issue as well.

VCC 13, *There is no common template for estimates*, is compared to L&P 3, *Lack of adequate methodology or guidelines for estimating* [6]. The problem, as perceived by the interviewees in the study underlying this paper, is usage of different, more or less official templates, if any. There seems to be no common template, or if there is such a template, there are difficulties with communicating it to the employees.

The interviewees also state that *there is a lack of estimation competence and the existing competence is not used properly* (VCC 14) which corresponds to L&P 13, *Insufficient analysis when developing the estimate*.

VCC 2, *The project organization does not exclusively own the relationship to the suppliers leading to conflict of interest* corresponds to some extent to issue 4, found by Lederer and Prasad [6], *Lack of coordination of systems development, technical services, operations, data administration etc. functions during the development*.

Table 3. Politic issues compared to issues found by Lederer and Prasad [6]

VCC issues	Lederer and Prasad issues [6]
VCC 10. The estimates are affected by the budget and management goals. (VCC 9. Unrealistic estimates are presented at the project milestones, in order for the project to continue.)	L&P 22. Pressures from managers, users or others to increase or reduce the estimate
-	L&P 21. Reduction of project scope or quality to stay within estimate, resulting in extra work later
VCC 8. The padding is removed from the estimates when detected by management.	L&P 20. Removal of padding from estimate by manager
-	L&P 19. Red tape (paperwork)
VCC 11. Different company cultures and estimation techniques lead to difficulties when communicating estimates to the FMC representatives.	-
VCC 12. There is a lack of competence in estimating costs for development done by suppliers.	-

Three of the politic issues (table 3) found by Lederer and Prasad [6] were validated by the issues found in this study, L&P 19, *Red tape*, could not be validated.

VCC 10, *The estimates are affected by the budget and management goals* corresponds to L&P 22, *Pressures from managers, users or others to increase or reduce the estimate*. This, together with removal of padding (VCC 8, L&P 20) leaves less room for risk management.

Two new issues emerged in this study that were not mentioned by Lederer and Prasad [6], namely VCC 11, *Different company cultures and estimation techniques lead to difficulties when communicating estimates to the FMC representatives* and VCC 12, *There is a lack of competence in estimating costs for development done by suppliers*.

Table 4. Management issues compared to issues found by Lederer and Prasad [6]

VCC issues	Lederer and Prasad issues [6]
VCC 1. There is an error in tracking the project costs leading to difficulties while comparing them to the estimated costs.	L&P 18. Performance reviews don't consider whether estimates were met L&P 8. Lack of project control comparing estimates and actual performance
-	L&P 24. Lack of careful examination of the estimate by Information Systems Department management
VCC 3. The satisfaction with time spent on estimates is low.	-
VCC 4. Management estimation goals are not taken seriously.	-
VCC 5. Dependences of other projects are not taken into account when estimating costs.	-

Among the Lederer and Prasad issues [6] in the management category presented in table 4, two are validated by the results of this study, L&P 24, *Lack of careful examination of the estimate by Information Systems Department management*, could not be validated.

VCC 1, *There is an error in tracking the project costs leading to difficulties while comparing them to the estimated costs*, mentioned also among the issues in table 2, leads to L&P 8, *Lack of project control comparing estimates and actual performance* as well as the inability to *consider whether estimates were met* (L&P 18).

Three new issues emerged in this study that did not correspond to the issues found by Lederer and Prasad [6], namely VCC 3, *The satisfaction with time spent on estimates is low*, and VCC 4, *Management estimation goals are not taken seriously*, VCC 5, *Dependences of other projects are not taken into account when estimating costs*.

3.3 Comparison to Other Literature

Five of the issues found in this study are not confirmed by the issues found by Lederer and Prasad [6], see table 5. In order to investigate the validity of those issues, the comparison to other literature was made.

VCC 3, *The satisfaction with time spent on estimates is low*, VCC 4, *Management estimation goals are not taken seriously*, and VCC 5, *Dependences of other projects are not taken into account when estimating costs* addresses the problems described in Van Genuchten paper [14]: *No commitment by personnel to the plan* and *Priority shifts*.

The problematic relationship between vendor and supplier, mentioned in VCC 12, *There is a lack of competence in assessing suppliers' estimates and resources*, has been discussed by many researchers, Bruce et al [20] describe in their paper both risks and negative experiences associated with such collaborations. They mention *more costly and complicated development*, *loss of control ownership* and *differing aims and objectives leading to conflicts*.

VCC 11, *Different company cultures and estimation techniques lead to difficulties when communicating estimates to the FMC representatives*, could not be confirmed by other software engineering literature.

Table 5. VCC issues not reported in the Lederer and Prasad paper [6]

VCC 3: The satisfaction with time spent on estimates is low.

VCC 4: Management estimation goals are not taken seriously.

VCC 5: Dependences of other projects are not taken into account when estimating costs.

VCC 11: Different company cultures and estimation techniques lead to difficulties when communicating estimates to the FMC representatives.

VCC 12: There is a lack of competence in assessing suppliers' estimates and resources.

4 Discussion of Results and Validity

The objective of the study reported here was to elicit the factors that affect the process of software cost estimation in system development projects. The study was performed at Volvo Car Corporation.

The paper focuses on the factors that affect the process of cost estimation negatively. The analysis was divided in tree parts, analysis of the results of the study underlying this paper, validation of the results of a quantitative study [1, 6] with a similar purpose and comparison of the issues found in this study that were not present among the issues in the comparing study [1, 6] to other software engineering literature.

4.1 Discussion of Results

The results of this study were used for validation of top 16 issues reported in the Lederer and Prasad paper [6]. 62,5% of the Lederer and Prasad issues [6] were validated by the results of the study underlying this paper.

Five issues were found in this study were nor reported by Lederer and Prasad [6]. A comparison of those issues to other software engineering literature was made.

Looking at the methodology issues it can be seen that the prerequisites for successfully using the methodologies mentioned by the interviewees in the study underlying this paper are insufficient. Experts should be used to make successful expert judgment. This is contradicted by the VCC 14, *There is a lack of estimation competence and the existing estimation competence is not used properly*. To make analogies to earlier projects, those projects must be documented, both as concerns the results and estimates. However, VCC 1 states that *There is an error in tracking the project costs leading to difficulties when comparing them to the estimated costs*. Not being able to compare estimates with the results makes it impossible to understand which (if any) parts of the estimates were too optimistic or pessimistic. Inadequate time reports by the developers and not reporting which projects they spend time on make it impossible to know how much each project actually costs.

Delays in a certain project might lead to changes in plans for other ongoing or forthcoming projects (VCC 5, *Dependences of other projects are not taken into account when estimating costs*). The delays can be more or less permanent, making the comparison of early estimates and results even harder.

The interviewees state that the development work must be done properly, no matter what the situation, and that the start of production must not be delayed. This attitude, together with management goals for cost savings and greater efficiency, could lead to producing optimistic estimates in order to be able to continue the project. The management goals for increased effectiveness and savings are also believed to lead to unrealistic estimates (see VCC 4, *Management estimation goals are not taken seriously* and VCC 10, *The estimates are affected by the budget and management goals*)

To make estimates as accurate as possible including suppliers' prices, the functional and project parts of the organization must be able to cooperate. However, cooperation is complicated by their budgets being separated which might lead to competition instead of cooperation (VCC 2, *The project organization does not exclusively own the relationship to the suppliers leading to conflict of interest*).

Among the politic issues (table 3) there is a connection that points toward a sense of frustration over the way the estimates are handled. The management and budget goals (VCC 10, *The estimates are affected by the budget and management goals*) lead to reducing the estimates and removing padding (VCC 8, *The padding is removed from the estimates when detected by management*). There also seems to be frustration over unsatisfactory communication with suppliers and FMC representatives.

VCC 5, *Unclear requirements*, (user communication issues, table 1) is an important issue in terms of estimates. Not knowing at an early stage what is to be developed is tantamount to not knowing what is to be estimated and paid for. *Allowing changes in requirements late in the development process* (VCC 7) adds to the uncertainty in early estimates.

The differences between the results reported by Lederer and Prasad [6] and the results reported in this paper could depend on several reasons. VCC is a company with a long history of primarily mechanical development with development processes not fully adapted to increasing amount of software components. The participants in the Lederer and Prasad are spread in several different industries.

Also, issues VCC 5, *Dependences of other projects are not taken into account when estimating costs*, VCC 11, *Different company cultures and estimation techniques lead to difficulties when communicating estimates to the FMC representatives* and VCC 12, *There is a lack of competence in assessing suppliers' estimates and resources* point at the fact that estimation of effort in a project can not be viewed as an isolated event in a multi project environment with stakeholders outside the organization in question.

4.2 Validity Discussion

To increase *construct validity* investigator triangulation (two evaluators) was performed. A team of two researchers was present during the data collection and data analysis. During data analysis each of the researchers performed the analysis independently. The independent results were compared and merged to a list of 14 issues.

One of the researchers has been working at the VCC's unit of Manufacturing Engineering for 8 years. However, the unit of Manufacturing Engineering has not been a part of the study; the study was performed at the Electronics development department.

The chain of evidence was established and maintained from the case study questions and protocol to summarized interviews and the study report. The key informants, in this case both senior researchers and representatives from the industry followed the study and were invited to review the results before publication. The reports were written in a way that did not disclose the identity of the participants in the study.

According to Yin [16], the *internal validity* is an important issue mainly in explanatory studies. Internal validity is increased by establishing a causal relationship where certain conditions lead to other conditions. In the study presented here the problem of interviewee not feeling comfortable in talking about the sensitive issues such as presenting unrealistic estimates could be discussed. To increase interviewees' trust the interviews were not recorded and interviewees were guaranteed anonymity.

To increase the *external validity* of this case study the results of the similar studies were used to triangulate the results. Only one of the issues found in this study was not confirmed by other software engineering literature. Similarities found lead to believe that the generalization might be possible.

To increase *reliability* it must be ensured that the study can be repeated, with the same results, according to Yin [16]. This case study was carefully documented in order to make it possible for it to be replicated later on. All the data have been stored, linking the case study protocol with interview summaries, citations and the *results database*.

5 Summary

The purpose of the study that this paper is based on was to explore whether there are any underlying factors that affect the process of software cost estimation in systems development projects. The case study was conducted at Volvo Car Corporation. 14 inhibitors were reported in this paper and used to validate the results of a quantitative study with a similar objective [6]. 62,5% of the issues found in the qualitative study were validated by this study, the rest could not be confirmed.

The findings of the study underlying this paper were compared to other existing software engineering literature in order to triangulate the results. Only one of the issues could not be confirmed.

Future Work

To improve the process of cost and effort estimation, the organizational issues should be explored further. In this study the organizational issues were divided among the management, politic and to some extent user communication and methodological issues in order to mirror the Lederer and Prasad study [6], however, many of these issues are interdependent and we suggest a classification in to organizational and methodological issues instead to simplify further studies.

Observing the results of this and similar studies there seems to be a lack of deeper analysis, such as root cause analysis, and usage of external theory that could explain the issues found. This issue should be addressed in forthcoming studies.

Acknowledgements

This research has been conducted within the COSY project which is funded by the Swedish industry and government joint research program IVSS – Intelligent Vehicle Safety Systems.

We would like to thank the Volvo Car Corporation for giving us the opportunity to conduct this study, and the interviewees that participated in the study for taking time from their busy schedules in order to help us.

Also, we would like to thank associated professor Sofia Börjesson at Technology, Management and Economics department at Chalmers University of Technology for valuable input during the phases of preparation, analysis and reporting.

References

1. Lederer, A.L., Prasad, J.: Informations systems software cost estimating: a current assessment. *Journal of information technology* 8(1), 22–33 (1993)
2. Heemstra, F.J.: Software cost estimation. *Information and Software Technology* 34(10), 627–639 (1992)
3. Moores, T.T., Edwards, J.S.: Could Large UK Corporations and Computing Companies Use Software Cost Estimating Tools?-A Survey. *European Journal of Information Systems* 1(5), 311–319 (1992)
4. Grimm, K.: Software technology in an automotive company - major challenges. In: *Proceedings of 25th International Conference on Software Engineering*, pp. 498–503 (2003)
5. Broy, M.: Challenges in automotive software engineering. In: *Proceeding of the 28th international conference on Software engineering*, pp. 33–42 (2006)
6. Lederer, A.L., Prasad, J.: Causes of inaccurate software development cost estimates. *Journal of Systems and Software* 31(2), 125–134 (1995)
7. Jørgensen, M., Sheppard, M.: A Systematic Review of Software Development Cost Estimation Studies. *IEEE Transactions on Software Engineering* 33(1), 33–53 (2007)
8. Hill, J., Thomas, L.C., Allenb, D.E.: Experts' estimates of task durations in software development projects. *International Journal of Project Management* 18(1), 13–21 (2000)
9. Hughes, R.T.: Expert judgement as an estimating method. *Information and Software Technology* 38(2), 67–75 (1996)
10. Boehm, B., Clark, B., Horowitz, E., Westland, C., Madachy, R., Selby, R.: Cost models for future software life cycle processes: COCOMO 2.0. *Annals of Software Engineering* 1(1), 57–94 (1995)
11. Shepperd, M., Schofield, C.: Estimating software project effort using analogies. *IEEE Transactions on Software Engineering* 32(11), 736–743 (1997)
12. Kitchenham, B.: *Software Metrics: Measurement for Software Process Improvement*. Blackwell Publishers, Malden (1996)
13. Phan, D., Vogel, D., Nunamaker, J.: The Search for Perfect Project Management. *Computerworld*, 97–100 (1988)
14. van Genuchten, M.: Why is software late? An empirical study of reasons for delay in software development. *IEEE Transactions on Software Engineering* 17(6), 582–590 (1991)
15. Subramanian, G.H., Breslawski, S.: An empirical analysis of software effort estimate alterations. *Journal of Systems and Software* 31(2), 135–141 (1995)
16. Yin, R.: *Case study research: design and methods*, 3rd edn. (2003)

17. Buchanan, D.A., Huczynski, A.: *Organizational Behaviour: An Introductory Text*. Prentice-Hall, Englewood Cliffs (1997)
18. Anda, B., Hove, S.E.: Experiences from Conducting Semi-structured Interviews in Empirical Software Engineering Research. In: *Proceedings of the 11th IEEE International Software Metrics Symposium (METRICS 2005)* (2005)
19. Jørgensen, M.: Top-down and bottom-up expert estimation of software development effort. *Information and Software Technology* 46(1), 3–16 (2004)
20. Bruce, M., Leverick, F., Littler, D., Wilson, D.: Success factors for collaborative product development: a study of suppliers of information and communication technology. *R&D Management* 25(1), 33–44 (1995)

Impact of Base Functional Component Types on Software Functional Size Based Effort Estimation

Luigi Buglione¹ and Cigdem Gencel²

¹ École de Technologie Supérieure (ETS) / Engineering.it

Luigi.Buglione@computer.org

² Blekinge Institute of Technology, Department of Systems and Software Engineering
cigdem.gencel@bth.se

Abstract. Software effort estimation is still a significant challenge for software management. Although Functional Size Measurement (FSM) methods have been standardized and have become widely used by the software organizations, the relationship between functional size and development effort still needs further investigation. Most of the studies focus on the project cost drivers and consider total software functional size as the primary input to estimation models. In this study, we investigate whether using the functional sizes of different functionality types, represented by the Base Functional Component (BFC) types; instead of using the total single size figure have a significant impact on estimation reliability. For the empirical study, we used the projects data in the International Software Benchmarking Standards Group (ISBSG) Release 10 dataset, which were sized by the COSMIC FSM method.

Keywords: Functional Size Measurement, Effort Estimation, COSMIC, Base Functional Component, International Software Benchmarking Standards Group (ISBSG).

1 Introduction

Forty years after the term “software engineering” was coined [28] great effort has been put forth to identify and fine tune the “software process” and its proper management. Unique tools and techniques have been developed for software size, effort, and cost estimation to address challenges facing the management of software development projects [16][42][45].

A considerable amount of these efforts have been put on software size measurement based on the fact that software size is the key measure. Function Point Analysis (FPA) was designed initially in 1979 [1] by Albrecht. This method was aimed at overcoming some of the shortcomings of measures based on Source Lines of Code (SLOC) for estimation purposes and productivity analysis, such as their availability only fairly late in the development process and their technology dependence.

FPA method was based on the idea of determining size based on capturing the amount of functionality laid out on software functional requirements. They take into account only those elements in the application layer that are logically ‘visible’ to the

user and not the technology or the software development methodology used. Since the introduction of the concept, the topic of FPA evolved quite a bit. Many variations and improvements on the original idea were suggested [11], some of which proved to be milestones in the development of Functional Size Measurement (FSM).

FPA was designed in a business application environment and has become a *de facto* standard for this community. During the following years, a large number of variants for both business application software and for other application domains (such as real-time, Web, Object Oriented, and data warehouse systems)¹ were developed. In the '90s, work was initiated at the International Organization for Standardization (ISO) level to lay the common principles and foundations for regulating *de jure* standards in FSM. Between 1998 and 2005, the 14143 standard family was developed [31]² [33]-[37] with four instantiations matching with those requirements; the Common Software Measurement International Consortium Full Function Points (COSMIC FFP) [38][46] the International Function Point Users Group (IFPUG) FPA [39][43], MarkII FPA [40][44] and the Netherlands Software Metrics Association (NESMA) FSM [41] methods. A fifth FSM method, the Finnish one by FISMA [48], will be standardized in a while. The evolution of current FSM methods is shown in Figure 1.

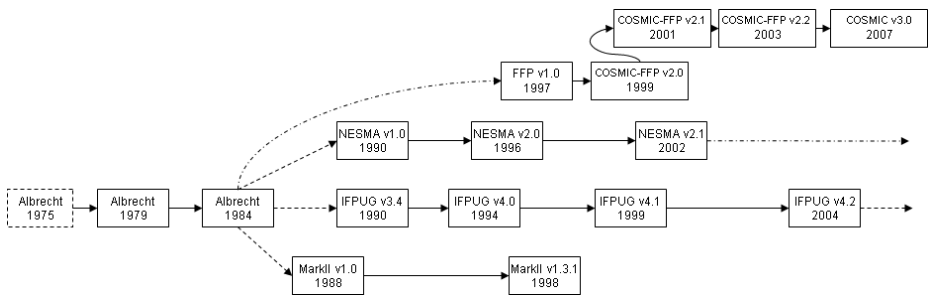


Fig. 1. Evolution of the main Functional Size Measurement (FSM) methods

Among those, COSMIC³ [46] adopted in 2003 as ISO 19761 [38], has been defined as a 2nd generation FSM method as a result of a series of innovations, such as a better fit with both real-time and business application environments, identification and measurement of multiple software layers, different perspectives of functional users from which the software can be observed and measured, and the absence of a weighting system.

Due to these constructive progresses, FSM has begun to be widely used for software size measurement. The number of benchmarking data on the projects which were measured by FSM methods has significantly increased in well-known and recognized benchmarks such as the one by ISBSG [13] with more than 4,100 projects. On the other hand, one of the major uses of software size measurement is its use in software effort estimation for software management purposes. However, effort estimation still remains a challenge for software practitioners and researchers.

¹ Please refer to [42] and [11] for a detailed list and a history of FSM-like methods.

² Part 1 (14143-1) has recently been updated (February 2007) [32] from its first release [31] (1998).

³ From version 3.0, the old name of this method (COSMIC-FFP) is become simply 'COSMIC'.

Effort estimation based on the functional size figures have just begun to emerge as more empirical data are collected in benchmarking datasets as in ISBSG dataset. The nature of the relationship between functional size and effort has been explored in many studies (see Section 2). The project related attributes such as ‘Team Size’, ‘Programming Language Type’, ‘Organization Type’, ‘Business Area Type’ and ‘Application Type’ were considered in the estimation models. However, the common conclusion of these studies was that although different models are successfully used by different groups and for particular domains, none of them has gained general acceptance by the software community due to the fact that no model is considered to perform well enough to fully meet market needs and expectations.

The general approach of the existing studies is the functional size of a software system is expressed as a single value obtained by a specific FSM method. This single value is derived from a measurement function in all ISO-certified FSM methods, and it is the result of adding together the functional sizes of different Base Functional Component (BFC)⁴ Types to obtain a total functional size. Each BFC Type represents different type of functionality to be provided to the users.

In our previous study [47], we made an analysis on the ISBSG dataset to test our hypothesis which states that the effort required to develop the unit size of each of the BFC Types, which provide different user functionalities is different and hence contributes to total effort at different levels. The results showed that using the functional sizes of each BFC Type as inputs to effort estimation improve the estimation reliability. In that study, we considered ‘Application Type’ to form the homogenous sub-groups of projects for the statistical analysis.

In the study presented here, we further investigate the contribution of different functionality types represented by BFC Types to total development effort. We again utilized the project data, which were measured by COSMIC-FFP [46] in the ISBSG dataset Release 10 [13]. In this case, we formed the sub-groups of projects with respect to ‘Development Type’. Then, we made Pareto analysis to further investigate the effect of the size of the projects on the estimation reliability. We also analyzed the distribution of different BFC Types in different Application Types.

The paper is organized as follows: Section 2 presents some background on functional size measurement and related work on its relationship to project effort. Section 3 presents the data preparation process. Section 4 presents the data analysis and Section 5, the conclusions of this study.

2 Related Work

There is a large body of literature on software effort estimation models and techniques in which a discussion on the relationship between software size and effort as a primary predictor has been included, such as [2][5][6][14][15][17][18].

Other factors related to non-functional characteristics of software projects are also included in many estimation models. Significant variations on the impact of other project cost drivers have been observed. Therefore a number of experimental studies were performed to investigate their impact on the size-effort relationship. Among the

⁴ BFC Type: A defined category of BFCs. A BFC is an elementary unit of an FUR defined by and used by an FSM method for measurement purposes [31].

cost drivers investigated, 'Team Size', 'Programming Language Type', 'Organization Type', 'Business Area Type', 'Application Type' and 'Development Platform' have been found to affect the size-effort relationship at different levels of significance [23][24][25][26][27][29]. Among these, the most significant are reported in [23][24] to be 'Team Size', 'Business Area Type' and 'Application Type'.

Constructive Cost Model (COCOMO) II [6], the revised version of the original COCOMO [5] takes into account the cost drivers in the estimation models and provide for measuring functional size and converting this result to SLOC. However, 'backfiring' of SLOC from functional size still can not account for the extra uncertainty introduced by adding another level of estimation [7][8][9].

In [22], Leung and Fan discuss both the strengths and weaknesses of effort estimation models. They evaluated the performance of existing models as well as of newer approaches to software estimation and found them as unsatisfactory. Similarly, in a number of studies, such as [2][19][20][21], related work on effort and cost estimation models is assessed and compared. They concluded that the models, which are being used by different groups and in different domains, still have not gained universal acceptance.

Most of the above approaches use functional size as the primary predictor and consider other project parameters in effort estimation. Abran et al. [3] used the 2003 version of the ISBSG repository to build estimation models for projects sized by the FPA method. They defined the concept of a software functional profile as the distribution of function types within the software. They investigated whether or not the size-effort relationship was stronger if a project was close to the average functional profile of the sample studied. For each sample, it was noted that there was one function type that had a stronger relationship with project effort. Moreover, the sets of projects located within a certain range of the average profile led to estimation models similar to those for the average functional profile, whereas projects located outside the range gave different regression models, these being specific to each of the corresponding subsets of projects.

In [4], the impact of the functional profile on project effort was investigated using the ISBSG repository. The ISBSG projects included in this analysis were sized by COSMIC method. In COSMIC, a functional profile corresponds to the relative distribution of its four BFC Types for any particular project. It was observed that the identification of the functional profile of a project and its comparison with the profiles of their own samples can help in selecting the best estimation models relevant to its own functional profile.

In [10], the types of functionalities a software system can provide to its users are identified, and a multidimensional measure which involves measuring the functional size of each functionality type is defined. It was suggested that experimental studies should be conducted to find the relationship between the functional size of each functionality type and the effort needed to develop the type of functionality that can pioneer new effort estimation methods.

In [47], Gencel and Buglione explored whether effort estimation models based on the BFC types, rather than those based on a single total value would improve estimation models. They observed a significant improvement in the strength of the size-effort relationship.

3 Data Preparation

In this study, the projects data in the ISBSG 2007 Repository CD Release 10 [13] were used for the statistical analysis. The ISBSG Repository includes more than 4,106 projects data on a very wide range of projects. Among those, 117 projects were sized using COSMIC-FFP. The projects cover a wide range of applications, development techniques and tools, implementation languages, and platforms. Table 1 shows the filtration process with respect to the project attributes defined in the ISBSG dataset.

Table 1. Filtration of ISBSG 2007 Dataset Release10

Step	Attribute	Filter	Projects Excluded	Remaining Projects
1	Count Approach ⁵	= COSMIC-FFP	3,989	117
2	Data Quality Rating (DQR)	= {A B}	5	112
3	Quality Rating for Unadjusted Function Points (UFP)	= {A B}	21	91
4	Development Type	= {New Development}	22	34
		= {Enhancement}		30
		= {Re-development}		5

In the first step, we filtered the dataset with respect to the ‘Count Approach’ attribute to obtain the projects measured by COSMIC. This step provided 117 projects.

In the second step, we analyzed these 117 projects with respect to ‘Data Quality Rating (DQR)’ to keep only the highest quality data for statistical analysis. In the ISBSG dataset, each project has a Quality Tag⁶ (A, B, C or D) assigned by the ISBSG reviewers based on whether or not the data fully meet ISBSG data collection quality requirements. Considering this ISBSG recommendation, 5 of the projects with a C and D rating were ignored, leaving 112 projects following this filtration step.

In the third step, we verified the availability of fields of size by functional type (or BFC) in the data set, for each of the 112 projects from step 2, since these fields are necessary for this study. The verification indicates that this information is not available for 21 of the projects, leaving 91 projects for the next step.

Since many factors vary simultaneously, the statistical effects may be harder to identify in a more varied dataset than in a more homogeneous one. Therefore, in Step 4, we built a series of homogeneous subsets considering the ‘Development Type’ attribute. We built homogeneous subsets for ‘New Development’, ‘Enhancement’ and ‘Re-development’ projects out of the 91 remaining projects. While forming the

⁵ No further filter has been considered with respect to the COSMIC versions.

⁶ A: The data submitted were assessed as sound, with nothing identified that might affect their integrity; B: The submission appears fundamentally sound, but there are some factors which could affect the integrity of the submitted data; C: Due to significant data not being provided, it was not possible to assess the integrity of the submitted data; D: Due to one factor or a combination of factors, little credibility should be given to the submitted data.

subsets, we removed the outlier projects which have very low productivity values. Since the data points for the Re-development projects were too few for statistical analysis (5 projects), we removed them from further analysis.

While exploring the nature of the relationship, we did not consider the impact of ‘Application Type’. In our previous study [47] we observed that the strength of relationship between functional size and effort are much lower when we formed homogeneous subsets with respect to Application type (0.23 for Subset 1; 0.56 for Subset 2 and 0.39 for Subset 3). But, we observed increases in R^2 values (0.23 to 0.41 for Subset 1; 0.56 to 0.60 for Subset 2 and 0.39 to 0.54 for Subset 3) when the functional sizes of each of the BFC Types are taken into account for effort estimation purposes instead of total functional size which motivated us to further investigate the effects of BFC Types on the strength of the relationship.

4 Statistical Data Analysis and Results

The primary aim of this study is to explore whether or not an effort estimation model based on the components of functional size rather than on only a total single value of functional size would improve estimation models and if so formulating the estimation model.

In this study, the two sub-datasets are first analyzed to determine the strength of the relationship between the total functional size and the development effort by applying a Linear Regression Analysis method. Then, the strength of the relationship between the functional sizes of the COSMIC BFC Types used to determine total functional size and development effort is analyzed by applying a Multiple Regression Analysis method. These findings are compared to the models representing the relationship between total functional size and effort. All the statistical data analyses in this study were performed with the GiveWin 2.10 [12] commercial tool and its sub-modules and the Microsoft-Excel ‘Data Analysis ToolPak’⁷.

4.1 Total Functional Size - Effort Relationship

For the Linear Regression Analysis [30], we have the independent variable as Functional Size and the dependent variable as the Normalized Work Effort (NW_Effort) as given by the following formula;

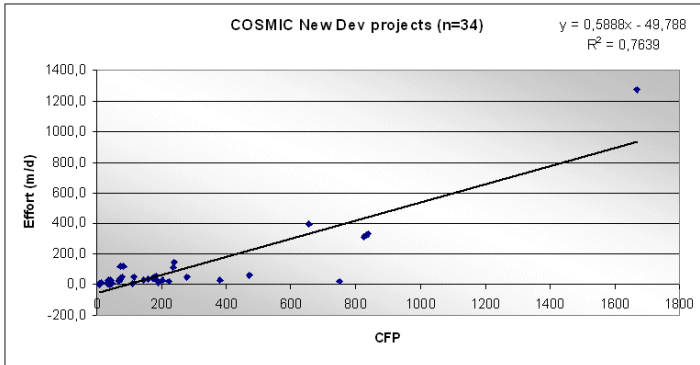
$$NW_Effort = B_0 + B_1 FunctionalSize \quad (1)$$

where B_0 and B_1 are the coefficients to be estimated from a generic data sample. Normalized Work Effort variable is used so that the effort data among the projects which do not include all the phases of the development life cycle are comparable.

Figure 2 shows the relationship between Normalized Work Effort and COSMIC Function Points (CFP). For the New Development Projects dataset, the R^2 statistic is better than that for the Enhancement Project datasets.

⁷ <http://office.microsoft.com/en-gb/excel/HP052038731033.aspx>

a) Sub-dataset 1: New Development Projects (n=34)



b) Sub-dataset 2: Enhancement Projects (n=30)

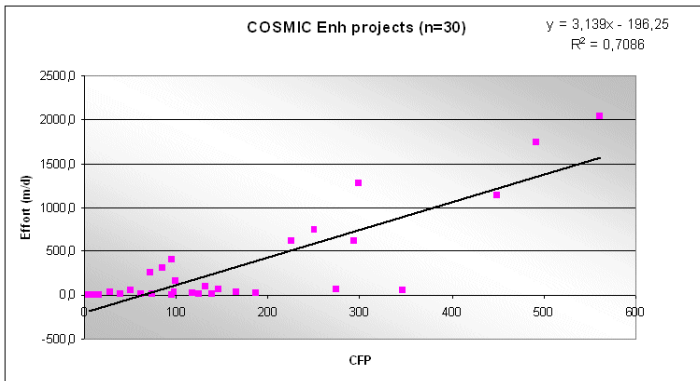


Fig. 2. The Relationship between Normalized Work Effort and COSMIC Functional Size

A significance test is also carried out in building a linear regression model. This is based on a 5% level of significance. An F-test is performed for the overall model. A ($\text{Pr} > F$) value of less than 0.05 indicates that the overall model is useful. That is, there is sufficient evidence that at least one of the coefficients is non-zero at a 5% level of significance. Furthermore, a t-test is conducted on each β_j ($0 \leq j \leq k$). If all the values of ($\text{Pr} > |t|$) are less than 0.05, then there is sufficient evidence of a linear relationship between y and each x_j ($1 \leq j \leq k$) at the 5% level of significance. The results of the linear regression analysis are given in Table 2.

For subsets 1 and 2, the Total Functional Size is found to explain about 76% and 71% of the NW_Effort respectively. See [50] for an exhaustive discussion and detailed explanation about the meaning of the statistical variables. Because two subsets obtained proper R^2 values against a quite high number of data points, they were not split by size ranges⁸ or by application types. In this case a further split, the too reduced number of data points would not assure a statistical significance of the obtained results.

⁸ See [51] for a size range classification applying Pareto Analysis, applied on ISBSG r9 data repository.

Table 2. Regression Analysis Results (Normalized Work Effort – Total Functional Size)

Subset 1: New Development Projects							
	Coeff	StdError	t-value	t-prob	Split1	Split2	reliable
Constant	-49.78763	24.48831	-2.033	0.0504	0.0363	0.4419	0.7000
Functional Size	0.58882	0.05787	10.174	0.0000	0.0000	0.0000	1.0000
R² = 0.7639							
	value	prob					
normality test	28.5832	0.0000					
Subset 2: Enhancement Projects							
	Coeff	StdError	t-value	t-prob	Split1	Split2	reliable
Constant	-196.24813	83.73519	-2.344	0.0264	0.2963	0.0081	0.7000
Functional Size	3.13900	0.38040	8.252	0.0000	0.0004	0.0000	1.0000
R² = 0.7086							
	value	prob					
normality test	4.3408	0.1141					

4.2 Functional Sizes of BFC Types – Size-Effort Relationship

The COSMIC method [38][46] is designed to measure the software functional size based on its Functional User Requirements (FURs). Each FUR is decomposed into its elementary components, called Functional Processes⁹. The BFCs of this method are assumed to be Data Movement Types, which are of four types; Entry (E), Exit (X), Read (R) and Write (W). The functional size of each Functional Process is determined by counting the Entries, Exits, Reads and Writes in each Functional Process, and the Total Functional Size is the sum of the functional sizes of the Functional Processes.

In this study, the Multiple Regression Analysis method [30] was used to analyze the relationship between the dependent variable Normalized Work Effort and the functional sizes of each BFC Type as the dependent variables. The following multiple linear regression model [30] that expresses the estimated value of a dependent variable y as a functions of k independent variables, x₁, x₂, , x_k, is used:

$$y = B_0 + B_1x_1 + B_2x_2 + \dots + B_k X_k \tag{2}$$

where B₀, B₁, B₂, B_k are the coefficients to be estimated from a generic data sample. Thus, the effort estimation model can then be expressed as:

$$NW_Effort = B_0 + B_1(E) + B_2(X) + B_3(R) + B_k(W) \tag{3}$$

where, NW_Effort (Normalized Work Effort) is the dependent variable and E, X, R and W are the independent variables representing the number of Entries, Exits, Reads and Writes respectively. In building a multiple linear regression model, the same significance tests as discussed in the previous section are carried out. Table 3 shows the multiple regression analysis results.

⁹ Functional Process: “an elementary component of a set of FURs comprising a unique, cohesive and independently executable set of data movements” [38].

Table 3. Multiple Regression Analysis Results (Normalized Work Effort – Functional Sizes of BFC Types)

Sub-dataset 1: New Development Projects dataset							
Observations: 34							
	Coeff	StdError	t-value	t-prob			
Constant	-31.83818	18.46448	-1.724	0.0953			
E	0.72694	0.38916	1.868	0.0719			
X	0.01875	0.25507	0.073	0.9419			
R	-0.03702	0.24675	-0.150	0.8818			
W	2.21199	0.42239	5.237	0.0000			
R² = 0.8919							
normality test		value	prob				
		13.2388	0.0013				
After F presearch testing,							
	Coeff	StdError	t-value	t-prob	Split1	Split2	reliable
Constant	-32.10285	17.75256	-1.808	0.0803	0.1592	0.0360	0.7000
E	0.74298	0.23129	3.212	0.0031	0.0004	0.0000	1.0000
W	2.17018	0.30448	7.128	0.0000	0.0000	0.4214	0.7000
R² = 0.8918							
Sub-dataset 2: Enhancement Projects Dataset							
Observations: 30							
	Coeff	StdError	t-value	t-prob			
Constant	-46.26395	67.37480	-0.687	0.4986			
E	-0.47787	1.91093	-0.250	0.8046			
X	7.37899	1.40681	5.245	0.0000			
R	-1.76768	1.35114	-1.308	0.2027			
W	8.08448	2.59471	3.116	0.0046			
R² = 0.8755							
normality test		value	prob				
		3.3048	0.1916				
After F presearch testing, specific model of WE;							
	Coeff	StdError	t-value	t-prob	Split1	Split2	reliable
X	7.61616	1.31971	5.771	0.0000	0.0000	0.0000	1.0000
R	-2.51783	0.99965	-2.519	0.0180	0.1747	0.0129	0.7000
W	7.55544	2.47507	3.053	0.0050	0.1043	0.0058	1.0000
R² = 0.8713							

In Table 4, the results from the two approaches are summarized. The results show that the R^2 is higher using the four BFC Types rather than the single total COSMIC FPs (+16.7% for new development; +23.6% for enhancement projects).

Another observation from the regression analysis results is that the functional sizes of not all BFC Types are found to be significant in estimating the effort. Two of the BFC Types, i.e. Entry and Write for New Development projects and Exit, Read and Write for Enhancement projects were found to model Normalized Work Effort.

Table 4. Comparison of the Results

Sub-datasets	# of Data Points	R ² (Using Total Functional Size (CFP))	R ² (Using BFC Types)	Increase ¹⁰ (%)
Sub-dataset 1: New Development	34	0.76	0.89	+16.7%
Sub-dataset 2: Enhancement	30	0.71	0.88	+23.6%

So, the next two questions were; 1) What about the prediction capability of an estimation model using only the BFC Types found to be significant in estimating the effort, not necessarily all the four ones at a time? 2) Is there a correlation between the contribution of BFC Types to total functional size and the BFC Types which are found to be significant in estimating the effort? Table 5 shows the results for Question 1.

Table 5. Comparison of the Results

		R ²	FORMULA
Sub-dataset 1: New Development Projects (n=34)	Total functional size (CFP)	0.7639	$Y=0.5888*CFP-49.788$
	E/X/W/R	0.8919	$Y=0.72694*E+0.011875*X-0.03702*R+2.21199*W-31.83818$
	E/X	0.8918	$Y=0.74298*E+2.17018*W-32.10285$
Sub-dataset 2: Enhancement Projects(n=30)	Total functional size (CFP)	0.7086	$Y=3.139*CFP-196.25$
	E/X/W/R	0.8755	$Y=-0.47787*E+7.37899*X-1.76768*R+8.08448*W-46.26395$
	X/R/W	0.8713	$Y=7.61616*X-2.51783*R+7.55544*W$

Thus, for New Development projects, the functional sizes of only E and W types of BFCs and for Enhancement Projects, X, R and W types can as better estimate the effort as when the functional sizes of all four types are used. In order to answer Question 2, we analyzed the distribution of the BFC Types with respect to the Development Type (see Figure 3).

The contribution to total functional size to Enhancement projects by R type BFC is the greatest, while X and E types contribute more for New Development projects. In terms of BFC Types, E, X and W types are predominant in New Development projects, while R in Enhancement ones.

Thus, we could not find a correlation between the level of contribution of BFC Types to total functional size and the ones which are found to be significant in estimation capability of an estimation model.

¹⁰ It was calculated as the relative increment: $[(R^2(\text{BFC})-R^2(\text{CFP}))/R^2(\text{CFP})]$.

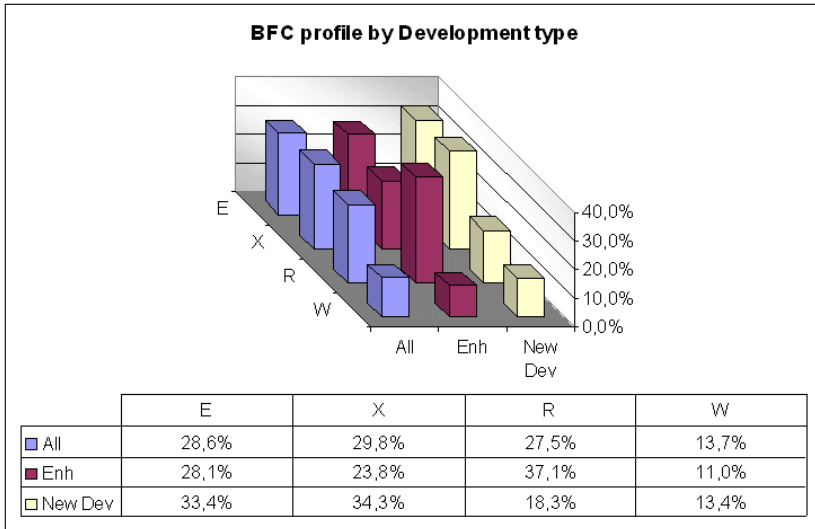


Fig. 3. The distribution of BFC Types by Development Type

5 Conclusions and Prospects

This study has explored whether an effort estimation model based on the functional sizes of BFC Types rather than the total functional size value would provide better results. Our hypothesis was that the development effort for each of the BFC Types, which provide different user functionalities, might be different.

The R^2 statistics were derived from Linear Regression Analysis to analyze the strength of the relationship between total functional size and normalized work effort. The results were compared to the R^2 statistics derived from the Multiple Regression Analysis performed on the Functional Sizes of the BFC Types and Normalized Work Effort. We observed increases in R^2 values (0.76 to 0.89 for New Development projects and 0.71 to 0.88 for Enhancement projects) when the functional sizes of each of the BFC Types are taken into account for effort estimation purposes instead of the total functional size. The results showed a significant improvement, i.e. +16.7 % for new development projects and +23.6% for enhancement projects, in the effort estimation predictability.

Another interesting observation in this study is that the functional sizes of all BFC Types are not found to be significant in estimating the effort. Two of the BFC Types, i.e. Entry and Write for New Development projects and Exit, Read and Write for Enhancement projects were found to better model Normalized Work Effort.

We also analyzed the dominating BFC types in each of the datasets analyzing the frequency distribution. For New Development projects, it is the Entry (33.4%) and Exit (34.3%) that are dominant among the four BFC types. For Enhancement projects Entry (28.1%), Exit (23.8%) and Read (37.1%) that are all dominant. The results of these analysis showed that there is no correlation between the dominating BFC Types

in the dataset and the BFC Types which are found to be significant in estimating the effort.

Our hypothesis in this study was developing different functionality types requires different amounts of work effort and contributes to effort estimation in different levels of significance. The results of this study confirmed our hypothesis. Although we built some estimation formulas based on the data in ISBSG dataset, our aim in this study was not to arrive at a generic formula but rather compare the conventional approach to effort estimation and our approach discussed in this paper. Further research is required to analyze which BFC Types are significant in estimating effort and to conclude the ones to be used for establishing reliable estimation models. Further work should also include comparisons with related work performed with the IFPUG FPA method.

Because of the improvements in the estimation results just using four proxies instead of the solely functional size unit value, the organizational consideration would be the data gathering process. Usually, only the total functional size values are stored, not the whole detail derived from the measurement. However, with a low additional cost in terms of time in the data insertion it would be possible to obtain better estimation premises. In process improvement terms, using the terminology of a well known and proven maturity model as Capability Maturity Models Integration (CMMI) [49], this action would have a positive impact on:

- **PP** (*Project Planning*, Specific Practice (SP)1.4 about the estimation model used for deriving estimates comparing estimated and actual values;
- **MA** (*Measurement & Analysis*, SP2.3) about the storage of project data;
- **OPD** (*Organizational Process Definition*) about the definition of the measurement repository (SP1.4);
- **GP** (*General Practice*) **3.2** (*Collect Improvement Information*), that is the generic practice crossing all the PA (Process Areas) about the capability of collecting info to be used for improving the organizational unit's results.

Thus, starting to consider which BFC Types are significant in estimation instead of using total size figures and using establishing estimation models considering different functionality types is promising. In order to verify these conclusions and find other eventual useful relationships, further studies will also be conducted on the ISBSG dataset for the projects measured by IFPUG FPA.

References

- [1] Albrecht, A.J.: Measuring Application Development Productivity. In: Proc. Joint SHARE/GUIDE/IBM Application Development Symposium, pp. 83–92 (1979)
- [2] Abran, A., Ndiaye, I., Bourque, P.: Contribution of Software Size in Effort Estimation. Research Lab in Software Engineering, École de Technologie Supérieure, Canada (2003)
- [3] Abran, A., Gil, B., Lefebvre, E.: Estimation Models Based on Functional Profiles. In: International Workshop on Software Measurement – IWSM/MetriKon, Kronisburg (Germany), pp. 195–211. Shaker Verlag (2004)

- [4] Abran, A., Panteliuc, A.: Estimation Models Based on Functional Profiles. III Taller Internacional de Calidad en Tecnologías de Información et de Comunicaciones, Cuba, February 15-16 (2007)
- [5] Boehm, B.W.: Software Engineering Economics. Prentice-Hall, Englewood Cliffs (1981)
- [6] Boehm, B.W., Horowitz, E., Madachy, R., Reifer, D., Bradford, K.C., Steece, B., Brown, A.W., Chulani, S., Abts, C.: Software Cost Estimation with COCOMO II. Prentice Hall, New Jersey (2000)
- [7] Neumann, R., Santillo, L.: Experiences with the usage of COCOMOII. In: Proc. of Software Measurement European Forum 2006, pp. 269–280 (2006)
- [8] De Rore, L., Snoeck, M., Dedene, G.: COCOMO II Applied In A Banking And Insurance Environment: Experience Report. In: Proc. of Software Measurement European Forum 2006, pp. 247–257 (2006)
- [9] Rollo, A.: Functional Size measurement and COCOMO – A synergistic Approach. In: Proc. of Software Measurement European Forum 2006, pp. 259–267 (2006)
- [10] Gencel, C.: An Architectural Dimensions Based Software Functional Size Measurement Method, PhD Thesis, Dept. of Information Systems, Informatics Institute, Middle East Technical University, Ankara, Turkey (2005)
- [11] Gencel, C., Demirors, O.: Functional Size Measurement Revisited. Scheduled for publication in ACM Transactions on Software Engineering and Methodology (July 2008)
- [12] GiveWin 2.10, <http://www.tspintl.com/>
- [13] ISBSG Dataset 10 (2007), <http://www.isbsg.org>
- [14] Hastings, T.E., Sajeev, A.S.M.: A Vector-Based Approach to Software Size Measurement and Effort Estimation. IEEE Transactions on Software Engineering 27(4), 337–350 (2001)
- [15] Jeffery, R., Ruhe, M., Wieczorek, I.: A Comparative Study of Two Software Development Cost Modeling Techniques using Multi-organizational and Company-specific Data. Information and Software Technology 42, 1009–1016 (2000)
- [16] Jones, T.C.: Estimating Software Costs. McGraw-Hill, New York (1998)
- [17] Jørgensen, M., Molokken-Ostvold, K.: Reasons for Software Effort Estimation Error: Impact of Respondent Role, Information Collection Approach, and Data Analysis Method. IEEE Transactions on Software Engineering 30(12), 993–1007 (2004)
- [18] Kitchenham, B., Mendes, E.: Software Productivity Measurement Using Multiple Size Measures. IEEE Transactions on Software Engineering 30(12), 1023–1035 (2004)
- [19] Briand, L.C., El Emam, K., Maxwell, K., Surmann, D., Wieczorek, I.: An Assessment and Comparison of Common Software Cost Estimation Models. In: Proc. of the 21st Intern. Conference on Software Engineering, ICSE 1999, Los Angeles, CA, USA, pp. 313–322 (1998)
- [20] Briand, L.C., Langley, T., Wieczorek, I.: A Replicated Assessment and Comparison of Software Cost Modeling Techniques. In: Proc. of the 22nd Intern. Conf. on Software engineering, ICSE 2000, Limerick, Ireland, pp. 377–386 (2000)
- [21] Menzies, T., Chen, Z., Hihn, J., Lum, K.: Selecting Best Practices for Effort Estimation. IEEE Transactions on Software Engineering 32(11), 883–895 (2006)
- [22] Leung, H., Fan, Z.: Software Cost Estimation. Handbook of Software Engineering, Hong Kong Polytechnic University (2002)
- [23] Angelis, L., Stamelos, I., Morisio, M.: Building a Cost Estimation Model Based on Categorical Data. In: 7th IEEE Int. Software Metrics Symposium (METRICS 2001), London (April 2001)
- [24] Forselius, P.: Benchmarking Software-Development Productivity. IEEE Software 17(1), 80–88 (2000)

- [25] Lokan, C., Wright, T., Hill, P.R., Stringer, M.: Organizational Benchmarking Using the ISBSG Data Repository. *IEEE Software* 18(5), 26–32 (2001)
- [26] Maxwell, K.D.: Collecting Data for Comparability: Benchmarking Software Development Productivity. *IEEE Software* 18(5), 22–25 (2001)
- [27] Morasca, S., Russo, G.: An Empirical Study of Software Productivity. In: Proc. of the 25th Intern. Computer Software and Applications Conf. on Invigorating Software Development, pp. 317–322 (2001)
- [28] Naur, P., Randell, B. (eds.): *Software Engineering, Conference Report*, NATO Science Committee, Garmisch (Germany), 7-11 October (1968)
- [29] Premraj, R., Shepperd, M.J., Kitchenham, B., Forselius, P.: An Empirical Analysis of Software Productivity over Time. In: 11th IEEE International Symposium on Software Metrics (Metrics 2005). IEEE Computer Society Press, Los Alamitos (2005)
- [30] Neter, J., Wasserman, W., Whitmore, G.A.: *Applied Statistics*. Allyn & Bacon (1992)
- [31] ISO/IEC 14143-1: Information Technology – Software Measurement – Functional Size Measurement – Part 1: Definition of Concepts (1998)
- [32] ISO/IEC 14143-1: Information Technology – Software Measurement – Functional Size Measurement – Part 1: Definition of Concepts (February 2007)
- [33] ISO/IEC 14143-2: Information Technology – Software Measurement – Functional Size Measurement - Part 2: Conformity Evaluation of Software Size Measurement Methods to ISO/IEC 14143-1:1998 (2002)
- [34] ISO/IEC TR 14143-3: Information Technology – Software Measurement – Functional Size Measurement – Part 3: Verification of Functional Size Measurement Methods (2003)
- [35] ISO/IEC TR 14143-4: Information Technology – Software Measurement – Functional Size Measurement - Part 4: Reference Model (2002)
- [36] ISO/IEC TR 14143-5: Information Technology – Software Measurement – Functional Size Measurement – Part 5: Determination of Functional Domains for Use with Functional Size Measurement (2004)
- [37] ISO/IEC FCD 14143-6: Guide for the Use of ISO/IEC 14143 and related International Standards (2005)
- [38] ISO/IEC 19761:2003, Software Engineering – COSMIC-FPP: A Functional Size Measurement Method, International Organization for Standardization(2003)
- [39] ISO/IEC IS 20926:2003, Software Engineering-IFPUG 4.1 Unadjusted Functional Size Measurement Method - Counting Practices Manual, International Organization for Standardization (2003)
- [40] ISO/IEC IS 20968:2002, Software Engineering – MK II Function Point Analysis – Counting Practices Manual, International Organization for Standardization (2002)
- [41] ISO/IEC IS 24570:2005, Software Engineering – NESMA functional size measurement method version 2.1 – Definitions and counting guidelines for the application of Function Point Analysis, International Organization for Standardization (2005)
- [42] Symons, C.: Come Back Function Point Analysis (Modernized) – All is Forgiven! In: Proc. of the 4th European Conf. on Software Measurement and ICT Control (FESMA-DASMA 2001), Germany, pp. 413–426 (2001)
- [43] The International Function Point Users Group (IFPUG). *Function Points Counting Practices Manual* (release 4.2), International Function Point Users Group, Westerville, Ohio (January 2004)
- [44] United Kingdom Software Metrics Association (UKSMA). *MkII Function Point Analysis Counting Practices Manual*, v 1.3.1 (1998)
- [45] Thayer, H.R.: *Software Engineering Project Management*, 2nd edn. IEEE Computer Society Press, Los Alamitos (2001)

- [46] The Common Software Measurement International Consortium (COSMIC). COSMIC-FFP v.3.0, Measurement Manual (2007)
- [47] Gencel, C., Buglione, L.: Do Different Functionality Types Affect the Relationship between Software Functional Size and Effort? In: Proceedings of the Intern. Conf. on Software Process and Product Measurement (IWSM-MENSURA 2007), Palma de Mallorca, Spain, November 5-8, 2007, pp. 235–246 (2007)
- [48] FISMA, PAS Submission to ISO/IEC JTC1/SC7 – Information Technology – Software and Systems Engineering – FISMA v1.1 Functional Size Measurement Method, Finnish Software Metrics Association (2006), http://www.fisma.fi/wp-content/uploads/2007/02/fisma_fsmm_11_iso-final-1.pdf
- [49] CMMI Product Team, CMMI for Development, Version 1.2, CMMI-DEV v1.2, Continuous Representation, CMU/SEI-2006-TR-008, Technical Report, Software Engineering Institute (August 2006), <http://www.sei.cmu.edu/pub/documents/06.reports/pdf/06tr008.pdf>
- [50] Maxwell, K.: Applied Statistics for Software Managers. Prentice Hall, Englewood Cliffs (2002)
- [51] Santillo, L., Lombardi, S., Natale, D.: Advances in statistical analysis from the ISBSG benchmarking database. In: Proceedings of SMEF (2nd Software Measurement European Forum), Rome (Italy), March 16-18, 2005, pp. 39–48 (2005), <http://www.dpo.it/smef2005>

Managing Uncertainty in ERP Project Estimation Practice: An Industrial Case Study

Maya Daneva

University of Twente
m.daneva@utwente.nl

Abstract. Uncertainty is a crucial element in managing projects. This paper's aim is to shed some light into the issue of uncertain context factors when estimating the effort needed for implementing enterprise resource planning (ERP) projects. We outline a solution approach to this issue. It complementarily deploys three techniques to allow a tradeoff between ERP projects requiring more effort than expected and those requiring less. We present the results of a case study carried out in a telecommunication company site.

1 Introduction

ERP effort estimation is a key consideration when ERP consultants prepare a bid in a request-for-proposal process and when an ERP adopter compares alternative bids to make a choice on their ERP implementation consulting partner. Those involved in bid preparation or bid comparison are well aware of the fact that ERP projects very often suffer from unexpected rework and delays [1,10,14,21,26] caused by factors going beyond their control. While this situation might sound like a common symptom for many types of software projects (including ERP), researchers [8,14,17,24,25,27,28] indicate that commonplace effort/duration estimation techniques don't fit as solution vehicles in ERP project context, thus leaving both consultants and adopters with little or no support in their effort estimation efforts. For example, [5,9,10,14,17,18,20,21,24,26,27,28], indicate that a typical ERP project contains multiple sources of uncertainty, and that historical data don't exist for some significant sources of volatility. Examples of such context characteristics are the degree of adjusting the vendor's off-the-shelf package to the specific ERP-adopter's requirements [10,14,17,20], the complex interaction between software engineering and business (re)engineering activities within the project [1,5,10,21], the strength of the coupling among the modules making up the package [14,18,27].

In this article, we start a systematic study of balancing uncertainties in ERP project estimation from the perspective of the ERP-adopting organization and as part of the ERP requirements engineering (RE) process. Our objective is to provide adopters with a vehicle to help them reason about cost and schedule implications of their ERP implementation decisions which they may need to make at the stage of early requirements. We show how an integrated approach, which complements a traditional effort/duration model (namely COCOMO II) with the concepts of portfolio

management and Monte Carlo simulation, allows a tradeoff between ERP projects requiring more effort than expected and those requiring less. Two ideas are key to our approach to uncertainty: (i) the use of probability distribution definitions to characterize project context factors [19], and (ii) the use of portfolios of projects [12], instead of treating projects separately, as traditional models do (e.g. COCOMO II).

We structure the presentation as follows. Section 2 discusses how ERP projects and custom software projects are different from efforts estimation perspective. In Section 3 we provide some background, including a discussion of effort estimation techniques, which bear some similarity to our approach. Therein, we also construct our multi-concept-based solution approach and, in Section 4, we present the case study in which we applied it. Some possible validity threats are analyzed in Section 5. Early conclusions about how our approach needs to be improved are presented in Section 6.

2 Thinking of ERP Systems from Effort Estimation Perspective

We mean this section a sidebar for readers who are less familiar with cross-organizational ERP systems and ERP projects. Our motivation for including a discussion on ERP systems from effort estimation perspective is to help the readers understand the rest of the paper and to avoid misunderstandings.

ERP systems are packaged software solutions, the key function of which is to coordinate work in a business. They are the vehicles modern organizations use to achieve true business connectivity, a state in which everyone knows what everyone else is doing in the business all over the world and at the same time. In the past decade, the importance of ERP systems to companies has dramatically increased as companies have begun to realize how decisive the impact of ERP is on their future: organizations and their business partners and customers have started developing ‘value webs’, and ERP systems have become the tool of choice for obtaining the coordination support that companies need and for staying interconnected [7,10,21]. By ‘value web’, we mean a set of different, independent (or nearly independent) businesses forming a business network — for example, the business value web of Cisco Systems, a company who collaborates with a large number of its big customers worldwide. Cisco simplified and standardized its customer-facing processes through an Oracle 11i Everest ERP solution which linked its 30000 customers and partners involved in Cisco’s so-called Quote-to-Cash chain [4]. A value web can also be any large company which has restructured as a set of nearly independent business units, each responsible for its own profit and loss. For example, Monsanto, a chemical engineering business, including dozens of business units most of which use an SAP solution as their collaboration support platform [10].

An ERP implementation project is the customization and introduction of a cross-organizational ERP system in a value web. Our research effort is focused on investigating measurement models which can be used for ERP project cost estimation at the requirements engineering stage of ERP implementation projects; for example, estimating effort at the ERP bidding stage, at which point requirements are not yet fully known. Following [16], we consider a project quote to consist of three components: estimated cost, profit, and contingency. Here, however, we focus on the models used

to estimate cost in particular, and, for this reason, we leave aside profit and contingency.

Literature sources [9,10,14,17,18,20,21,23,24,25,27,28] comparing ERP projects to other projects indicate that, unlike business information systems projects (e.g. data warehousing or workflow management systems) or custom software projects, ERP projects:

1. are broad in terms of functionality, covering thousands of business activities;
2. treat the cross-organizational business processes in a value web as the fundamental building blocks of the system;
3. deliver a shared system which lets the business activities of one company become an integral part of the business of its partners;
4. create system capabilities far beyond the sum of the ERP components' individual capabilities, which, allows the resulting system to qualitatively acquire new properties as result of its configuration;
5. may well include diverse configurations, each of which matches the needs of a unique stakeholder group, which, in turn, implies the presence of cost drivers unique to each configuration;
6. deliver a system which is incomplete once the ERP project is over, because an ERP solution must mirror rapidly-changing business requirements, and so be adjusted regularly to accommodate current business needs;
7. don't have an identified owner at cross-organizational system level, as the system is shared;
8. may well have a low level of organizational awareness of what new project activities (e.g. identifying and analyzing capability gaps, investigation and mapping of configuration options [20]) are to be added in order to plan and manage the ERP project, and what the factors are that drive effort for these new activities.
9. are not "built" in the sense that a master architect envisions the parts and their relationships; rather they evolve into existence and change through their life cycles as new shared pieces of functionality are built, existing intra-organizational systems connect to become shared, and shared parts of the system are disintegrated as soon as needs of sharing processes and data disappear.

The analyses by the above authors suggest that these characteristics pose effort estimation challenges which are well beyond those encountered in ordinary business information systems or custom projects. Clearly, existing models account for drivers, which model a subset only of the phenomena essential for ERP effort estimation. For example, the models to date would - hopefully with some adaptation, handle a single ERP system instance, a single version or a single configuration, but would leave estimators with very little guidance on how to estimate effort/time for those implementation projects targeting multiple ERP configurations, or co-existing ERP instances of the same package [21]. Traditional models are also inflexible in that they take as inputs pre-defined parameters [14,17,24], that is, they consider size to be a one-dimensional concept (e.g. function points or lines of code). This is not enough in the ERP project realities which prompt the use of a multi-dimensional size measure [24]. For example, the preliminary empirical research [9] done by the author on how ERP

adopters and consultants define ‘size’, yielded three categories of definitions: ‘size’ was referred to as an attribute of the implementation tasks (e.g. ‘size’ is defined as the number of ERP transactions to be configured), as an attribute of the ERP user community (e.g. the number of users), or as an attribute of the ERP functionality (e.g. function points). (Within each definition category, there also were different opinions on what ‘size’ means.) We found, that among these definitions, FPs – as a characteristic of functionality, is the only one which has been used as input in the models of the COCOMO II family. Furthermore, traditional models rest on an accumulated body of software measurement knowledge from the past 25-30 years, while effort estimation for ERP could not take advantage of such a body of knowledge merely because software engineering and business (re)engineering processes are inseparable in ERP projects. This, in turn, poses a unique challenge to effort estimation analysts because the growing complexity of the cross-organizational business processes means growing complexity in the ERP solution that embeds these processes [21]; suppose, we apply Glas’ estimation [13] that for every 25% increase in complexity of the task to be automated, the increase in complexity of the solution is 100%, one could imagine the magnitude of complexity ERP-adopters face.

The literature we reviewed in this section provides evidence that the nine characteristics above (labeled 1-9) make it almost impossible for ERP-adopters to determine a level of trust in any estimate. Examples of some specific barriers to trust, which researchers [1,10,23,24,25] have found to be traceable to the above ERP project characteristics, include: lack of consensus on the objectives of the estimates, no known steps to ensure the integrity of the estimation process, no historical evidence at the ERP adopter's site supporting a reliable estimate, or the inability to clearly see whether or not estimates are consistent with consultants’ demonstrated accomplishments on other projects in comparable organizations in the same sector.

3 Sources, Approach and Related Work

Our solution rests on four types of sources: (i) the COCOMO II reference model [2] that lets us account for ERP adopter’s specific cost drivers, (ii) the Monte Carlo simulation [19] which lets us approach the cost drivers’ degrees of uncertainty, (iii) the effort-and-deadline-probability-based portfolio management concept [12] which lets us quantify the chance for success with proposed interdependent deadlines for a set of related ERP projects, and (iv) our own experience in ERP RE [4,5,6]. We chose the combination of (i), (ii) and (iii), because other researchers already experimented with it [15] and found it encouraging. Unlike [15], where the three methods were used complementarily for the purpose of custom software contract bidding, we adapt each of the methods to the context of ERP projects and we adopt their joint use therein.

3.1 COCOMO II

COCOMO II [2] is one of the best-known algorithmic model for setting budgets and schedules as a basis for planning and control. It comprises (i) five scale factors, which reflect economies and diseconomies of scale observable in projects of various sizes, and (ii) 17 cost drivers, which serve to adjust initial effort estimations. In ERP project

settings, at least three of the scale factors are directly related to the joint RE and architecture design activities, and thus raises the role of architects in reducing project costs. COCOMO II allows ERP teams to include in their estimates (i) the maturity level of the ERP adopting organization, (ii) the extent to which requirements' and system architecture's volatility is reduced before ERP configuration, and (iii) the level of team cohesion and stakeholders' participation. In COCOMO II, the degrees of both the scale factors and the cost drivers vary from extra low, very low, low and nominal to high, very high and extra high. Suppose ERP project stakeholders assign a degree to each scale factor and cost driver, the estimation of project effort and duration will result from the two equations below:

$$\text{Effort} = A \times (\text{Size})^E \times \prod_{i=1}^{17} EM_i \quad (1)$$

$$\text{and Time} = C \times (\text{Effort})^F \quad (2)$$

where E and F are calculated via the following two expressions, respectively:

$$E = B + 0.01 \times \sum_{j=1}^5 SF_j \text{ and}$$

$$F = D + 0.2 \times (E - B)$$

In (1) and (2), *SF* stands for the scale factors, and *EM* means cost drivers.

3.2 The Monte Carlo Simulations

To obtain more realistic estimates, we approached the inherent uncertainty of the cost drivers by applying NOSTROMO [19], a Monte Carlo simulation technique used at the THAAD Project Office (USA). This is a problem-solving technique used to approximate the probability of certain outcomes by running multiple trial runs, called simulations, using random variables. When used in combination with COCOMO II, repeatedly running the model many times and collecting samples of the output variables for each run helps the estimation analysts produce an overall picture of the combined effect of different input variables distribution on the output of the model.

3.3 The Portfolio Management Concept

We couple the above techniques with a portfolio management concept [12] based on an effort-and-deadline-probability model that allows us to quantify the uncertainty associated with a project estimate. We chose it because (i) it is applicable at the stage of requirements or project bidding [12], (ii) its only input requirement is a record of previous projects; although it does require an effort estimate for every project, it need be nothing more sophisticated than a subjective opinion [12]; and (iii) it fits with the ERP adopters' project realities suggesting that an ERP project is implemented as a portfolio of interdependent subprojects [5,6]. Each subproject is a piece of functionality (or an ERP module) linked to other pieces (or modules). For example, the Sales and Distribution module in a package is tightly linked with the Accounts Receivable

and Profits Center Reporting functionality of the Financial Accounting and Controlling modules [22]. Suppose we have a set of interdependent subprojects, the effort-and-deadline-probability model [12] will yield (i) the probability of portfolio's success with the proposed deadlines for each subproject in this portfolio, and (ii) a set of new deadlines which will result in a required probability of success. The portfolio success is judged by two conditions applied to any two subprojects *a* and *b* for which $deadline_a$ is earlier than $deadline_b$. The conditions are that: (i) subproject *a* is to be over by $deadline_a$ and (ii) subproject *a* and subproject *b* are to be over by $deadline_b$. In other words, the conditions require all subprojects planned with a deadline before $deadline_b$, to be completed by $deadline_b$, rather than just project *b*. This is the key to the portfolio approach, because uncertainty about completion of project *b* incorporated uncertainty from all previous projects.

Suppose the ERP adopter engages in total *E* people in the project and let *d* be the number of work days it takes from start date to deadline, then the total available resources is *E**x**d*. So, suppose an ERP portfolio *Y* is made up by *n* subprojects, the success conditions are represented as follows:

$$\begin{pmatrix} Y_1 \\ Y_1 + Y_2 \\ \dots \\ Y_1 + Y_2 + \dots + Y_n \end{pmatrix} \leq E \begin{pmatrix} d_1 \\ d_2 \\ \dots \\ d_n \end{pmatrix} \quad (3)$$

where Y_i is the estimated effort for subproject *i* to succeed. We check if, for any *j*, (*j*=1..*n*), the sum of Y_1, \dots, Y_j is greater of *E**x* d_j . If this is true, then deadline d_j has failed. Success probabilities result from simulations in which Y_1, \dots, Y_n are generated from a predetermined probability distribution. If we deem Y_1, \dots, Y_n is satisfying all conditions, then we say that the portfolio *Y* succeeds. The portfolio's probability of success is equal to the ratio of the number of successes in the set *Y* to the number of trials in the simulation.

3.4 How It Fits Together?

Our solution approach consists of eight steps which are presented in Figure 1. Because we designed our approach with the RE stage in mind, we suggest **Unadjusted Function Points (FP)** [5] be used as a size estimate. This is consistent with the position of the COCOMO II authors [2, page 17]. We chose this measure of functional size because (i) it is applicable to any ERP package and not to a specific package's context [5] and (ii) it's the only measure of size that fits the project stage of early requirements. Furthermore, to account for uncertainty of the ERP project context, we suggest the COCOMO II model take as inputs the probability distributions of the five COCOMO scale factors and 17 cost drivers, instead of using as inputs single values (as in [2]). This design choice has been recommended by the THAAD Project Office [19] and by the JLP NASA researchers as well. Deploying the Monte Carlo simulation manes to ascribe a particular distribution type to an input variable in a model, get randomly-selected values, feed them into the COCOMO II model and, then, see how likely each resulting outcome is. In other words, for each uncertain factor, our

approach yields possible effort and duration estimation values. In contrast to CO-COMO II, our output is the probability distributions of effort and duration and not the most likely effort and duration (which COCOMO II creates).

The probability distributions are, then, fed into the portfolio management method [12]. To run it, we first formulate a condition for success, as in (3), then we bunch projects into portfolios and we obtain the probability of successfully delivering the projects under time constraints as well under effort constraints.

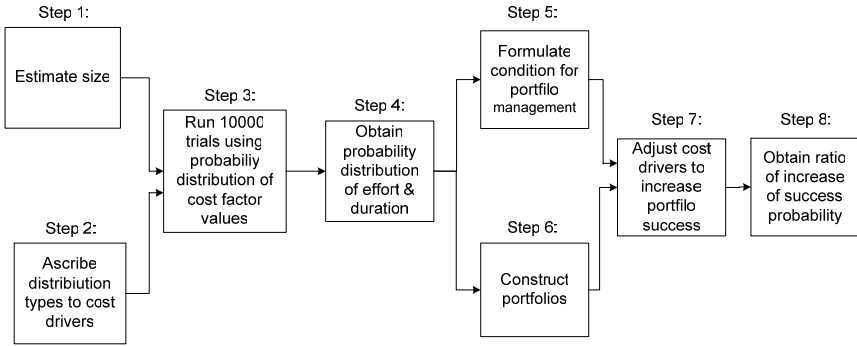


Fig. 1. The solution approach: a high-level view

4 The Case Study

The research methodologist R. Yin [31] makes the note that a case study has a distinct advantage when a ‘how’ or ‘why’ question is being interrogated about a contemporary set of events over which the researcher has little or no control. In software engineering, case studies are also useful in answering a “which is better” question [30] and, here, this is what we are after. Below, we describe the application of our approach and state our expectation of it (Section 4.1) and the results we obtained (Section 4.2)

4.1 Application of the Method

The solution approach was applied in a setting of a large organization-wide ERP roll-out that included eight functional modules of one ERP package (namely SAP) and covered three locations of a North American telecommunication company [5]. The modules were: Material Management, Sales and Distribution, Service Management, Accounts Payable, Accounts Receivable, Plant Maintenance, Project System, and Asset Management. Our data came from 13 SAP projects implemented in the case study company. The projects were carried out between November, 1997 and October, 2003. In this period, the author was employed by the case company as a SAP process analyst and was actively involved in the projects. The ERP implementation process model adopted in the context of the projects was the AcceleratedSAP (ASAP) RE process [22]. It is a project-specific process, engineered and standardized by SAP, and

provided to clients by ASAP-certified consulting partners. The ASAP process has been extensively elaborated in [22]. The practical settings for our 13 projects have been described in detail in [5]. They included the following: to manage implementation complexity, each of our projects was broken down in a number of subprojects reflecting the number of components to be configured. For example, the first project had to implement six components and was broken down in six subprojects. The total number of our subprojects in which the standard ASAP process was instantiated was 67. For each subproject, there was a dedicated RE team. This is a group of individuals who are assigned to a specific subproject, contribute time to and run the RE cycle for this subproject, and deliver the business requirements document for a specific SAP component. Each RE team consisted of one or two SAP consultants who provided in-depth knowledge in both the ASAP implementation process and the SAP components, and a number of business representatives, the so-called process owners. They were department managers and subject matter experts who contributed the necessary line know-how, designed new processes and operational procedures to be supported by the SAP modules, and provided the project with the appropriate authority and resources. All process owners had above average level of experience with IT-projects in their departments and, before starting the projects, attended a three-hour training session on the ASAP process. Next, we considered our consultants as an even mix of experts, new hires and novices. Each expert had at least 5 years of configuration and integration experience with a specific SAP functional module. Most experts had ASAP RE experience. Our consulting partners provided evidence that their less experienced staff-members completed the standard training courses on both the ASAP process and the corresponding SAP modules. However, none of the consultants had any experience in the telecommunication sector; they were unaware of the requirements principles in this domain and were supposed to carry out RE activities under novel and challenging conditions. All the teams were supported by a process architect responsible for architecting the solution, sharing process knowledge and consulting on ongoing basis with the teams on SAP reuse, process methods, and RE tools. The architect was the only resource the teams shared. The 67 teams worked separately and with relatively little communication among them. This allowed us to initially consider and include 67 subprojects in our case study.

For each of the 13 projects, we got (i) project size data, (ii) reuse levels, (iii) start and end dates, and (iv) scale factor and cost driver ratings. Size was measured in terms of unadjusted IFPUG FP [6]. Reuse levels were formed by using a reuse indicator that included reused requirements as a percentage of total requirements delivered [5]. Next, the effort multipliers A , B , and EM in equation (1) and (2) and the scale factors SF were calibrated by using ERP effort data collected between 1997 and 2004 in the case study company.

Because we had the ratings of the cost drivers and scale factors only and no knowledge about the uncertainty of the ratings, we assigned to each factor its distribution type and its parameters of probability distribution (namely center, min and max)

based on previously published experiences and recommendations by other authors [15,19]. For example, this case study used McDonald's [19] default 'high' levels of uncertainty associated to the ratings of the RESL, DATA, ACAP and PCAP cost drivers [2]. (Because of space limitation, we refer readers to reference [2] which gives detailed definitions of these cost drivers). The level of uncertainty determines - in turn, the distribution type to be assigned to each cost driver: normal, triangular, and uniform for low, medium and high uncertainty, respectively.

We also opted to use a lognormal distribution for functional size, which was motivated by the observations of Chulani et al [3]. These researchers investigated the size distribution and indicate that its skew is positive and that log(size) is likely to be a normal distribution.

With this input data (namely, the COCOMO II factors and uncertainty values), we run Monte Carlo simulations which gave us samples of (i) effort, expressed in person-month, and (ii) time, expressed in months. Generally, a Monte Carlo simulation consists of many - often thousands of, trials, each of which is an experiment where we supply numerical values for input variables, evaluate the model to compute numerical values for outcomes of interest, and collect these values for later analysis. In this case study, we used 10000 trials and generated the samples of effort and time, as presented in Figure 2 and Figure 3, respectively. In these histograms, the Y-dimension shows the frequency with which a value was observed in the sample of 10000 trials. The X-dimension shows the value range. Because the average subproject involved four professionals (two business users, one external consultant and one internal IS team members), we adopted the assumption for E to be 4.

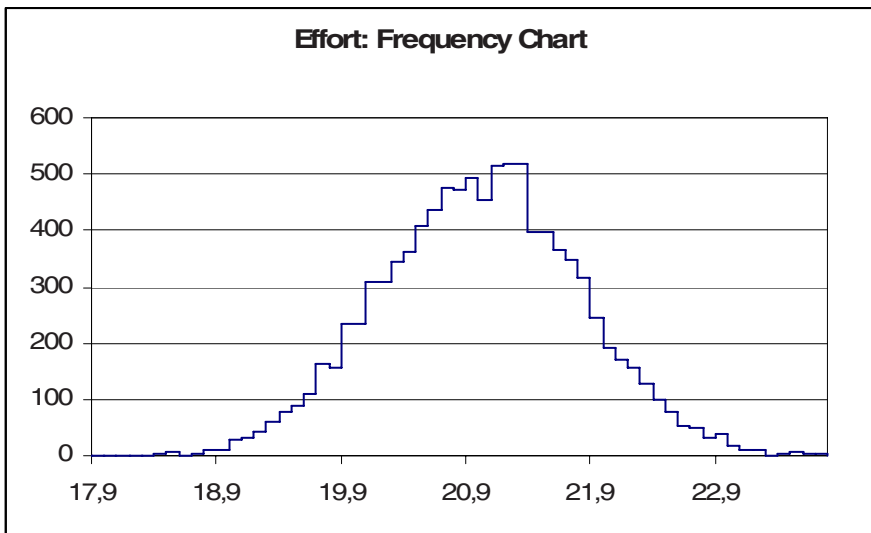


Fig. 2. The Monte Carlo histogram of the probability distribution of effort (in person/months)

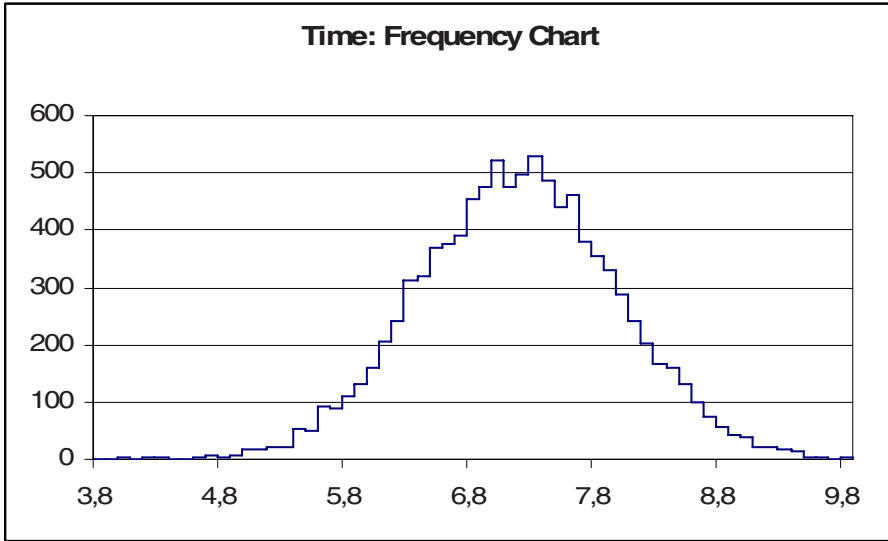


Fig. 3. The Monte Carlo histogram of the probability distribution of time (in months)

Based on the observation that COCOMO II provides time estimation as in (2), we formulated the following condition for portfolio management in terms of time constraints:

$$\begin{pmatrix} T_1 \\ T_1 + T_2 \\ \dots \\ T_1 + T_2 + \dots T_n \end{pmatrix} \leq \begin{pmatrix} m_1 \\ m_2 \\ \dots \\ m_n \end{pmatrix} \tag{4}$$

where T_i is the ERP implementation time in months for subproject i . In this condition, we did not include the number of people E , because COCOMO II assumed an average number of project staff [2] which was accounted in (2). Furthermore, as recommended in [15], we attempted to improve the chances for portfolio success by adjusting the cost drivers and scale factors. Hence, we adopted the assumption that for projects with two different ratings for the same factor, the probability of success for each project will be different too. Finally, our case study plan included assessment of how much the probability of success increased when treating ERP projects as a portfolio. We expected that the subprojects with high uncertainty ratings would benefit more from portfolio management, than the projects with low uncertainty ratings would do.

4.2 Results

This sections reports on the results with respect to: (i) what we observe when adjusting COCOMO II cost drivers, and (ii) what we learnt from the probability of success of highly-uncertain projects when managing them as a portfolio.

To understand how cost drivers and scale factors make a difference in terms of project success, for each one of them we constructed two portfolios: the first one had this driver/cost factor rated ‘very high’ for all projects and the second portfolio had it rated ‘very low’ for all projects. For example, we found that when selective reuse [6] was practiced in ERP projects, the probability of success was higher under both time and effort constraints. For the purpose of illustrating this point, we report on the results (see Table 1) yielded when constructing two portfolios of subprojects, namely the first one with the factor of REUSE rated as *very high* for all subprojects and the second one with REUSE rated very low for all subprojects. We make two notes: First, that *low* level of reuse in an ERP project indicates massive customization of the standard components and that a high level of reuse indicates limited customization [5]. Second, we ruled out the rating ‘extremely high’ as it’s relatively rarely to be observed in a ERP project context [6,10]. Table 1 suggests that when a project is composed of subprojects all of which have REUSE rated *very high*, the probability of success is greater under both time and effort constraints.

Table 1. Analysis of the probability of success for the factor REUSE under effort constraints and time constraints

REUSE rating	Probability of success	
	Under effort constraints	Under time constraints
Very low	68.78%	76.52%
Very high	96.87%	98.88%

We observed that 13 out of the 17 factors from the COCOMO II model can be adjusted in a way that maximizes the probability of success. These 13 factors are: database size (DATA), product complexity (CPLX), REUSE, documentation (DOCU), platform volatility (PVOL), analyst capability (ACAP), programmer capability (PCAP), personnel continuity (PCON), applications experience (APEX), language and tool experience (LTEX), use of software tools (TOOL), multi-site implementation (SITE), required implementation schedule (SCED).

Regarding our second group of results, our observations suggest that bundling ERP projects as a portfolio had the advantage over managing projects separately in terms of ability to explicitly and systematically approach uncertainty. We compared the probability of success for projects under effort constraints and for projects under time constraints, respectively (Table 2 and Table 3). They indicate that the probabilities of success for projects with high uncertainty ratings are greater when those projects are managed as a portfolio.

Table 2. Increase in probability of success for low and high uncertain projects under effort constraints

Uncertainty level	Probability of success		Ratio of increase (a)/(b)
	Individual projects (a)	Portfolio (b)	
Low uncertainty	93.78%	98.81%	1.05
High uncertainty	84.31%	97.76%	1.16

Table 3. Increase in probability of success for low and high uncertain projects under time constraints

Uncertainty level	Probability of success		Ratio of increase (a)/(b)
	Individual projects (a)	Portfolio (b)	
Low uncertainty	15.76%	87.52%	5.55
High uncertainty	8.31%	75.91%	9.13

5 Evaluation of Validity Concerns

We did obviously a preliminary step only towards a better understanding of the major phenomena that cause uncertainty in ERP effort estimation. To this end, we could only say that we need to carry out a few replication studies so that the findings of this study can be consolidated and transformed into recommendations to ERP project managers. As per the recommendation of research methodologists [30,31], we did an early assessment of the following validity [30] threats:

First, the major threat to external validity arises from the fact that the company’s projects might not be representative for the entire population of ERP adopters. We however, believe that our project context is typical for the telecommunication companies in North America: we judge these settings typical because they seemed common for all SAP adopting organizations who were members of the American SAP Telecommunications User Group (ASUG). The ASUG meets on regular basis to discuss project issues and suggest service sector-specific functionality features to the vendor for inclusion in future releases. The SAP components our case company implemented are the ones which other ASUG companies have in place to automate their non-core processes (accounting, inventory, sales & distribution, cell site maintenance).

Second, when constructing the portfolio, the author based her choice of ‘very low/very high’ ratings on her own experience in implementing ERP. While for some drivers, as reuse, the author did research on what reuse levels are achievable in an ERP project [6], for others the author set up the ratings in a way that - clearly, could be subjective. However, this design choice was the only possible way to go, given the fact that, to the best of our knowledge, there is no published research on the CO-COMO II factor ratings which are more common in ERP context. We plan, in the future, to research the topic of economies and diseconomies of scale in ERP projects, hoping that new knowledge will help refine our approach.

Next, we deployed complementary three models of three types. However, we are aware that there are other promising effort estimation modeling techniques by each type. For example, there is a number of approaches using portfolio concepts [29] which might be good candidates for the ERP settings. In the future, we are interested in investigating whether different modeling choices sustain our results or limit the validity of our findings to the subset of the analyzed models.

6 Conclusions

In this case study we have demonstrated that the complementary use of Monte Carlo simulation, a portfolio management method and a parametric empirical model

(COCOMO II) can help counterpart the uncertainty in early ERP effort estimation based on business requirements. The ultimate objective of the approach is to ensure that setbacks in some ERP implementations are balanced by gain in others. Our approach is positioned to leverage off the current body of knowledge in both software economics and ERP RE. The targeted effect was to systematically cope with two aspects inherent to ERP project contexts: (i) lack of ERP-adopter's specific historical information about the context and (ii) strong bias by outsourcing partners and ERP consultants in cost estimation. We found this approach to be one good alternative to ERP-adopters as they no longer have to live with whatever estimates are given to them by ERP consultants.

The case study provided evidence that led us to conclude the following:

- (i) when managed as a portfolio, highly-uncertain ERP projects have a greater chance to succeed under time and under effort constraints,
- (ii) subprojects with high uncertainty ratings would have greater advantages from portfolio management than projects with low uncertainty ratings would do.
- (iii) it's possible to adjust cost drivers so that one increases the probability of success for highly uncertain ERP projects, a company might have to implement. We have also shown that 13 out of the 17 COCOMO II cost drivers can be adjusted to increase the chances for success.

With respects to (1) and (2), our results agree with the observation by Jiamthubthugsin and Sutivong [15] who experimented with the portfolio management method in the context of custom projects. Though, we must acknowledge (i) that we have preliminary results only and (ii) that related validity concerns [30] remain our most important issue. In the next six months, we will work in collaboration with three European companies to carry out a series of experiments and case studies to test our approach. The results will serve to properly evaluate its validity and come up with an improved version of our method.

Acknowledgements

The author thanks the anonymous reviewers for their comments and suggestions, which greatly improved the clarity and value of this work. The author also thanks the following organizations without whose support this research program would not have become a reality: the Netherlands Science Organization (NWO) for supporting the CARES project and the QuadREAD project, CTIT for supporting the COSMOS project, IFPUG, NESMA and ISBSG, for making their resources available to me, and the QuadREAD industry partners for the stimulating discussions that helped make sure this research remained industry-relevant.

References

- [1] Arnesen, S., Thompson, J.: How to Budget for Enterprise Software, Strategic Finance, January 2005, pp. 43–47 (2005)
- [2] Boehm, B.: Software Cost Estimation with COCOMO II. Prentice Hall, Upper Saddle River (2000)

- [3] Chulani, S., Boehm, B., Steece, B.: Bayesian Analysis of Empirical Software Engineering Cost Models. *IEEE Trans on SE* 25(4), 573–583
- [4] Cisco Systems, How Cisco Upgraded Their Purchasing, http://www.cisco.com/web/about/ciscoitnetwork/business_of_it/erp_purchasing.html
- [5] Daneva, M.: ERP Requirements Engineering Practice: Lessons Learnt. *IEEE Software* 21(2), 26–33 (2004)
- [6] Daneva, M.: Measuring Reuse of SAP Requirements: a Model-based Approach. In: *Proc. of Symposium on Software Reuse*. ACM Press, New York (1999)
- [7] Daneva, M., Wieringa, R.J.: A Requirements Engineering Framework for Cross-organizational ERP Systems. *Requirements Engineering Journal* 11, 194–204 (2006)
- [8] Daneva, M.: Approaching the ERP Project Cost Estimation Problem: an Experiment. In: *Int'. Symposium on Empirical Software Engineering and Measurement (ESEM)*, p. 500. IEEE Computer Society Press, Los Alamitos (2007)
- [9] Daneva, M.: Preliminary Results in a Multi-site Empirical Study on Cross-organizational ERP Size and Effort Estimation. In: *Proc of the Int. Conf. on Software Process and Product Measurement (MENSURA)*, Palma, Spain, pp. 182–193. UIB Press (2007)
- [10] Davenport, T.: *Mission Critical: Realizing the Promise of Enterprise Systems*. HBS Press (2000)
- [11] Fenton, N.E., Pfleeger, S.L.: *Software Metrics: A Rigorous and Practical Approach*, 2nd edn. PWS Publishing (1998)
- [12] Fewster, R.M., Mendes, E.: Portfolio Management Method for Deadline Planning. In: *Proc. of METRICS 2003*, pp. 325–336. IEEE, Los Alamitos (2003)
- [13] Glas, R.L.: *Facts and Falacies of Software Engineering*, p. 58. Perason Education, Boston
- [14] Hansen, T.: Multidimensional Effort Prediction for ERP System Implementation. In: Meersman, R., Tari, Z., Herrero, P. (eds.) *OTM 2006 Workshops*. LNCS, vol. 4278, pp. 1402–1408. Springer, Heidelberg (2006)
- [15] Jiamthubthugsin, W., Sutivong, D.: Profolio Management of Software Development Projects Using COCOMO II. In: *Proc. of ICSE 2006*, pp. 889–892 (2006)
- [16] Kitchenham, B.A., Pickard, L., Linkman, S., Jones, P.: Modelling Software Bidding Risks. *IEEE Transactions on Software Engineering* 29(6), 542–554 (2003)
- [17] Koch, S.: ERP Implementation Effort Estimation Using Data Envelopment Analysis. In: Abramowicz, W., Mayr, H.C. (eds.) *Technologies for Business Information Systems*, pp. 121–132. Springer, Dordrecht (2007)
- [18] Luo, W., Strong, D.M.: A Framework for Evaluating ERP Implementation Choices. *IEEE Transactions on Engineering Management* 5(3), 322–333 (2004)
- [19] McDonald, P., Giles, S., Strickland, D.: Extensions of Auto-Generated Code and NOSTROMO Methodologies. In: *Proc. of 19th Int. Forum on COCOMO*, Los Angeles, CA
- [20] Parthasarathy, S., Anbazhagan, N., Evaluation, E.R.P.: Implementation Choices Using AHP. *International Journal of Enterprise Information Systems* 3(3), 52–65 (2007)
- [21] Rettig, C.: The Trouble with Enterprise Systems, *Sloan Management Review*. Fall 49(1), 21–27 (2007)
- [22] SAP AG, *ASAP Methodology for Rapid R/3 Implementation: User Manual*, Walldorf (1999)
- [23] Stamelos, I., Angelis, L., Morosio, M., Sakellaris, E., Bleris, G.: Estimating the Development Cost of Custom Software. *Information & Management* 40, 729–741 (2003)
- [24] Stensrud, E.: Alternative Approaches to Effort Prediction of ERP Projects. *Inf.&Soft Techn.* 43(7), 413–423 (2001)
- [25] Stensrud, E., Myrvtveit, I.: Identifying High Performance ERP Projects. *IEEE Trans. Software Engineering* 29(5), 398–416 (2003)

- [26] Summer, M.: Risk Factors in Enterprise Wide Information Systems Projects. In: Special Interest Group on Computer Personnel Research Annual Conference Chicago, Illinois, pp. 180–187
- [27] Vogelesang, F.: Using COSMIC FFP for Sizing, Estimating and Planning in an ERP Environment. In: Int'l Workshop on Software Measurement, Potsdam, pp. 327–342. Shaker Publ. (2006)
- [28] Vogezang, F.: Application Portfolio Management: How much Software Do I Have? In: Proc. of the Software Measurement Forum (SMEF), Italy (2007)
- [29] Verhoef, C.: Quantitative IT Portfolio Management. *Science of Computer Programming*, vol. 45(1), pp. 1–96 (2002)
- [30] Wohlin, C.: *Experimentation in Software Engineering*. Springer, Heidelberg (2000)
- [31] Yin, R.: *Case Study Research, Design and Methods*, 3rd edn. Sage Publications, Newbury Park (2002)

The Effect of Entity Generalization on Software Functional Sizing: A Case Study

Oktay Turetken^{1,*}, Onur Demirors¹, Cigdem Gencel², Ozden Ozcan Top¹,
and Baris Ozkan¹

¹ Informatics Institute, Middle East Technical University, 06531, Ankara, Turkey
{oktay, demirors, ozden, bozkan}@ii.metu.edu.tr

² Blekinge Institute of Technology, Department of Systems and Software Engineering
cigdem.gencel@bth.se

Abstract. In this paper we discuss a specific result derived from a multiple case study. The case study involved implementation of IFPUG Function Point Analysis and COSMIC Functional Size Measurement methods in an innovative military software development project by different groups of experts. Application of these methods in a case that provides a number of size measurement challenges enabled us to observe significant improvement opportunities for the methodologies. In this paper, we depict the utilization of the entity generalization concept in two FSM methods and based on our observations we discuss the effects of different interpretations of the concept for measuring the software functional size.

Keywords: Functional size measurement, COSMIC FSM, IFPUG FPA, entity generalization.

1 Introduction

Poor estimation remains to be one of the main reasons for software project failures. Functional Size Measurement (FSM) methods are advocate for providing necessary input for estimation models. FSM methods are intended to measure the size of software by quantifying the functionality delivered to the user. Since the introduction of the concept [2], a variety of FSM methods have been formulated and many improvements have been made on these methods [11].

FSM methods have their own definition of functionality, utilize different counting rules for the different functional components of functional user requirements and have their own units and scales for their measures. In spite of these differences, they are expected to produce similar results as they are based on a set of shared principles. A number of research studies have been performed in order to clarify their conceptual basis and establish the common principles of FSM methods [8], [10], [13].

* This study is supported by The Scientific and Technological Research Council of Turkey (TUBITAK), Project 107E010.

The objectives of this paper are to discuss how the concept of entity generalization is considered in commonly used FSM methods; the International Function Point Users Group Function Point Analysis (IFPUG FPA) [17] and the Common Software Measurement International Consortium FSM (COSMIC FSM) [18] and to investigate how different interpretations of this concept affect the functional size of the software measured by these methods. Findings are based on the case study we conducted on an innovative military software development project. Specifically, we observe how entity abstractions - in the form of inheritance or generalization/specialization between entities or classes - may lead to different assumptions when identifying elementary components for the measurement and the effects of these different assumptions on the functional size. We evaluate the methods based on our findings and discuss the difficulties we faced during the implementation of the methods.

The paper is organized as follows: Section 2 briefly summarizes the FSM methods and related research. In section 3, we describe the case study. Section 4 presents our findings and conclusions.

2 Related Work

Measuring the software size with the ‘functionality’ attribute was first introduced by Albrecht [2] in his Function Point Analysis (FPA) method. With the refinements of the technique, FPA has evolved into the IFPUG FPA [12] method. During the following years, several new measurement methods ([1], [5], [22], [29]) or extending the applicability of FSM methods to different functional domains in addition to business application software ([21], [28], [30]) have been developed. Studies by Symons’ [26] and Gencel et al. [11] provide detailed discussions on FSM methods.

The publication of the first ISO/EIC’s 14143-1 standard [14] in 1998 aimed at clarifying the fundamental concepts of FSM. It defined concepts such as ‘Functional User Requirements (FUR)¹’, ‘Functional Size²’, Base Functional Component (BFC)³’, ‘BFC Type⁴’ and the FSM requirements that should be met by a candidate method. Currently, Mark II FPA (MkII FPA) [16], IFPUG FPA [17], COSMIC FSM [18] and Netherlands Software Metrics Association FSM (NESMA FSM) [19] are certified by ISO as being international standards.

Earlier FSM methods have been criticized of lacking support for concepts such as inheritance and aggregation [1], [7], [24], which are generally associated with object-oriented methodologies. This creates ambiguities and difficulties in determining the functional components and measuring the functional size.

In order to better reflect the needs of object-oriented (OO) software development methodologies, several approaches have been proposed. Some of these works that

¹ FURs: A sub-set of the user requirements. The FURs represent the user practices and procedures that the software must perform to fulfill the users’ needs.

² Functional Size: A size of the software derived by quantifying the FUR.

³ BFC: An elementary unit of FUR defined by and used by an FSM Method for measurement purposes.

⁴ BFC Type: A defined category of BFCs. A BFC is classified as one and only one BFC Type.

adapt FPA method to OO concepts yield results that are similar to what would have been obtained by directly applying IFPUG FPA. Whitmire [31] considered each general class as a logical file and methods sent across the application boundary as transactional functions. For classes that are part of an inheritance hierarchy, “if the generalization is truly part of the application domain, it is counted as a separate logical file”. If the generalization was build for the ease of modeling, general class is counted with each specialized class. In IFPUG FPA, a logical file is a user identifiable group of logically related data or control information. Internal Logical Files (ILFs) are maintained within the boundary of the application, whereas External Logical Files (EIFs) are maintained within the boundary of another application.

Fetcke et al. [7] defined rules for mapping OO-Jacobson method to concepts from IFPUG FPA and verified the rules by applying them in three case studies. For inheritance relationships, they defined two rules. First; an *abstract* class is not visible to the user and does not relate logical files itself. It is rather a candidate for a record element type (RET) for each class that inherits its properties. RETs are optional or mandatory *subgroup* of data elements within an ILF or EIF. They influence the degree of functional complexity (low, average, high) of logical files. Second; specialized classes of a *concrete* general class are candidates for a logical file or a RET of that class. With these presumptions, however, the work does not elucidate whether specialized classes are logical files of their own or RETs for the general class.

To overcome these difficulties Abrahao et al. [1] proposed OO-Method Function Points (OoMFP) for measuring the functional size of OO systems which is compliant with the IFPUG FPA rules.

Similarly, Caldiera et al. [3] adapted IFPUG FPA rules for measuring object oriented analysis and design specifications. They proposed alternative ways for identifying logical files and handling entity abstractions, but did not propose clear rules or recommendations of when and under what conditions each can be applied. In an inheritance hierarchy, a logical file may comprise all the entities in the hierarchy or every entity can be mapped to a logical file.

Mapping of object oriented modeling concepts onto the measurement constructs has also been studied for COSMIC FSM. Jenner [20] proposed a mapping for the concepts used in UML diagrams onto the abstract COSMIC FSM model. It is argued that UML sequence diagrams have a more appropriate level of granularity to measure functional size. Diab et al. [6] proposed a set of formal rules for applying COSMIC FSM to object-oriented specifications. The work proposes a formalization of the COSMIC FSM measure for the real-time object oriented modeling language.

3 The Case Study

We conducted a multiple-case study in order to evaluate and explore the similarities and differences between FSM methods. By implementing IFPUG FPA [12] and COSMIC FSM [28] methods, different measurers measured the functional size of a case project.

In the scope of this paper, we deal only with the differences among methods in handling the entity generalization and how these affect the measurement results between the functional size figures obtained by different measurers. Therefore, although the description of the whole case study is presented; in this paper, the results which are related to these specific questions on entity generalization are discussed. It should be noted that our aim here is not to find out which of the methods is better, but to shed light into the improvement opportunities of each of these methods.

3.1 FSM Methods Utilized

In general, FSM methods first requires the functional user requirements (FUR) to be decomposed into ‘Transactions’, which involve inputting, outputting and processing of items or groups of data, triggered by events outside the software [15]. From transactions, BFCs are identified and then each of these is categorized to BFC Types and the attributes relevant for obtaining the base counts are identified. The next step is the actual measurement where the functional size of each BFC is measured by applying a measurement function to the BFC Types and the related attributes. The overall functional size of the software system is computed by summing up the results.

In IFPUG FPA, the BFCs are classified as the Transactional Function (TF) Types and Data Function (DF) Types. DF may be an Internal Logical File (ILF) or an External Interface File (EIF), whereas a TF can be of the type; External Input (EI), External Output (EO), or External Inquiry (EQ). These components are weighted according to their complexity and their weights are summed. The functional complexity of each logical file is based on the number of record element types (RETs) (subgroup within a logical file) and the number of data element types (DETs) within a logical file. A DET is a unique user recognizable, non-repeated field which is equivalent to the ‘entity attribute’. The functional complexity of a logical file can be low, average or high, each corresponding to an IFPUG function point value.

In COSMIC FSM each FUR is decomposed into ‘Functional Processes’ (FP) and each of these FPs is assumed to comprise a set of sub-processes, called Data Movement Types. Data movement types are the BFCs of this method. A data movement moves one or more data attribute belonging to a single ‘data group’, where each included data attribute describes a complementary aspect of the same ‘object of interest’. An object of interest is any ‘thing’ or a conceptual object that is identified from the point of view of the Functional User Requirements. It is equivalent to ‘entity-type’ in entity relationship (ER) analysis or ‘class’ in UML [23]. There are four kinds of Data Movement Types: Entry (E), Exit (X), Read (R), and Write (W). Each of these is defined as a BFC Type [18]. The value in COSMIC FP is the total number of data movements performed in the software system.

A detailed discussion on the differences and similarities between these two methods can be found in [10].

3.2 Description of the Case Project and the Software Application

The case project involved the development of a conceptual modeling tool (KAMA) that provides a common notation and a method for the conceptual model developers

in different modeling and simulation development projects particularly in the military domain. The project was started in June 2005 and completed in July 2007.

The total number of project staff worked consisted of 21 people; 1 project manager, 1 assistant project manager, 2 steering committee members, 1 project coordinator, 8 researchers, 1 software development team leader, 1 quality assurance team leader, 4 software engineers (1 part-time), 1 part-time test engineer and 2 quality engineers (1 part-time). The efforts utilized for the project totaled up to 1,832 person-days. Table 1 gives the details of the efforts utilized for the tool development part of the project.

Table 1. The development effort for the case project

Software Development Life Cycle Phase	Effort (person-days)
Development Processes	1,287
Software Requirements Analysis	227
Software Design	185
Software Coding & Unit Testing	670
Testing	205
Management	135
Supporting Processes	410
Total	1,832

The types of software tools and programming languages used in the development phases were as follows: Rational Software Architect as the software analysis and design tool, Requisite Pro as the requirements management tool, and C# as the programming language. Unified Modeling Language (UML) [23] was used for representing analysis and design. Related IEEE standards were utilized for the project work products, which were kept under configuration control by the Subversion tool.

With respect to CHAR Method defined in [15], the functional domain of the KAMA is determined as 'Information System'.

KAMA is a graphical modeling tool that supports a specific notation based on UML. It supports the development of conceptual models with a set of diagrams, model elements and their relationships. Each diagram simply consists of a set of model elements and the relationships between them. The type of model elements in a diagram and the type of the relationships that can exist between them is determined by the diagram type. The notation comprises 8 diagram types, 10 model element types and 15 relationship types. The diagram entity has a common set of attributes maintained for all types. For the model elements, on the other hand, together with the common attributes that are maintained by all, there exist attributes specific to types. Similar situation also holds for relationship types. Fig. 1 depicts the model element, relationship and diagram entities and partial data model for the entity abstractions with an extended entity-relationship (EER) model.

The characteristics of the data entities to be maintained by the tool make it a good candidate for generalization/specialization practices to be applied for model element, diagram and relationship entities.

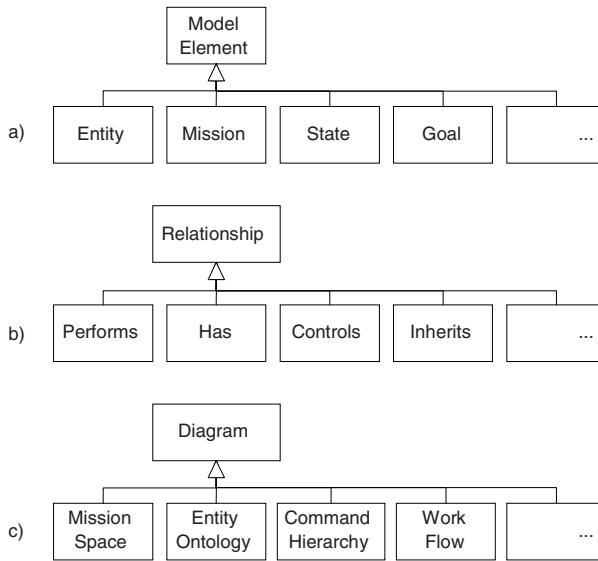


Fig. 1. Entity generalizations for diagrams, model elements and relationships

3.3 Case Study Conduct

The functional size measurement of the KAMA was performed by IFPUG FPA and COSMIC FSM based on the software requirements specification (SRS) document. Together with the requirements statements, the document included UML use case diagrams, activity diagrams describing the details of the use cases and a data model in the form of a simple class diagram.

The measurements were performed independently by two groups of measurers, each of which involved two measurement experts. The measurements were performed by different groups to better understand potential measurement variances caused by assumptions and interpretations of the different measurers. The results were verified according to the measurement rules of each method by the measurer who himself did not involved in that measurement process. In the group of measurers, one of them holds a PhD. degree in the related subjects; two are PhD. students and one is an MSc. student working on related subjects, in particular on software functional size measurement. All the measurers received training for at least one of the FSM methods and they all measured at least one project previously.

The measurement results are given in Table 2 and Table 3. Since ISO view takes the adjustment of the functional size via quality and technical requirements outside the scope of the FSM [13], we do not take into account the adjustment phase of the IFPUG FPA for the purpose of this case study.

It took 105 person-hours of effort to measure the functional size of KAMA implementing COSMIC FSM. The measurement with IFPUG FPA took 102 person-hours. Although the functional size for each of the method differs significantly, the effort values utilized for the measurement were quite similar.

Table 2. Case Project - IFPUG FPA Size Measurement Details

No. of Elementary Processes	No. of ILFs	No. of EIFs	No. of EIs	No. of EOs	No. of EQs	Functional Size (IFPUG FP) (Unadjusted)
45	11	0	26	1	18	306

Table 3. Case Project - COSMIC FSM Size Measurement Details

No. of Functional Processes	No. of Entries	No. of Exits	No. of Reads	No. of Writes	Functional Size (COSMIC FP)
55	61	154	314	160	697

4 Findings and Conclusions

Although the rules for identifying the BFCs and BFC types differ for each method, using similar concepts and comparable attributes, decomposition of the functional user requirements into ‘transactions’ is expected to yield the same set of transactions. While this is not explicitly asserted by any of the FSM methods, it is one of the underlying assumptions of research related with the conversion of the sizes or unification of these methods [8], [4], [10], [25]. When the same groups of measurers involve in the measurement process, they usually identify similar or identical transactions for measurements performed by different FSM methods [10], [11].

However, in this case study, the differences in the interpretation of the rules by different measurers cause significant differences in the measurement results. During the case study conduct, the measurers faced some difficulties in identifying the entities and transactions. This was mainly due to the structure of the data to be maintained by the application and the way the FSM methods handles entity generalization.

IFPUG Implementation Results. Although IFPUG FPA measurement process does not give any precedence rule for identifying the data and transactional functions, in our case study we started with the data functions. Because, we utilized ILFs to better identify the transactional functions and valuing their complexity. The complexity of a transactional function is dependent on the number of ILFs/EIFs maintained during the transactions as well as the total number of input and output DETs.

IFPUG FPA takes the complete inheritance hierarchy as a single *logical file*, with a RET for each subclass [12]. For example, the complete inheritance hierarchy of ‘model elements’ is considered as one ILF with a number of RETs for each special type (Fig. 2). Thus, with respect to the counting rules, the functional complexity of the model element ILF is *high* and so the contribution on the total functional size is 15 IFPUG function points (FP). The affect of number of RETs on functional size were limited in the sense that, with 10 RETs for each of the special model element type having attributes of their own, the contribution of the ILF is increased from 7 to 15 function points (complexity level from low to high).

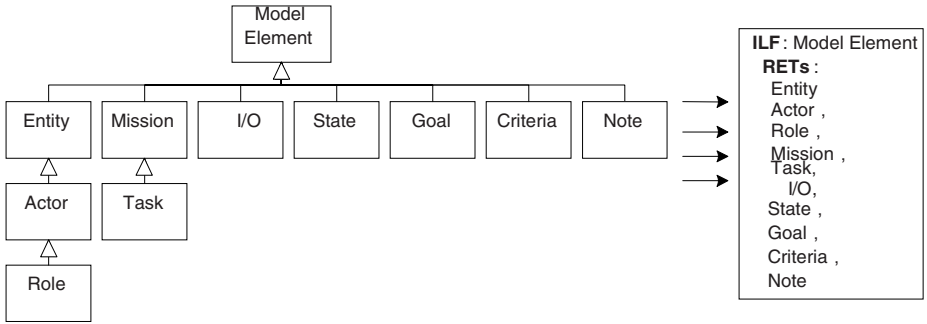


Fig. 2. A mapping from entities to an ILF in IFPUG FPA

Identifying the data functions (ILF & EIF) was useful in determining the transactional functions, because the primary intend of the transactional functions (elementary processes in IFPUG FPA) is to maintain one or more logical files (create, update, read, delete, etc.). Besides, the functional complexity of a transactional function depends on the number of logical files referenced and the total number of input and output DET to and from the transaction.

Unifying all special entities in the inheritance hierarchy into an ILF also combined many of the transactions performed on each of the special entity. For example, a transaction of creating an ‘actor’ model element was combined with creating a ‘state’ model element, even though system may need to behave in a different way for each of them. It can be argued that those two entities are separate in the user domain and whether the application handles both entities in the same way or not can be a design choice rather than a decision to be given in the requirements phase.

The difference for those two cases can be significant for applications similar to KAMA, where entity abstractions (aggregation, generalization, etc) are applied extensively. For example, for the elementary process of creating a model element, the size is 6 FP (complexity level being high). On the other hand, having separate element creation process for each special type would result significantly larger values in total. For 10 specific types, the result would be 60 FP (each having 6 FP with functional complexity level high). Applying the same principle for other generalized entities (relationship and diagram types) and related elementary processes (update, deletion, read, etc.), the difference would be more substantial.

Based on these assumptions, where we consider each special type as a separate ILF, we re-measured the size and the resulting value turned out to be 1789 FP, as opposed to 306 FP in the first measurement performed in the case study. The number of elementary processes increased from 45 to 260 and the number of ILFs increased from 11 to 41. 485% difference in the functional size is significant.

Another notable difficulty about IFPUG FPA is related to the counting rules for transactional functions. One of the rules to be applied in order for an elementary process to be counted as a unique occurrence of an elementary process (external input-EI, external output-EO or external inquiry-EQ) is the following [12]:

“The set of data elements identified is different from the sets identified for other external inputs/outputs/inquiries for the application.”

In the context of entity generalization/specialization, this can be interpreted in a way which is different than the practices applied in the counting manual and other guiding sources [9]. For example, with respect to the practices applied regarding the rules in the counting manual, creating an 'actor' and 'state' model elements is considered as a unique external input maintaining the 'model element' ILF. However, with respect to the rule given above, we can argue that, if the 'actor' and 'state' model elements have different attributes other than the ones they have in common, creating each of them can be considered as different elementary processes. Because, creating an 'actor' model element will maintain a different set of DETs than creating the 'state' model element. This interpretation yet again may result considerable differences in the result. In order to observe the affect of such an interpretation on our case project, we recalculated the functional size. The resulting size value was 512 FP, which is 67% more than the original 306 FP value. The number of ILFs remained the same but the number of elementary processes increased from 45 to 82. Hence, different interpretations and assumptions regarding the counting rules and the structure of the data led to differences in functional size, which was significant for our case.

COSMIC Implementation Results. For the COSMIC FSM measurement case, the measurement group had some difficulties particularly in identifying the functional processes and measuring their functional size. One of the main reasons for that was the lack of clear assistance in the measurement manual [28] for distinguishing processes that maintain a set of 'objects of interests' that can be abstracted to a general entity. The group's tendency was to treat all special entities as separate object of interests, and consider each transaction performed on them as separate functional processes. For example, two functional processes; creating a 'mission space' diagram and creating 'entity ontology' diagram are considered as separate since they maintain two different object of interests and they are triggered by different triggering events perceived by the functional user (triggering event 1 - the user wants to create an 'entity ontology' diagram; triggering event 2 - the user wants to create 'mission space' diagram). Accordingly, we obtained totally 270 functional processes. However, the processing logic of the functional processes which maintain sub-entities is the same. Therefore, we considered those as the same and measured only one. Based on this assumption, we obtained 55 functional processes in total, which was only 20% of the value obtained in first measurement.

Another difficulty was the measurement of each functional process which maintains sub-entities. The measurement group needed to refer to COSMIC FSM guideline for sizing business applications software [27]. The COSMIC FSM measurement manual [28] recommends the reader to refer to this guideline for the details on determining object of interests and separate data groups. To better handle generalization, the guideline introduces a new term; 'sub-type object of interest'.

According to the guideline, sub-types are the specialized entities (classes) that are in the lowest level in the inheritance hierarchy. The general principle is that where there is a need to distinguish more than one sub-type in the same functional process, each sub-type is treated as a separate object of interest. Hence, according to the rules in the guideline, instead of having separate functional processes for each special

entity, their contribution on the functional size was taken into account by including additional data movements for each of the special entity (sub-type object of interest) in the functional processes. However, if the functional process did not need to distinguish special entities, only the general entity is referred. For example, creating a model element is a functional process that requires distinguishing each type of model element. For 10 special entities, there were 10 Entry and 10 Write data movements in the functional process.

COSMIC FSM guideline for sizing business applications software defines specific rules to handle entity generalizations in measuring the functional size of a functional process and provides examples demonstrating how generalizations can be reflected to the measurement practice. We still faced difficulties in identifying the functional processes maintaining sub-entities. Identifying functional processes are derived by the set of triggering events sensed by each of the functional user. In our case study, although their processing logics are similar, we arrived at separate functional processes each maintaining different sub-entities. Therefore, it is necessary to extend COSMIC FSM measurement manual with specific rules in order to clarify the procedure to be followed when identifying and combining similar functional processes which maintain sub-entities that are generalized. In addition, it is still arguable whether generalization or specialization practices can be performed in the user domain or they belong to the solution domain and are design issues.

In our study, we observed that, with different interpretations and assumptions, significantly different set of base functional components (BFCs) for the same software can be identified and this can occur not only among different FSM methods but also for the same method. We observed that there is an improvement opportunity for both methods regarding the rules to better accommodate entity generalizations, since current rules are subject to ambiguity and interpretation.

References

1. Abrahao, S., Poels, P., Pastor, O.: A Functional Size Measurement Method for Object-Oriented Conceptual Schemas: Design and Evaluation Issues. *Software & System Modeling* 5(1), 48–71 (2006)
2. Albrecht, A.J.: Measuring Application Development Productivity. In: *Proc. of the IBM Applications Development Symposium*, Monterey, California, pp. 83–92 (1979)
3. Caldiera, G., Antoniol, G., Fiutem, R., Lokan, C.: Definition and Experimental Evaluation of Function Points for Object Oriented Systems. In: *Proceedings of the 5th International Symposium on Software Metrics*, Bethesda (1998)
4. Cuadrado-Gallego, J.J., Rodriguez, D., Machado, F., Abran, A.: Convertibility Between IFPUG and COSMIC Functional Size Measurements. In: Münch, J., Abrahamsson, P. (eds.) *PROFES 2007*. LNCS, vol. 4589, pp. 273–283. Springer, Heidelberg (2007)
5. DeMarco, T.: *Controlling Software Projects*. Yourdon press, New York (1982)
6. Diab, H., Frappier, M., St. Denis, R.: Formalizing COSMIC-FFP using ROOM. In: *ACS/IEEE Inter. Conf. on Computer Systems and Applications*, pp. 312–318 (2001)

7. Fetcke, T., Abran, A., Nguyen, T.H.: Mapping the OO-Jacobson Approach into Function Point Analysis. In: Proceedings of TOOL 1997, Santa Barbara, CA (1998)
8. Fetcke, T., Abran, A., Dumke, R.: A Generalized Representation for Selected Functional Size Measurement Methods. In: International Workshop on Software Measurement (2001)
9. Garmus, D., Herron, D.: Measuring the Software Process: A Practical Guide to Functional Requirements. Prentice Hall, New Jersey (1996)
10. Gencel, C., Demirors, O.: Conceptual Differences Among Functional Size Measurement Methods. In: Proc. of the First International Symposium on Empirical Software Engineering and Measurement - ESEM 2007, Madrid, Spain, pp. 305–313 (2007)
11. Gencel, C., Demirors, O.: Functional Size Measurement Revisited. ACM Transactions on Software Engineering and Methodology (to be published, 2008)
12. International Function Point Users Group (IFPUG), Function Point Counting Practices Manual, Release 4.2.1 (2005)
13. IEEE Std. 14143.1: Implementation Note for IEEE Adoption of ISO/IEC 14143-1:1998 - Information Technology- Software Measurement- Functional Size Measurement -Part 1: Definition of Concepts (2000)
14. ISO/IEC 14143-1: Information Technology - Software Measurement - Functional Size Measurement - Part 1: Definition of Concepts (1998, revised in 2007)
15. ISO/IEC TR 14143-5: Information Technology - Software Measurement - Functional Size Measurement - Part 5: Determination of Functional Domains for Use with Functional Size Measurement (2004)
16. ISO/IEC IS 20968:2002: Software Engineering - MK II Function Point Analysis - Counting Practices Manual (2002)
17. ISO/IEC IS 20926:2003: Software Engineering - IFPUG 4.1 Unadjusted Functional Size Measurement Method - Counting Practices Manual (2003)
18. ISO/IEC 19761:2003: Software Engineering - COSMIC-FFP: A Functional Size Measurement Method (2003)
19. ISO/IEC IS 24570:2005: Software Engineering - NESMA functional size measurement method Ver.2.1 - Definitions and counting guidelines for the application of FPA (2005)
20. Jenner, M.S.: COSMIC-FFP and UML: Estimation of the Size of a System Specified in UML – Problems of Granularity. In: Proc. the Fourth European Conference on Software Measurement and ICT Control, pp. 173–184 (2001)
21. Jones, T.C.: A Short History of Function Points and Feature Points. Software Productivity Research Inc., USA (1987)
22. NESMA, Definitions and Counting Guidelines for the Application of Function Point Analysis, Version 2.0 (1997)
23. OMG, Unified Modeling Language: Superstructure, Ver.2.0, Formal/05-07-04, Object Management Group (2005)
24. Rains, E.: Function points in an Ada object-oriented design? OOPS Messenger 2(4), 23–25 (1991)
25. Symons, C.: Software Sizing and Estimating: MkII Function Point Analysis. John Wiley, Chichester (1993)
26. Symons, C.: Come Back Function Point Analysis (Modernized) – All is Forgiven!. In: Proc. of the 4th European Conf. on Software Measurement and ICT Control (FESMA-DASMA 2001), Germany, pp. 413–426 (2001)
27. The Common Software Measurement International Consortium (COSMIC): Guideline for Sizing Business Applications Software Using COSMIC-FFP, Version 1.0 (2005)
28. The Common Software Measurement International Consortium (COSMIC): COSMIC Method, Version 3.0, Measurement Manual (2007)

29. The United Kingdom Software Metrics Association: MkII Function Point Analysis Counting Practices Manual, V.1.3.1 (1998)
30. Whitmire, S.A.: 3D Function Points: Scientific and Real-time Extensions to Function Points. In: Proceedings of the 1992 Pacific Northwest Software Quality Conference (1992)
31. Whitmire, S.A.: Applying function points to object-oriented software models. In: Keyes, J. (ed.) Software Engineering Productivity Handbook, pp. 229–244. McGraw-Hill, New York (1992)

Towards a Capability Model for the Software Release Planning Process — Based on a Multiple Industrial Case Study

Markus Lindgren¹, Rikard Land², Christer Norström², and Anders Wall³

¹ ABB Force Measurement, Västerås, Sweden
`markus.lindgren@mdh.se`

² School of Innovation, Design, and Engineering
Mälardalen University, Västerås, Sweden
`rikard.land@mdh.se`, `christer.norstrom@mdh.se`

³ ABB Corporate Research, Västerås, Sweden
`anders.wall@se.abb.com`

Abstract. Software release planning is an important activity for effectively identifying the customer needs generating best business, especially for incremental software development. In this paper we propose a capability model for improving the release planning process of an organization. Using this model it is possible to 1) determine the capabilities of an organization’s release planning process, and 2) identify areas for improvement. The model is based on empirical data from a multiple case study involving 7 industrial companies, all being producers of software intensive systems. We also provide examples of how the proposed capability model can be applied using the companies from the study.

1 Introduction

Release planning can be seen as a company-wide optimization problem involving many stakeholders where the goal is to maximize utilization of the often limited resources of a company and turn them into business benefit [1]. As input to release planning is a set of *needs* that, when implemented as part of a product, provides some business/customer value. Release planning results in a decision of what to include in future *release(s)* of a product, and consequently, a decision of what *not* to include; normally the cost of implementing all proposed needs exceeds the budget allocated to a release. Thus, the *set of needs* needs to be prioritized in order to maximize the business value of the included needs. In addition, there are constraints that must be considered during release planning [1], such as, time-to-market and dependencies between needs. An overview of some relevant aspects of release planning is illustrated in Fig. 1.

Poorly performed release planning can result in “wrong” features being released to the market and/or being released at the “wrong” point in time. Another possible impact of poor release planning is inefficient use of the resources available to an organization. Ultimately, release planning impacts how successful and profitable an organization can become.

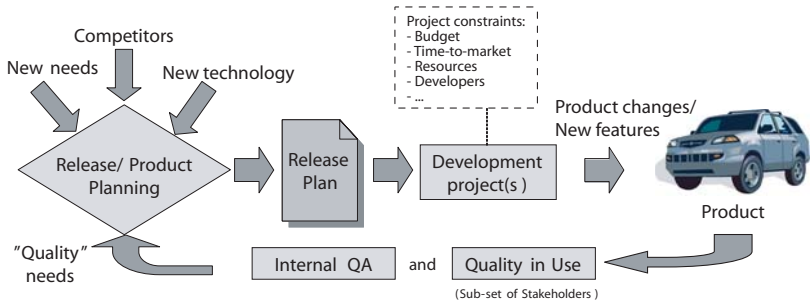


Fig. 1. Overview of relevant aspects of release planning

In this paper we propose a capability model for improving the release planning process of an organization, which is based on empirical data from a multiple case study on software release planning in industry [2,3,4]. Using this capability model it is possible to:

- Determine the capabilities of an organizations release planning process.
- Identify areas for improvement.

We also illustrate how this capability model can be applied to determine the capabilities of a company’s release planning process, using the companies from our multiple case study as examples.

The outline of this paper is as follows: Section 2 presents a selection of related work, Section 3 describes the research method, Section 4 presents the release planning capability model, Section 5 presents examples of using the release planning model, Section 6 provides a more extensive release planning example from industry, putting things into context, and Section 7 discusses future work. Finally, in Section 8 we summarize the paper.

2 Related Work

Release planning research is mainly focused on formalizing the release planning problem, typically by formulating the problem as an optimization problem, where customer value should be optimized while subject to a set of constraints [1,5]. In addition, there are a number of tools being developed implementing these algorithms [6]. However, there is also work indicating that the release planning problem in itself is “wicked” and therefore hard, and possibly unsuitable, to formulate as an optimization problem [7].

In [8,1] two different approaches to release planning research are discussed, referred to as the *art* and *science* of release planning. The approaches formulating release planning as an optimization problem belong to the science approach, while we in this paper are more focused on the art of release planning. We discuss how a company can perform release planning from a more practical point of

view. Furthermore, it is rare to consider how the existing system impacts release planning, exception being [8,6], which again are science approaches.

Requirements engineering is related to release planning, e.g., prioritization of needs is a common problem. However, release planning is a more general problem, since it, e.g., considers resource constraints [6]. Focal Point is one example of a requirements prioritization tool based on the work presented in [9].

Research within the area of process improvement is active, where the perhaps most well-known model and most used in practice is the CMMI [10]. CMMI is focussed on an organization's capabilities and maturity of running product development project(s). It specifies practices that must be adhered to in order to reach a specific CMMI level, where CMMI level 1 represents an organization with lowest maturity and CMMI level 5 represents the highest maturity. However, CMMI provides little detail on how to perform release planning. The CMMI process areas being related to release planning are *project planning*, *requirement development*, and *requirement management*. Within these process areas there are practices for capturing and managing requirements, but when it comes to how to select which features to include in the next release of a product there is no information or practice; CMMI in several cases merely states "resolve conflicts". This paper can be seen as an extension of CMMI that addresses some areas where CMMI provides little or no information.

There are also a number of standards which have parts related to release planning, for example, IEEE Std. 830:1998, 1220:2005, 12207:1996, and 15288:2002. IEEE Std. 830 specifies how a complete, correct, and non-ambiguous software requirement specification should be written. IEEE Std. 1220 specifies the tasks required throughout a system's life cycle to transform stakeholder needs, requirements, and constraints into a system solution. IEEE Std. 12207 specifies a common framework for software life cycle processes, similarly IEEE Std. 15288 defines a framework for systems engineering life cycle processes. As these are standards, they rarely state how to perform a specified required task, instead they state that the task must be performed (somehow). In this paper we provide more "hands-on" approaches to release planning; surprisingly little is concerned with release planning in these standards.

To conclude, existing research provide little information concerning how to improve an organizations release planning process, and there is little work on how to consider the quality of the existing system during release planning.

3 Research Method

In this paper we investigate the effectiveness of performing software release planning. We do this by proposing an initial model expressing the capabilities of an organizations release planning process, which aid in identifying areas for improvement. The model is a first attempt in this direction which we expect to be refined and become more detailed over a period of time. This work is based on empirical data collected during a multiple case study involving 7 industrial companies [4]. For confidentiality reasons there are no company names, no names of

interviewed people, and no absolute numbers on, e.g., budget, in this paper nor in [4], but where possible we present relative figures.

In our study we have used semi-structured interviews as the primary data collection method (with a common line of questions/topics), sometimes complemented by documents received from the interviewees. The main alternative, direct observations, has not been used due to the topic being studied containing company sensitive information and partly due to practical limitations.

In conducting the study we have followed the recommendations by Yin [11] for multiple case studies. We have addressed *construct validity* in multiple ways. First, there have always been two researchers present during each interview in order to reduce possibilities of misunderstandings. Second, interview notes have been taken during each interview, which have been sent to interviewees for approval [4]. Third, we have had two test interviews to improve our interview setup. In total we have interviewed 16 people (excluding test interviews), typically 2–4 persons per company to achieve data triangulation.

To strengthen the *internal validity* of our study we have used multiple researchers when performing analysis and made use of pattern-matching techniques, and we have considered rival explanations. To increase *reliability* of our study, all collected data, and derivations thereof, are stored in a database accessible only to the researchers in the study, e.g., interview notes and merged notes per company. In addition, the study design is documented, which includes the interview questions. Using this material it is possible to trace conclusions to collected data, and vice versa. To counter *researcher bias* multiple researchers have been involved in most of the steps of this study. Furthermore, companies that the researchers in the study are affiliated with are excluded from the study.

Thanks to industrial contacts we have been able to find a relatively large number of companies and persons willing to participate in our study, which aids in increasing the *external validity* of our results. However, there is risk that the selection is not fully generalizable to other domains and/or nationalities/cultures. In selecting people to interview we have asked our contact person(s) at each company for references to people working with release planning. Our interviews have mainly been with product managers, managers, and project leaders.

All companies in the study develop software intensive embedded systems with a typical life cycle of 10–20 years. However, the companies are in different product domains, e.g., automation, telecommunication, and automotive. Table 1 presents some relative data concerning the characteristics of products developed, produced, and sold by the studied companies in order to provide a feel for their main characteristics. In Table 1 *Volume* refers to the produced product volume, while the rows *% Software*, *% Electronics*, and *% Mechanical* is our subjective judgment of the products' software, electrical, and mechanical content, which in turn reflect the amount of resources these companies invest in these areas. Case 3 is excluded from the table since it is a management consulting company that has no products. The case numbering in Table 1 is consistent with [4,2].

We are partly using a grounded theory approach [12] in this research, since we define a model based on observed data. The model we present in this paper

Table 1. Characteristics of companies in the study, where VH = Very high, H = High, M = Medium, L = Low, and VL = Very low

Case	1	2	4	5	6	7
Volume	VH	H	VH	L	M	L-M
% Software	L-M	L-M	H	L-M	M	H
% Electronics	M	M	H	M	M	M
% Mechanical	H	H	L	H	M	L
Employees	H	H	VH	L-M	VL	VL-L
% in R&D	VL	VL	H	L-M	VL	VL

has partially been validated in a workshop with participants from our study. However, since we created the model based on collected data, the same data cannot be used to validate the model. Nevertheless, it can be used to motivate and illustrate the model until further studies validate it.

Proper validation of the model requires a baseline with which to compare, for example, the state before changing the release planning process in an organization, and then collect data after introducing the change; as has been done for CMMI [13]. We have not yet reached such a state in this research.

4 Improving the Release Planning Process

In its generalized form, release planning can be considered to consist of three different process activities, as illustrated in Fig. 2:

Elicit Needs. Collect stated, and unstated, needs from the different stakeholders. Other literature, e.g. [8], refers to this activity as requirements elicitation, which typically refers to a phase within a development project. We mainly refer to need collection occurring prior to forming the development project.

Make Release Decision. Prioritize the needs such that the cost and schedule for realization fits within the constraints of the release, and decide the contents of the release. Again, this is an activity that primarily occurs prior to the development project.

Realize Needs. Typically performed as product development project(s) within the research and development (R&D) part of the organization, where the prioritized needs are implemented as part of a product(s).

These process activities can, but need not, be performed in sequence; typically these are continuous activities with data flow between them.



Fig. 2. Overview of the release planning process

In this section we present a capability model for improving an organization's release planning process, inspired by the Test Process Improvement (TPI) [14] framework, and partly by the CMMI [10], therefore there are some similarities. Our focus is placed on need elicitation and making the release planning decision, while we are aware of there being other important areas within release planning as well, such as, choice of time horizon and resolving need dependencies.

Each of the following sub-sections describe *key-areas* within the three process activities from Fig. 2. For each key-area a capability scale is presented, with levels from A–D, where A represents lowest capability and D represents highest capability. Using these descriptions it is possible to pinpoint the capability for an organization within each key-area. While describing the key-areas we also present some examples of how the key-area can be applied in practice, using examples from our multiple case study [4]. However, it should be noted that we do not suggest that it is always economical for an individual organization to strive for level D in each key-area.

We have derived the set of key-areas based on what we have observed as being important activities in our study; the set we present is in no way guaranteed to be complete. Within each key-area we have ranked observed data from the cases within each area to form the capability levels. The ranking has partly been validated in a workshop with participants from our study. However, we expect our proposed capability model to be refined and detailed in the future; this is only a first attempt at building a capability model for release planning. For example, in this model there are capability levels from A to D, however, it is not necessary for there being four levels within each key-area. Furthermore, the model has been devised with a focus on software, although it may be possible to apply the model in other domains as well.

In TPI [14] there are also key-areas with levels A–D. In addition, there is a *test maturity matrix* relating the level within each key-area to a test maturity scale, from 0 to 13. For example, for an organization to have rating 3 it is required to have level A in the key-area *Estimation and planning*, level B in *Test specification techniques*, etc. In our work we currently haven't reached a state where we can present a similar maturity matrix for release planning. Yet, the key-areas will allow an organization to identify possible improvement key-areas.

4.1 Elicit Needs

We have identified the following key-areas within the process activity elicit needs: *need Elicitation* and *need documentation*.

Need elicitation: Refers to how and from which stakeholders needs are elicited. Elicited needs are prioritized, in other key-areas, and a decision is made of what to include in a release. The capability scale is as follows:

Level A. Adhoc. Needs are collected when opportunities arise, for example, during meetings with customers or other stakeholders. Typically this activity is unstructured and performed by product management.

Level B. Formal Path. Each (important) stakeholder group has a formal path for passing need requests to product management. Typically these needs are collected prior to upcoming release planning decisions. This formal path should be described in the process description for the organization.

A stakeholder group refers to a specific “type” of stakeholders, e.g., the end-customer can be one such type. Other such possible types are developers, testers, and commissioners. What differentiates these are that they each use the product in different ways, and therefore also have different needs.

Level C. Stakeholders Prioritize Needs. Needs are collected using both A and B, but with the addition of each stakeholder group also assigning priorities to the needs. Product management, which makes the release decision, receives a set of prioritized need lists, one from each group, and is required to make the release plan decision and to be able to motivate this decision.

Level D. Stakeholders Rate Needs Based on Product Strategy. An extension of C where the internal stakeholders of the company assign priorities based on the product and/or company strategy; external stakeholders prioritize needs according to level C.

Example 1. Case 1 fulfils level B by having three parts within the organization, which each focus on a separate area of the product [4]. These areas are *product features*, *product quality*, and *cost-cut* (mainly related to production cost), which each propose needs to product management; illustrated in left part of Fig. 3. In a similar way Case 4 has a formal path for collection of feature needs and quality needs; illustrated in right part of Fig. 3.

Example 2. Case 4 develops a product platform used by two other parts of their organization, O_1 and O_2 [4]. One way in which Case 4 fulfils level B for need elicitation is that product management for O_1 and O_2 propose needs to product management for Case 4. In addition, system responsables from O_1 , O_2 , and from Case 4 propose needs to product management for Case 4, as is illustrated in Fig. 4. Furthermore, they apply a principle, called “one-voice”, where each group (i.e., each line to product management in Fig. 4) prioritize the set of needs before passing them to product management for Case 4, thereby fulfilling level C.

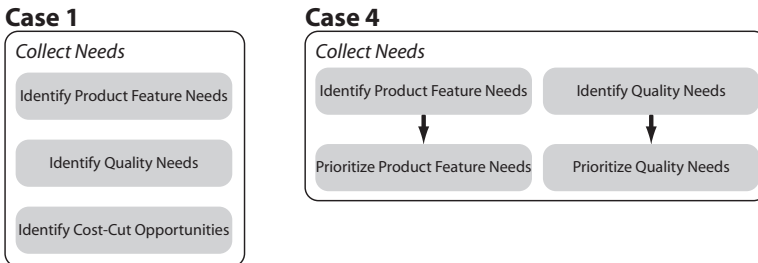


Fig. 3. Two examples of a formal path existing for need elicitation

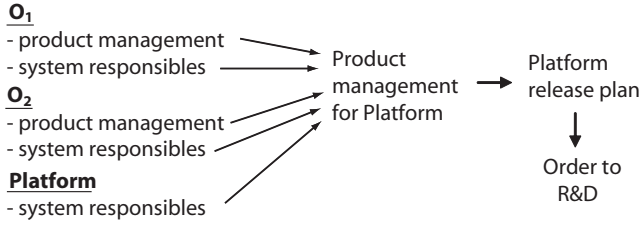


Fig. 4. Example of having a formal path for need elicitation

Documentation: For a need to become eligible for prioritization into a release there typically needs to be some documentation, e.g., a short description of what business benefit the need aims to fulfil. The capability scale is as follows:

Level A. Adhoc. Only a short description of the need, if any.

Level B. Template. A common template for need documentation consisting of at least: a short description of the need, an initial cost estimate for realizing the need, an initial return-of-investment calculus, and a statement concerning the consequences on the existing system of introducing the need. Typically, this documentation should be at most one page.

Level C. Tool Support. The information from Level B are stored in a tool/database, which can be accessed by all people involved in need elicitation.

Level D. Type of Need. An extension of Level C where the needs are classified according to at least the following types: new feature, quality improvement, cost-cut.

Example 3. In our study [4] all the companies use some form of template for the proposed needs. However, its form range from a short statement, a one page statement (as in level B), to templates with 3 PowerPoint slides.

4.2 Make Release Decision

These are the key-areas we have identified belonging to making the release decision: *decision Material*, *product strategy*, and *release plan decision*.

Decision material: In addition to the need documentation described in Section 4.1 there can be different types of studies that refine the needs and produce decision material, which complement the mentioned need documentation. The purpose of the decision material, produced via studies, is both for increasing confidence of data and risk reduction. Typically this is related to, e.g., refining cost-estimates, determine consequences for the existing system, refining return-of-investment calculus. The capability scale is as follows:

Level A. Adhoc. No formal decision material is used, instead the decision material is formed by the “gut-feeling” [2] of the individuals involved in making the release plan decision.

Level B. Unstructured Pre-study. A pre-study is performed with purpose of refining a need proposal. The results produced by the pre-study partly depend on which individual(s) perform the pre-study, and partly on the people ordering the pre-study.

Level C. Structured Pre-study. A structured pre-study [2], compared to an unstructured (level B), has a standardized set of issues which should be investigated in the pre-study. It considers alternatives, as is described in the process area Technical Solution in CMMI [10].

Level D. Feasibility Study. This is more oriented towards the solution for how to realize the proposed need(s) into the existing system, and investigating different alternatives. In addition to performing a structured pre-study, as in level C, a feasibility study is performed with the goal determining how the proposed need(s) can be realized into the existing system, the resources required to complete the task, refine cost-estimates, and address market issues; see feasibility study on Wikipedia.

Example 4. The development projects in Case 1 are to large degrees concerned with production issues, since their products have large mechanical content (see Table 1) and the production is both complicated and costly. Therefore, before any decisions are made they need to know the consequences for production (as far as possible). These consequences are investigated via structured pre-studies, illustrated in left part of Fig. 5.

Case 4 on the other hand delivers a product platform to O_1 and O_2 (previously mentioned in Example 2), which in turn use the platform to develop products. In case the platform is delivered at the wrong point in time or with too poor quality, this will have consequences on the efficiency for O_1 's and O_2 's development projects. To reduce this risk, and to improve confidence in the decisions being made are the correct ones, Case 4 use both pre-studies and feasibility studies; before and after each study it is possible to re-prioritize the proposed needs, illustrated in Fig. 5.

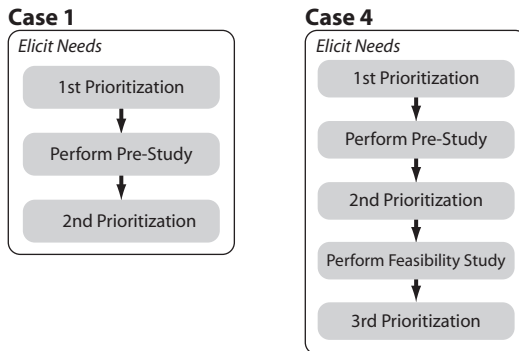


Fig. 5. Two examples of using studies to refine need proposals

Product strategy: Having a well-defined and clearly communicated product strategy often helps employees within an organization to know what to strive for; also an aid in daily prioritization of tasks. So far we have not yet reached a state where we can present a capability scale. Still, we present some examples from our industrial cases [4]:

Adhoc. Each release basically has its own focus and is not directly related to any product/business strategy.

Release Profile. Case 7 uses a *release profile*, which sets the top-level priorities for the next release. For example, the profile can be aimed at improving usability and/or extending the set of supported communication protocols.

Product Strategy. Each product produced by Case 1 has an attribute profile, consisting of more than 20 attributes, defining the target properties of the product. Furthermore, the attribute profile is clearly linked to the company strategy. Case 2 also has a clearly defined product strategy, with 8 core values and 3 premium values; these values are used when determining the business value of proposed development projects.

Another possibility expressing the product strategy is by using the *measure of effectiveness* (MOE) defined in IEEE Std. 1220:2005, which is an explicit mathematical expression "... by which an acquirer will measure satisfaction with products produced by the technical effort." The expression should capture the top-level goals, i.e., should not contain too many details. The difference compared to the previous strategy, is that it is measurable.

Release plan decision: The needs elicited in Section 4.1, including its documentation and decision material, is used as a basis for prioritization and followed by a decision of what needs to be included in the next release. We have not yet reached a state where we can present a full capability scale, the first two levels are presented, while the following two are proposals which might fit into the capability scale (these have not been observed in the industrial cases):

Level A. Adhoc. Needs are prioritized based on the "gut-feeling" [2] of product management.

Level B. Tool Support. A decision support tool, such as Focal Point, is used to aid in making the decision. The prioritization should make use of the product/company strategy.

Metrics (not based on observed data). Use metrics, e.g., production cost, maintenance costs, used product options, sold product volume, quality-in-use, and prediction of these variables, as support when making decisions. We have not yet defined a specific set of metrics, but to reduce the number of decisions based on "gut-feeling" [2] more objective data must be used.

Optimizing (not based on observed data). Once proper metrics are in place it might be possible to introduce optimization. For example, by adapting current "science" approaches [1,6] to consider these metrics when computing release plans. Further research is required to reach such a position.

4.3 Realize Needs

How to realize needs in development projects is not within the scope of this paper; refer to, e.g., CMMI [10] for a description of practices/key-areas. However, one related issue is given an organization with certain capabilities of its release planning process and a certain maturity for performing development projects, e.g., determined using CMMI, which of these two areas should be improved in order to have best effect?

The maturity and capabilities of performing development projects controls the *efficiency* with which needs can be implemented, while release planning is more focused on *effectiveness*, i.e., making sure that the correct features and quality improvements are released to customers. Consequently, it is not certain that a company with 100% efficiency in its development projects is the most successful one. This indicates there being a need for being at least as good at release planning as performing development projects.

5 Application of the Capability Model

In this section we evaluate the release planning key-areas presented in Section 4 for the companies in our multiple case study [4]. This evaluation is based on qualitative reasoning of the empirical data from our study, which is the same data used in developing the release planning capability model. Based on the results from the evaluation we also present suggestions for how to improve the release planning processes for the companies. We lack data for making explicit conclusions in some cases due to using a grounded theory approach in building our capability model, i.e., the model was constructed after collecting the data.

The results of our analysis is summarized in Table 5; case numbering is consistent with [4,2]. Below we comment on each key-area requiring further analysis to reach a conclusion concerning the capability level.

Need Elicitation: Case 1 fulfils level B since it has a formal path for product features, quality improvements, and cost-cut; as discussed in Example 1. Case 4 fulfils level C by having a formal path, for product features and quality needs, and by the stakeholders prioritizing the needs; as discussed in Example 2.

Table 2. Capability levels for each release planning key-area and company

Key-Area	Need Elicitation	Need Documentation	Decision Material	Product Strategy	Release Plan Decision
Case 1	Level B	Level A	Level C	Product strategy	Level A
Case 2	Level B-D	Level A?	Level C	Product strategy	Level A?
Case 4	Level C	Level B	Level D	Adhoc	Level A/B
Case 5	Level A	Level B	Level A-B?	Adhoc	Level A
Case 6	Level A	Level B	Level B	Adhoc	Level A
Case 7	Level A	Level A-B	Level B	Release profile	Level B

We have insufficient data for clearly determining the capability level for Case 2, but it is somewhere between level B-D on the capability scale. Development projects are rated using the product strategy, indicating part support for level D, but we lack data concerning the existence of a proper formal path (level B) for need elicitation and if stakeholders prioritize needs (level C).

For Case 5 our interviews cover release planning with a planning horizon of 5–10 years, and therefore we lack data for the more near time planning. Still, the data we have indicate level A.

Case 6 and Case 7 are rather similar, both having level A. This judgement is made since they do not have any formal path for needs from R&D. Though, there are formal paths from sales and marketing.

Need Documentation: Almost all companies in the study have some form of template for documenting proposed needs. For example, Case 5 and Case 6 document needs using up to three PowerPoint slides, with cost estimates, business impact, and a time plan. Case 4 uses a “one-pager” containing a slogan, business benefit, estimated cost, and impact on other parts of the system. Case 7 has an Excel template but seems to lack cost-estimate, but there may be other templates not covered during the interviews. Case 1 seems to use only a short-description, but which is refined in pre-studies and later stored as a change request in a database. For Case 2 we have no exact data.

Decision Material: Case 1 fulfils level C, as discussed in Example 4, and Case 4 fulfils level D, as also discussed in Example 4. Case 2 performs structured pre-studies and projects are also required to rate their impact on the 8 core values defined in their product strategy. The data we have from Case 6 and Case 7 indicate that they perform pre-studies, but the format for these pre-studies is not strictly defined (level B). We lack data for Case 5, but based on our impression from our interviews we suspect they are between level A–B.

Product Strategy: Case 1 and Case 2 have clearly defined product strategies as discussed in Section 4.2. Case 7 uses a release profile defining the top-level goals for the next release. Case 4, Case 5, and Case 6 did not seem to have a defined product strategy that had impact of how needs were prioritized, instead these companies based their decisions making “good business”.

Release Plan Decision: The release plan decision is usually made in a group discussion, where typically stakeholders need to “lobby” for their own case. In case there is data support the proposed need, e.g., a customer survey, then such data often has strong impact beneficial. Case 1, Case 4, Case 5 seem to handle in the decision making in similar ways. Case 7 uses the tool Focal Point [9] as an aid in decision making. Case 6 has tried using Focal Point, but considers to be a bit awkward when comparing needs with very different costs.

5.1 Improvement Proposals

Here we discuss some possible ways in which the companies in our study can improve their release planning processes.

Case 4 has highest capability level, among the companies in the study, for need elicitation, decision material, and need documentation; see Table 5. We have data from other sources indicating that Case 4 also has highest CMMI level among the investigated companies. We have not looked further into this issue. Areas where Case 4 possibly can improve is by defining a more clear product strategy and by employing decision support tools to a greater extent.

Case 7 can improve their need elicitation by having a formal path for R&D and possibly other important stakeholders. Probably they will benefit of using R&D for performing structured pre-studies, which in turn should result in better time and cost-estimates for development.

Case 6 is in a similar position as Case 7; improvement areas being need elicitation and decision material. Possibly they can improve by using a release profile. Case 1 has a very large product volume compared to most of the other cases (see Table 1) and make use of many metrics and customer surveys to track their performance. Possibly they can improve their need documentation.

For Case 2 and Case 5 we lack data for making good improvement suggestions. Still, it should be noted that Case 2, compared to the other companies, seems to take more decisions on lower organizational levels.

6 The Problem into Context

The different process activities and key-areas described in Section 4 can be implemented in many different ways. For example, it is possible to use them in a staged/waterfall manner or in an iterative way. What suits an organization usually depends on the business context, and therefore it is not necessarily related to release planning capabilities. However, to provide a better understanding for how the key-areas can be combined we provide one example from our study.

Example 5. One basic understanding concerning release planning for Case 4 [4] is that *there will be changes* during a release project, e.g., there will always be needs which aren't thought of during the initial planning of a release and there will be changes to proposed needs. To cope with this they do not assign more than 50% of the release budget to the initial release plan, the remaining 50% is planned to be used for needs and changes during the release project.

During the initial phases of release planning there is usually 4 times as many needs as can fit within the budget of a release. In order to identify the needs that provide best return-of-investment/customer benefit they iteratively refine needs using pre-studies and feasibility studies. These investigations explore more needs than can fit within the budget allocated to a release. The needs for which pre-studies are performed requires, if developed, roughly 130% of the release budget. Needs are prioritized after the pre-studies such that feasibility studies are performed for needs requiring roughly 110% of the release budget. Prioritization is also performed after the feasibility studies, which in the end result in a release plan requiring 100% of the release budget, as is illustrated in Fig. 6. Furthermore, the people capable of performing pre-/feasibility studies are often

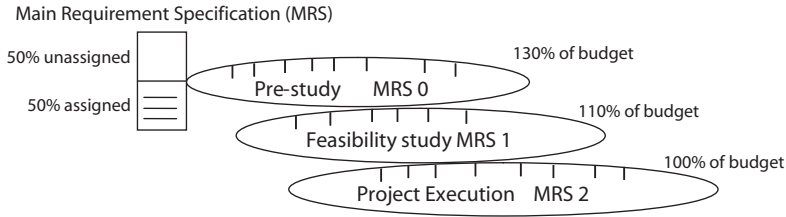


Fig. 6. Overview of iterative need refinement

a scarce resource, therefore these studies are performed throughout the release project; the vertical lines in Fig. 6 indicate start of study.

The of goals of Case 4’s release planning process are:

- Efficiency.** Obtain efficient use of the resources available to the organization, including, e.g., R&D, local sales offices, and marketing.
- ROI.** The goal of basically all companies is to generate profit, therefore investments should be made such that return of investment is maximized.
- Flexibility.** For most organizations it is desirable to have a flexibility that allows late detected needs, e.g., new customer needs, to be included into the current release, thereby enabling a short time-to-market.
- Risk reduction.** In all product development, and within most organizations, there are different kinds of risk. For product development one such potential risk is underestimating cost and/or time for development.

7 Future Work

Here we discuss some possibilities for future work resulting from this paper. There are possibly other “key-areas” for release planning processes, which remain to be identified. Work needs to be performed to develop a maturity matrix for release planning, as exists for TPI [14]. Improve the release planning process capability model, such that it relies less on subjective judgements, and rather use more objective data, e.g., measured directly on the quality and costs associated with a product. Further validation and refinement of the model is also required.

8 Conclusion

Software release planning is an important activity for selecting the needs that best fulfil customer needs and at the same time provide a sound return-of-investment to the organization developing the software. Ultimately, release planning impacts how successful an organization can become.

In this paper we present a capability model for improving the release planning process of an organization. Using this model it is possible to:

1. Determine the capabilities of an organization's release planning process.
2. Identify areas for improvement.

The model is based on empirical data obtained from a multiple case study on release planning in industry involving 7 different companies. All companies in the study are producers of software intensive embedded systems, where the products have a relatively long life cycle; typically in the range of 10–20 years. In the paper we also apply our proposed capability model on the companies in our study, and illustrate how the capabilities of their release planning processes can be determined and discuss opportunities for improvement.

Acknowledgements

This work was partially supported by the Knowledge Foundation (KKS) via the graduate school Intelligent Systems for Robotics, Automation, and Process Control (RAP), and partially supported by the Swedish Foundation for Strategic Research (SSF) via the strategic research centre PROGRESS.

References

1. Ruhe, G., Saliu, O.: Art and Science of Software Release Planning. *IEEE Software* 22(6), 47–53 (2005)
2. Lindgren, M., Norström, C., Wall, A., Land, R.: Importance of Software Architecture during Release Planning. In: *Proc. Working IEEE/IFIP Conference on Software Architecture (WICSA) 2008*. IEEE Computer Society, Los Alamitos (2008)
3. Lindgren, M., Land, R., Norström, C., Wall, A.: Key Aspects of Software Release Planning in Industry. In: *Proc. 19th Australian Software Engineering Conference*, IEEE Computer Society, Los Alamitos (2008)
4. Lindgren, M.: Release Planning in Industry: Interview Data. Technical Report MDH-MRTC-219/2007-1-SE, Mälardalen Real-Time Research Centre (2007)
5. Jung, H.W.: Optimizing Value and Cost in Requirements Analysis. *IEEE Software* 15(4), 74–78 (1998)
6. Saliu, M.O., Ruhe, G.: Supporting Software Release Planning Decisions for Evolving Systems. In: *29th Annual IEEE/NASA Software Engineering Workshop*, pp. 14–26. IEEE Computer Society, Los Alamitos (2005)
7. Carlshamre, P.: Release Planning in Market-Driven Software Product Development: Provoking an Understanding. *Requirements Engineering* 7(3) (2004)
8. Saliu, O., Ruhe, G.: Software release planning for evolving systems. *Innovations in Systems and Software Engineering* 1(2) (2005)
9. Karlsson, J., Ryan, K.: A Cost-Value Approach for Prioritizing Requirements. *IEEE Software* 14(5) (1997)
10. CMMI Product Team: CMMI for Development, Version 1.2. Technical Report CMU/SEI-2006-TR-008, Carnegie Mellon — Software Engineering Institute (2006)
11. Yin, R.K.: *Case Study Research: Design and Methods (Applied Social Research Methods)*, 3rd edn. Sage Publications Inc., Thousand Oaks (2003)
12. Strauss, M., Corbin, J.M.: *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*, 2nd edn. Sage Publications, Thousand Oaks (1998)

13. Gibson, D.L., Goldenson, D.R., Kost, K.: Performance Results of CMMI-Based Process Improvement. Technical Report CMU/SEI-2006-TR-004, Carnegie Mellon — Software Engineering Institute (2006)
14. Andersin, J.: TPI — a model for Test Process Improvement. Technical report, University of Helsinki, Department of Computer Science (2004)

From CMMI to SPICE – Experiences on How to Survive a SPICE Assessment Having Already Implemented CMMI

Fabio Bella*, Klaus Hörmann*, and Bhaskar Vanamali*

KUGLER MAAG CIE GmbH, Leibnizstr. 11
70806, Kornwestheim, Germany

{Fabio.Bella, Klaus.Hoermann, Bhaskar.Vanamali}@kuglermaag.com
www.kuglermaag.com

Abstract. Dealing with multiple models for process assessment and improvement is a challenging, time-consuming task. In the automotive sector, for example, several suppliers drive their process improvement on the basis of CMMI®. However, many car manufacturers require process capability ratings determined on the basis of Automotive SPICE™. The approach presented aims at preparing organizations already working according to CMMI for Automotive SPICE assessments. The approach was already successfully applied in industrial settings and lessons learned are discussed. The approach helps to avoid misunderstandings during assessments due to different model taxonomy, achieve appropriate process ratings, and save both effort and costs.

Keywords: Systems and Software Process Improvement, SPI Methods and Tools, Industrial Experiences and Case Studies, Process Assessment, Lessons Learned, Automotive and Transportation Systems.

1 Introduction

Whenever different customer groups ask for compliance with different models, a need for dealing with multiple models typically arises. An example of this situation is the Automobile Industry where some customers ask for ISO/IEC 15504 (SPICE) compliance, some for Automotive SPICE™ compliance, whereas many companies already started using the Capability Maturity Model® Integration (CMMI®) to drive their internal process improvement.

In the automotive industry, particularly in Europe, SPICE has become a mandatory standard and ticket-to-trade. Car manufacturers (OEMs) such as Audi, BMW, Daimler, Porsche and Volkswagen (the so called HIS group) but also Ford, Volvo, and FIAT are assessing their electronic/software suppliers based on Automotive SPICE. Additionally, the German Association of the Automotive Industry (VDA) has adopted Automotive SPICE as its reference standard for assessing the capability of companies supplying software-determined systems for the automotive industry.

* All authors are iNTACS ISO 15504 assessors, one is also an iNTACS ISO 15504 trainer, SEI SCAMPI Lead Appraiser, and CMMI Instructor.

Some big tier-one suppliers estimate their annual additional costs due to misalignment between the internally available CMMI-compliant processes and the expected Automotive SPICE-compliant processes to be greater than 1 Million Euro per year.

One important question arises: Do assessed organizations need to have knowledge about ISO/IEC 15504 / Automotive SPICE to successfully pass an assessment based on these standards? Yes. There are too many differences between these models that one could expect to pass an Assessment according to model A successfully if the previous process work has been concentrating on model B. In addition, it may become very difficult if not impossible to adhere to the tight assessment time schedules: misunderstandings may occur and, in the worst case, this may lead to an inappropriate capability determination. Our experience shows that applying Automotive SPICE in a CMMI environment has to be prepared. Process improvement and project staff needs to be trained on the basics of Automotive SPICE and relevant gaps need to be identified to allow process documentation to be revised where necessary.

In this paper, we present an approach that enables companies performing internal process improvement on the basis of CMMI to survive SPICE assessments through a systematic preparation process. In particular, this approach aims at avoiding redundant or even conflicting process-related activities. In the first case, process-related activities are performed twice: once for CMMI and again for SPICE. In the second case, process changes introduced to achieve better SPICE results can lead to process deterioration with respect to CMMI rating. The core of the approach relies on mappings between CMMI and Automotive SPICE.

Similar mappings have been provided for older versions of CMMI and Automotive SPICE, e.g. [10]. [3] provides a mapping between an older version of CMMI and the ISO/IEC 15504-2:1998. Numerous proprietary mappings have been performed but are not publicly available.

The AK13, the working group within the German VDA in charge of process assessments, is currently developing a detailed mapping between CMMI and Automotive SPICE and plans to provide a delta-list to support the evaluation of CMMI appraisals in comparison with Automotive SPICE. The list would also support companies implementing Automotive SPICE. We are supporting the VDA in performing this task.

The remainder of this paper is structured as follows. Section 2 presents an excursus on methodologies for process assessment and improvement. Section 3 presents the approach. Section 4 subsumes the differences between CMMI and Automotive SPICE. Section 5 introduces lessons learned when developing and applying it. Finally, section 6 summarizes the paper.

2 Excursus: CMMI, ISO/IEC 15504, Automotive SPICE

In an assessment/appraisal, the processes applied by an organization are examined and compared with the good practices included in a reference model by interviewing project staff and investigating documents from projects or other organizational functions. The objective is to determine to which degree these practices have been implemented within the organization which is commonly referred to as a capability level or maturity level.

The Capability Maturity Model (CMM) [4] was developed by the Software Engineering Institute (SEI). CMM is the first model defined to identify process-related risks by means of appraisals. The model is no longer maintained and has been replaced by the Capability Maturity Model Integration® (CMMI®). CMMI offers different models (called “constellations”), each of which addresses a different area of interest. Models for development [12] and acquisition [11] processes are available. A model for services is currently under development. A large number of companies apply CMMI for their improvement programs. Although the model is proprietary, it is a de-facto standard.

The international standard ISO/IEC 15504 addresses process assessments and the determination of process capability. The publishing process of the first five parts was concluded 2006. ISO/IEC 15504 Part 2 [7] defines the minimum requirements to perform an ISO/IEC 15504 compliant assessment. The standard allows the specification of a specific Process Assessment Model (PAM) based on a suitable Process Reference Model (PRM). The PRM contains high level definitions of processes in terms of process purpose and expected outcomes. Compliant PRMs are ISO/IEC 12207 [5] for software life cycle process, ISO/IEC 15288 [6] for Systems engineering life cycle process, and the Automotive SPICE PRM [2]. Conformant PAMs are the ISO/IEC 15504 Part 5 [8] and the Automotive SPICE PAM [1].

In 2001, an Automotive SPICE Initiative was founded to define a PAM for the automotive sector. The core team of this initiative are representatives of Audi AG, BMW AG, Fiat Auto S.p.A., Daimler AG, Dr. Ing. h.c. F. Porsche AG, Procurement Forum, Volkswagen AG and Volvo Car Corporation (representing Ford Europe, Jaguar and Land Rover).

The members of HIS (“Hersteller Initiative Software” - Car manufacturer Initiative Software), i.e., Audi, BMW, Daimler, Porsche, and Volkswagen together with other OEMs such as Ford, Volvo and FIAT are assessing their software suppliers on the basis of Automotive SPICE. Based on the findings of Automotive SPICE Assessments the OEMs will rate their suppliers in A, B, or C category, thus defining the contractual implications for future work. A C-supplier will not be considered for future quotations while a B-supplier will have to launch an improvement program.

Therefore, it is a ticket-to-trade to reach the required process capability. By the HIS members only, more than 250 ISO/IEC 15504 Assessments have been performed up to now. The total number is expected to increase steadily.

In the reminder of this paper, the term CMMI means CMMI for Development version 1.2 and the term Automotive SPICE means the PAM version 2.3.

3 An Approach for Surviving Automotive SPICE Assessment Having Already Implemented CMMI

The approach described in this section consists of the elements training, mappings process of the organization/CMMI/Automotive SPICE, a gap analysis, and a workshop for planning the necessary steps to prepare the Automotive SPICE assessment. All phases are conducted by an experienced Automotive SPICE assessor also skilled in CMMI.

Automotive SPICE Training. A basic Automotive SPICE training is provided to key roles. The training covers the principles and structure of Automotive SPICE, the capability dimension and the requirements regarding the most important processes (i.e., the processes in scope, typically the HIS Scope). The generic SPICE assessment approach and possibly some particularities of the assessment approach of the OEM requesting the assessment are introduced. For inexperienced organizations, a typical duration is two days for the improvement team and one day for the development team. For experienced organizations, shorter training may be enough.

Mapping 1: CMMI/Automotive SPICE. Even a mature organization working for many years according to CMMI can have substantial gaps with respect to Automotive SPICE¹. A list of the corresponding differences needs to be determined through a mapping. The mapping can be applied further on in future assessment preparations. Due to its importance, the mapping is discussed in more detail in this section.

Mapping 2: Processes of the Organization/CMMI/Automotive SPICE. A generic mapping between the models alone is not sufficient. There is a huge number of detailed differences between the models which may or may not be relevant, depending on how the organization has shaped its processes. It is therefore very important to have mappings available between model requirements and their implementation in terms of the processes and work products of the organization. In other words, if one wants to know if and where a particular model practice has been implemented, mapping 2 will identify the precise process elements and work products. At least one mapping for Automotive SPICE is required, a second mapping for CMMI is useful to evaluate if changes to processes might corrupt the CMMI compliance.

Gap Analysis. Even with mapping 2 some uncertainties remain: It is not clear to which degree the project(s) or organizational functions follow their own processes. This may be due to tailoring options chosen or simply due to low process adherence. Another reason is that often gaps cannot be judged purely on the basis of a process mapping but need the evaluation of practices actually being performed and of actual work products. This is why in all cases a concrete gap analysis has to be performed on the projects and organizational functions in scope. This gap analysis can be guided by the previous mappings. The assessor conducts meetings with the key roles, performing interviews and inspecting documents to determine the actual gaps and documents the results. The gap analysis can be usually conducted within two or three days.

Improvement Workshop. During the improvement workshop, the results of the gap analysis are discussed between the assessor and the key roles. The purpose of the workshop is to prioritize the actual gaps and determine a strategy for closing them before the assessment.

In the following, more details concerning the mapping are presented. CMMI and Automotive SPICE can be compared from at least three different perspectives: with respect to their structure, to their content, and the methods applied to analyze processes. In the following, the respective structures are compared and essential differences regarding the content are sketched.

A first important difference between CMMI and Automotive SPICE is that CMMI uses two representations, the “Staged Representation” and the “Continuous Representation”. However, only the Continuous Representation has the required structure that allows a comparison with Automotive SPICE.

¹ Which holds also true vice versa.

Table 1. Mapping main CMMI and Automotive SPICE concepts

CMMI	Automotive SPICE
Process Area	Process
Purpose	Process Purpose
Specific Goals	Process Outcomes
Specific Practices	Base Practices
Subpractices	-
Typical Work Products	Output Work Products
-	Work Product Characteristics
Generic Goals	Process Attributes
Generic Practices	Generic Practices
Generic Practice Elaborations	-
-	Generic Resources
Examples	-
Amplifications	-
Capability Levels	Capability Levels

Table 1 shows a mapping of the main concepts applied in the two models. For most of the concepts a proper translation can be found. For CMMI's Subpractices, Examples, and Amplifications exist no corresponding concepts in Automotive SPICE. CMMI, on the other side, does not include Generic Resources. Furthermore, only a weak correspondence exists between Specific Goals and Process Outcomes.

With respect to the granularity of the models, Automotive SPICE is subdivided into more processes (31 processes or 48, if additional processes from ISO/IEC 15504 Part 5 are considered) than CMMI (22 Process Areas).

With respect to the content, Fig. 1 depicts the relationships between CMMI process areas and the Automotive SPICE processes included in the HIS scope. As shown in the figure, CMMI covers most of the processes from the HIS scope. The process SUP.9 Problem Solution Management is not addressed by CMMI.

In the following, a list of differences between CMMI and Automotive SPICE is introduced.

RD, REQM Compared with ENG.2, ENG.4

- Communication mechanisms for disseminating requirements are not required in CMMI.
- Traceability requirements are much more explicit in Automotive SPICE
- In general, Automotive SPICE requires three different levels of requirements: customer, system, and software level. CMMI requires only two levels: customer and product level. The product level in CMMI can include as many sub-levels as necessary.

SAM Compared with ACQ.4

Automotive SPICE requires more details than CMMI with respect to the cooperation between customer and supplier:

- common processes and interfaces (e.g. regarding PP, PMC, QA, Testing, etc)
- regular exchange of information
- joint reviews

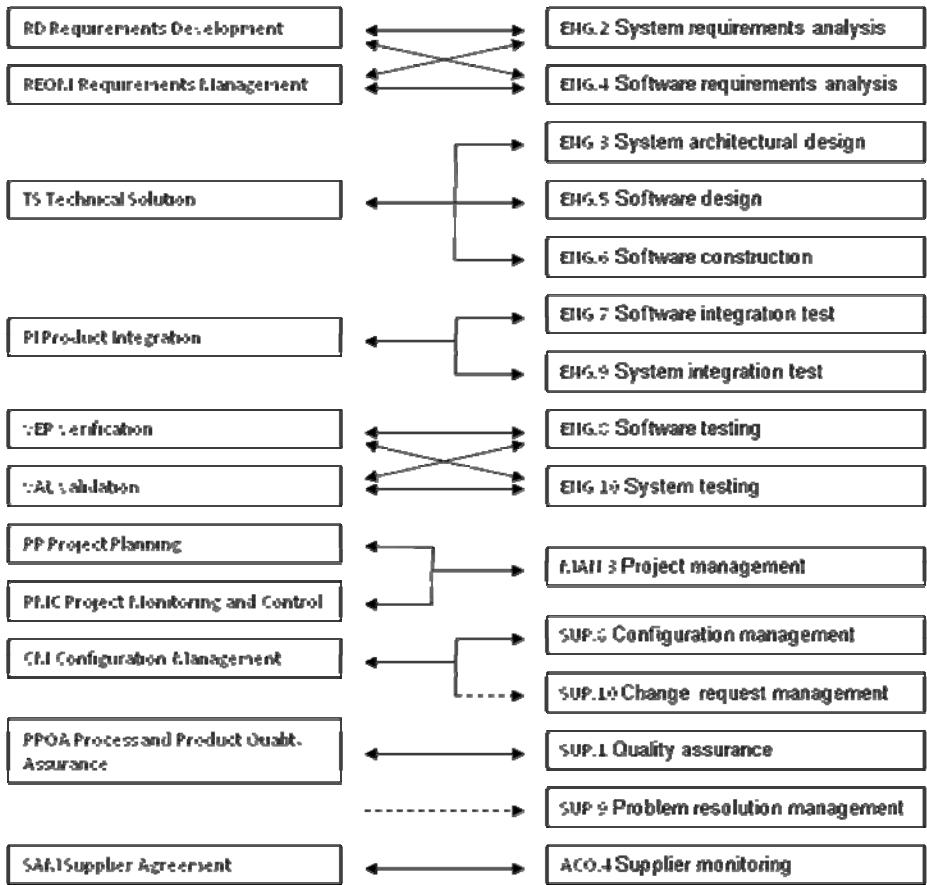


Fig. 1. Mapping CMMI / Automotive SPICE (HIS Scope)

- tracking of open issues and corrective actions
- change request management

PPQA Compared with SUP.1

- CMMI requires quality assurance strategy and plan on Level 2. Automotive SPICE requires them already on Level 1.
- Automotive SPICE requires explicitly that the organizational structure for quality assurance is independent from project management. CMMI claims just objectivity in this regard.
- CMMI does not require any escalation mechanism.

CM Compared with SUP.8, SUP.10

- CMMI requires configuration management on Level 2. Automotive SPICE requires it already on Level 1.
- CMMI does not require a branch management strategy

- CMMI does not require managing backup, storage, archiving, handling and delivery
- The practices to be implemented for the process SUP.10 (Change Management) are only partially addressed by CMMI within informative text.

TS Compared with ENG.3, ENG.5, ENG.6

- Verification of design and code is not addressed by TS but by the process areas VER/VAL. VER/VAL do not specify which products need to be verified and validated. This decision is up to the user.
- Communication mechanisms for disseminating the design are required by Automotive SPICE on Level 1. CMMI requires such mechanisms only indirectly through the generic practice “Stakeholder Involvement” on Level 2.
- Traceability requirements are addressed by CMMI within REQM not TS. Automotive SPICE presents much more explicit requirements with respect to traceability.
- CMMI does not require any description of dynamic behaviour.
- CMMI does not require any description of the objectives regarding resource consumption.
- Test criteria (ENG.5) and unit verification strategy (ENG.6) are addressed by CMMI in VER/VAL.
- In general, Automotive SPICE covers three different abstraction levels, i.e., system architectural design, software design, and software construction. CMMI covers only two levels explicitly, i.e., design and implementation. The different levels within design are described in informative text.

PI Compared with ENG.7, ENG.9

- Automotive SPICE requirements for planning, performing and documenting integration testing are more detailed.
- Traceability is explicitly required by Automotive SPICE. REQM not PI covers traceability issues.

4 Discussion of Differences between the Models

Both CMMI and Automotive SPICE aim at assessing and improving processes. The two models present, therefore, a great overlap in terms of concepts and content.

Process improvement is an expensive and time-consuming task. Therefore, if a company driving process improvement based on CMMI shall be assessed on the basis of Automotive SPICE, the CMMI-based processes must be utilized to the highest degree possible to demonstrate Automotive SPICE compliance. However, this is not an easy task, since the content of the two models is structured differently and their focus is placed on different subjects in some cases. The engineering processes, for instance, are addressed in Automotive SPICE by ten different processes, i.e., ENG.1-ENG.10. The same scope is addressed in CMMI by only six process areas (REQM, RD, TS, PI, VER, and VAL). As a further example, the process SUP.9 Problem resolution management is not addressed at all by CMMI.

As a consequence, there is quite a number of model differences to be resolved before an organization can achieve good results in an Automotive SPICE assessment. This is typically true even if the organization has already achieved good results (e.g., Maturity Level 3 and up) in previous CMMI appraisals.

5 Lessons Learned

The approach presented here is a combination of service elements which – each alone – have been developed over many years in a large number of customer projects. These customers were in different industries, most of them in the automotive industry. Because of the obvious benefits of each of the service elements it was a logical consequence to combine them into one service offering. We started offering this combination in 2006. Our lessons learned with the individual service elements are:

1. **Training:** Model training of some form is used in every improvement project, it is an integral part of it. How should one understand what is required by a model without being appropriately trained? So far we have performed many hundreds of such trainings. Regarding Automotive SPICE our experience is that usually a two days training for the improvement team is appropriate and a one day training for the staff members which are going to be interviewed. In case of a long-term improvement effort these trainings are performed long before any assessment activities start. In case of a pure assessment preparation (typically preparation for a supplier assessment) it can be rather short before the assessment due to schedule reasons.
2. **Mapping between models:** As soon as a new model version appears on the market we do usually have customer requests for advising them if these changes are relevant and what they should change. This is mostly relevant to those of our customers in the automotive industry who work according to CMMI and need to be compliant to Automotive SPICE. This was the case for recent changes from CMMI v1.1 to v1.2 and from Automotive SPICE PAM v2.3 to v2.3. Our experience with these mappings is that their benefit is limited. The reason for this is that these mappings can never tell you what exactly you have to change, they can only help you focus your attention for a more detailed analysis of your processes. Only this detailed analysis will unveil what precisely has to be changed.
3. **Mappings between the organization's processes and the model(s):** This is a standard recommendation we gave to most of our customers because it has so many advantages: it helps tracking progress when implementing the model, identifying risks and gaps still to be closed etc. It also helps later during process maintenance to prevent losing compliance to a model. There were certainly more than ten customers who followed this advice. Typically these were those customers who took it really seriously and also had the resources and budget for such activities. We believe that this type of mapping saves a tremendous amount of effort in case of a preparation for an assessment using a different model. The reason is that it allows, together with the previous mapping outlined in item 2 above, the very fast identification of candidates

for process enhancements to accommodate to the new model or to changes of a model.

4. **Gap Analysis:** We have performed certainly more than 100 gap analyses for CMMI, SPICE and Automotive SPICE. Most of them have been internally within a process improvement initiative to track progress and verify readiness for the final (external) assessments. Others have been for other purposes such as supplier assessments or the external validation that an improvement initiative was successful. Within the context of this paper the gap analysis is strongly recommended for a serious assessment preparation because it checks upon the differences between theory (how the processes should be, from their definition) and practice (how the processes are performed in reality). There are good reasons why there can be differences as outlined before.
5. **Improvement Workshop:** We have done this more than fifty times after performing assessments and appraisals. It allows explaining the gaps to the process improvement team and is especially necessary for less experienced organizations.

As mentioned before, there is little experience so far with organizations having adopted all five service element. However, our experience is that the more of these service elements are adopted, the higher is the probability to perform well in an Automotive SPICE assessment. We have seen this in at least six assessments of automotive companies being on CMMI Maturity Level 3 how smooth it went in a preparation phase of two to four months to pass Automotive SPICE supplier assessments with Capability Level three in all processes of the HIS scope.

6 Conclusions

In this paper, we addressed the problem of multiple models for process improvement to be applied within one single organization. The problem arises in all those cases in which different customer groups ask for compliance with different models. This is, for instance, the case in the automotive sector where many car manufacturers require process capability ratings determined on the basis of Automotive SPICE whereas several suppliers improve their processes on the basis of CMMI.

Automotive SPICE and CMMI address process assessment and improvement in a similar way and present a great overlap in terms of concepts and content. Nevertheless, they also show many differences that need to be addressed appropriately when preparing Automotive SPICE assessments.

In this paper, a brief overview of approaches for process assessment and improvement was given and the challenges were introduced that arise when implementing Automotive SPICE on top of CMMI. An approach was described which combines several good practices in preparing for Automotive SPICE assessments. Also, some lessons learned from applying this approach in different customer projects were presented.

In our experience, each of the individual elements of this approach increases the probability to pass an Automotive SPICE assessment successfully.

Acknowledgments. The authors would like to thank the following KUGLER MAAG CIE consultant colleagues for their experience reports and valuable comments: Dr. Karl-Heinz Augenstein, Dr. Michael Faeustle, Dr. Kurt Flad, Dr. Ewin Petry, and Mr. Dieter Wachendorf.

References

1. Automotive SIG: Automotive SPICE™ Process Assessment Model (PAM), RELEASE v2.3 (May 5, 2007)
2. Automotive SIG: Automotive SPICE™ Process Reference Model (PRM), RELEASE v4.3 (May 5, 2007)
3. Dorling, A.: CMMi Mapping to ISO/IEC TR 15504-2:1998 (last visited February 3, 2008), <http://www.isospice.com/articles/33/1/CMMi-Mapping-to-ISOIEC-TR-15504-21998/Page1.html>
4. Paulk, M., Weber, C., Garcia, S., Chrissis, M., Bush, M.: Key practices of the Capability Maturity Model, Version 1.1, Technical Report CMU/SEI-93-TR-025, Software Engineering Institute, Carnegie Mellon University (1993)
5. International Organization for Standardization (ISO): ISO/IEC 12207:1995/Amd.2:2004(E): Information technology - Software life cycle processes. Amendment 2. Genf (2004)
6. International Organization for Standardization (ISO): ISO/IEC 15288:2002(E): Systems engineering - System life cycle processes. Genf (2002)
7. International Organization for Standardization (ISO): ISO/IEC 15504-2:2003(E): Information Technology - Process assessment - Part 2: Performing an assessment. Geneve (2003)
8. International Organization for Standardization (ISO): ISO/IEC 15504-5:2006(E): Information technology - Process assessment - Part 5: An exemplar process assessment model. Genf (2006)
9. Rout, T.P., El Emam, K., Fusani, M., Goldenson, D., Jung, H.: SPICE in retrospect: Developing a standard for process assessment. *J. Syst. Softw.* 80(9), 1483–1493 (2007), <http://dx.doi.org/10.1016/j.jss.2007.01.045>
10. Sassenburg, H., Kitson, D.: A Comparative Analysis of CMMI and Automotive SPICE. European SEPG, Amsterdam/Netherlands (June 2006)
11. Software Engineering Institute: CMMI for Acquisition, Version 1.2 (November 2007)
12. Software Engineering Institute: CMMI for Development, Version 1.2 (August 2006)

A Model for Requirements Change Management: Implementation of CMMI Level 2 Specific Practice

Mahmood Niazi¹, Charles Hickman¹, Rashid Ahmad², and Muhammad Ali Babar³

¹ School of Computing and Mathematics, Keele University, ST5 5BG, UK
mkniazi@cs.keele.ac.uk, u2i42@ugl.keele.ac.uk

² College of EME, National University of Science & Technology, Rawalpindi, Pakistan
rashid@ceme.edu.pk

³ Lero, University of Limerick, Ireland
muhammad.alibabar@ul.ie

Abstract. OBJECTIVE – The objective of this research is to implement CMMI Level 2 specific practice – SP 1.3-1 manage requirements changes. In this paper we have proposed a model for requirements change management and also discussed initial validation of this model. This model is based on both an empirical study that we have carried out and our extensive literature review of software process improvement (SPI) and requirements engineering (RE).

METHOD – For data collection we have interviewed SPI experts from reputed organisations. Further work includes analysing research articles, published experience reports and case studies. The initial evaluation of the model was performed via an expert review process.

RESULTS – Our model is based on five core elements identified from literature and interviews: request, validate, implement, verify and update. Within each of these elements we have identified specific activities that need to take place during requirements change management process.

CONCLUSIONS – The initial evaluation of the model shows that the requirements change management model is clear, easy to use and can effectively manage the requirements change process. However, more case studies are needed to evaluate this model in order to further evaluate its effectiveness in the domain of RE process.

1 Introduction

Software Process Improvement (SPI) has been a long-standing approach promoted by software engineering researchers, intended to help organisations develop higher-quality software more efficiently. Process capability maturity models such as CMM, CMMI (Chrissis et al., 2003) and ISO/IEC 15504 (SPICE) are SPI frameworks for defining and measuring processes and practices that can be used by software developing organisations. However, the population of organisations that have adopted process capability maturity model is only a part of the entire population of software-developing organisations [1]. Deployment is often not only multi-project, but multi-site and multi-customer and the whole SPI initiative typically requires a long-term

approach. It takes significant time to fully implement an SPI initiative [2-5]. The failure rate of SPI initiatives is also very high, estimated as 70% [6; 7]. The significant investment and limited success are reasons for many organisations being reluctant to embark on a long path of systematic process improvement.

CMMI is the successor to CMM and is consistent with the international standard ISO/IEC 15504. The most well-known representation of CMMI is the “staged” representation, which has five “levels” of process maturity for organisations. Each of the five levels is composed of several process areas – for each process area, several goals are defined which contain different practices. For an organisation to reach a maturity level, they must satisfy the goals of the process areas for that level and all lower levels. The practices help an organisation understand how to achieve maturity goals and serve as examples of the activities to be addressed when undertaking a SPI programme.

Level 2 is the first level that defines a collection of process capabilities that largely focus on supporting process areas, but also includes some project management and engineering process areas. There are two goals in Level 2: Specific Goal one (SG1) - Manage Requirements and Generic Goal two (GG2) – Institutionalize Managed Process. SG1 contains five specific practices (SP) of which SP1.3-1 – manage requirements changes is the key practice.

This paper reports on the implementation of the specific practice - manage requirements changes - of CMMI Level 2. We have developed a model for requirements change management and have done initial validation of this model. The major contributions of this paper are:

- i. to present a requirements change management model in order to effectively manage requirements changes.
- ii. to evaluate the requirements change management model via an expert panel review process.

To achieve these objectives, we address the following research questions which are based on the Technology Acceptance Model (TAM) [8; 9]:

- RQ1. How can one implement CMMI Level 2 specific practice - manage requirements changes?
- RQ2. What is the perceived “ease of learning” of the outcome of the requirements change management practice implementation?
- RQ3. What is the “perceived usefulness” of the outcome of the requirements change management practice implementation?

This paper is organised as follows, Section 2 provides the background to the research. Section 3 describes the study method. In Section 4 the development of requirements change management model is described. Section 5 gives evaluation of our model. Section 6 presents conclusions and future work.

2 Background

Requirements engineering (RE) is concerned with describing a client’s problem domain (the context), determining the desired effects the client wants to exert upon that

domain (the requirements) and specifying the proposed Information Technology (IT) (the specification) to a) help enable those desired effects to occur and b) to give designers a specification to help them build the proposed IT. Thus the RE process has a huge impact on the effectiveness of the software development process [10]. When RE processes are ad hoc, poorly defined or poorly executed, the end product is typically unsatisfactory [11].

The Standish group reported that, on average, the percentage of software projects completed on-time and within budget has improved from 16% in 1995 [12] to 34% in 2003 [13]. However, nearly two-thirds of the projects examined in the 2003 report [13] were 'challenged' (i.e. only partially successful) with the authors observing that one of the main reasons for project failure is unstable requirements caused by poor management of RE processes. Several other studies have also identified problems with the RE process [10; 14-20]. A UK study found that of 268 documented development problems, requirements problems accounted for 48% of those problems [14].

The actual effort in requirements engineering is not very large. Alexander and Stevens [21] recommend that about 5% of project effort goes into requirements effort (elicitation, analysis, verification, validation, testing), not including specification. This might be about 25% of calendar time (or no more than three months dependent upon project size). They state that system specification might also take 20-25% of calendar time. Hoffmann and Lehner [22] examined 15 projects and found they expended on average 16% of project effort on RE activities. Chatzoglou and Macaulay [23] surveyed 107 projects and found requirements capture and analysis took over 15% of total elapsed time. In a study of 16 software projects, MacDonell and Shepherd [24] found that there was so much variance in effort in Project Planning and Requirements Specification phases and in comparison with overall project effort that no patterns could be drawn from it, except that without the requirements phase, or with a poor requirements phase, the project was not successful.

Software development is a dynamic process. It is widely reported that requirements often change during the software/system development process. These changes are inevitable and driven by several factors including ambiguities in original requirements, evolving customer needs, constant changes in software and system requirements, business goals, work environment and government regulation [25]. Volatile requirements are regarded as a factor that cause major difficulties during system development for most organisations in the software industry [26]. Simulation models of software development projects demonstrate that requirements volatility has a significant impact on development effort and project duration [27; 28]. Furthermore, volatile requirements contribute to the problems of software project schedule overruns and may ultimately contribute to software project failure [26; 29; 30].

Despite the importance of the requirements change management, little empirical research has been carried out in the domain of SPI on developing ways to effectively implement this specific practice (such as manage requirements changes) of SPI models such as CMMI. We have focused on this specific practice and designed a requirements change management model in this paper.

3 Study Method

We have used two data collection sources, i.e. SPI literature and requirements change management practices in two organisations. We have analysed and evaluated the data collected from both sources in order to produce a requirements change management model for implementing a CMMI level 2 specific practice.

The SPI literature considered in this research includes case studies, experience reports and high-level software process descriptions. Most of the reviewed studies report real life experiences of SPI implementation and provide specific guidelines and recommendations for SPI implementation. The data extraction phase of the literature review was confined to the material related to the description of the processes and practices of managing the requirements changes. Each piece of the extracted data was reviewed carefully and a list of reported characteristics of an effective process of managing requirements was produced. While every effort was made to control the researchers' bias during the literature search, papers selections, data extraction, and analysis phases, we do not claim that we followed a systematic process of reviewing the literature as recommended by evidence-based software engineering paradigm [31].

For collecting data about the requirement change management practices in two organisations, we used semi-structured interviews with one representative from each company. The interviewees were nominated by their respective companies as they were considered the most experienced and knowledgeable requirements engineers. Hence, they were considered the most appropriate persons to answer questions about requirements change processes in their respective organisations. Thus the sample is not random but a convenience sample because we sought a response from a person with a specific role within the organisation. In order to identify a suitable and relevant set of questions for the interview instrument, we consulted a number of books and research articles [4; 14; 32-37]. After designing a list of questions, we assessed their suitability by mapped them on the objectives of our research project. One of our colleagues also reviewed the interview instrument and helped us to improve it. For confidentiality reasons, we are not allowed to report the names of the companies whose requirements change management processes were studied. Hence, we will call them Company A and Company B in this paper.

Company A has more than two hundred employees that are directly employed for software production and/or maintenance. The company has an outstanding profile and repute earning history for more than 10 years. The company is predominantly concerned with logistics for outsource development. The scope of the company is multinational. The company believes that it is at CMMI level three but it has not been certified for CMMI maturity level by external auditors.

Company B is a small company with less than 20 professionals who are directly concerned with software production and/or maintenance. This company is predominantly concerned with embedded systems for in-house development for the last two years.

One of the researchers has conducted the interviews in face-to-face meeting sessions. The interviews consisted of a short series of questions in the form of a questionnaire (available from authors upon request), which allowed us to gather information on the demographics of the company followed by a list of questions about requirements change management processes. Each interview lasted for approximately

half an hour. We have used a Dictaphone to record the responses of the interviewees and also took extensive notes during the interviews.

To analyse the interviewees' responses, we have read the notes taken during the interviews in order to consolidate the important point that were made about requirements change management process during the interviews. We have also listened to the tapes many times in order to ensure that we have not missed out anything important relating to the requirements change management. This two steps process has given confidence that the transcription process has not changed the original data generated in the interviews.

4 Results

4.1 Findings from Literature

We have identified three requirements models from the literature suitable for requirements change management: the spiral-like change management process [38], Olsen's change management model [39], and Ince's change process model [40]. We believe these models can be adapted in order to implement the CMMI level 2 specific practice - requirements change management. This is because these are normative models that deal with requirements change management effectively. In the next sections, we provide brief reviews of these models.

4.1.1 The Spiral-Like Change Management Process [38]

This model divides the change management process into four cycles as shown in Figure 1:

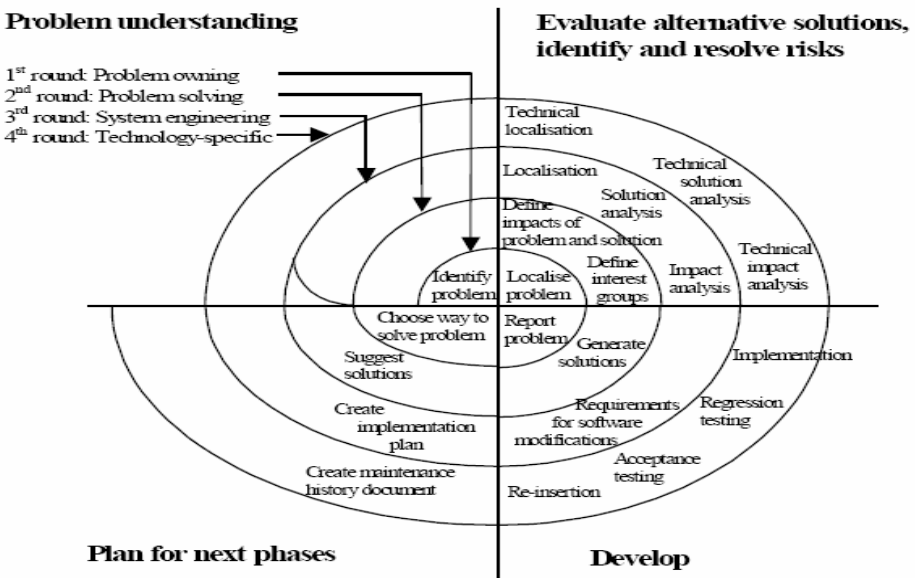


Fig. 1. The spiral-like change management process [38]

- i. 1st round: Problem owning
- ii. 2nd round: Problem solving
- iii. 3rd round: System engineering
- iv. 4th round: Technology-specific

Round 1 of this model is the initial cycle; the founder or owner of a problem begins this cycle. A problem can be a request to add a new feature or services in the system or to fix a problem in the existing system. At the end of the first cycle the owner decides whether a change needs to be made and if the change is deemed necessary, how it should be accommodated. Round 2 is only required if the change needs to be looked at from a non-technical viewpoint. Round 3 is the planning stage. It examines the change from a system point of view and makes an implementation plans for the round 4. Round 4 generates, implements and verifies the technical solution. The change is finished and the results of this change are recorded.

4.1.2 Olsen’s Change Management Model [39]

Olsen views the whole software development process as simply a queue of changes that need to be made as shown in Figure 2. Olsen believes that all work done by software designers changes. This model can be applied to both software development and maintenance as it is not life cycle dependent. The sources of changes are made available by the users who suggest possible requirement changes. These changes are then passed to the “manage change” section where these changes are managed by

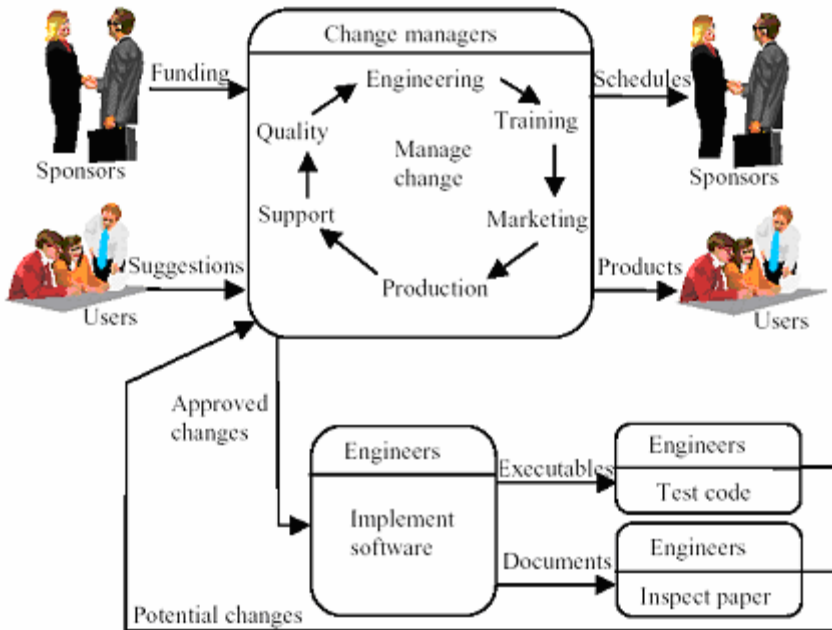


Fig. 2. Olsen’s change management model [39]

change managers. The approved changes are passed on to the implementation section where necessary changes are made in the software. After completing implementation, verification begins by testing code and by inspecting papers. When a change has been implemented and verified it is then passed back to change managers who will then release the change in a product for its users.

4.1.3 Ince's Change Process Model [40]

Ince's model focuses on how software configuration management relates to software change management. This model has two main sources of change requests, i.e. customer and development team as shown in Figure 3. In order for the change process to be initiated, a change request must be initiated in a software project. All such change requests are recorded in a change request note. The change control board then considers the suggested change. The change control board can reject the change (the change will not take place), batch the change (the change will take place but not immediately) or accept the change (the change is to be implemented at the earliest

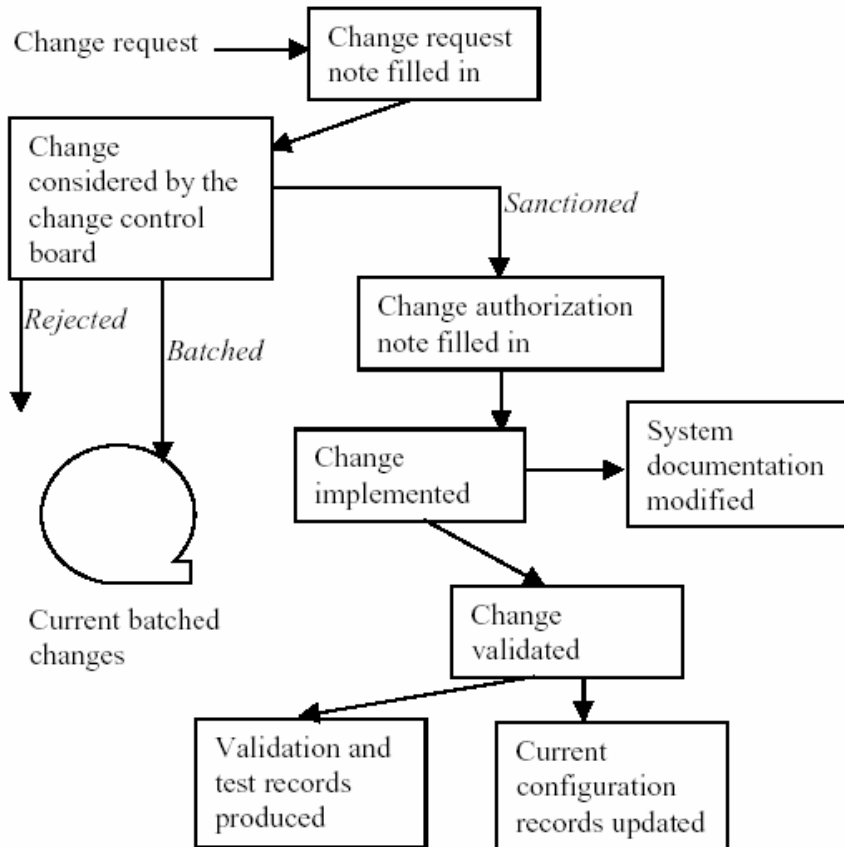


Fig. 3. Ince's change process model [40]

possible time). If the request for the change is successful, a change authorisation note must be filled. After this the change can be implemented and a system's documentation is modified. After implementation the change is validated. Validation and test records are then produced to document the changes that have been taken place. Finally the configuration records are updated and the staff is informed about the new changes.

4.2 Finding from Companies

Having discussed the findings from reviewing the literature, we now present the findings about the requirements change management processes of two companies based on analyzing of the data gathered through interviews with two requirements engineering experts of those companies.

4.2.1 Company A's Requirements Change Model

We shall now discuss the key points about the requirements change management process of Company A. Figure 4 shows the process company A follows to manage requirements changes. The findings from the interview are:

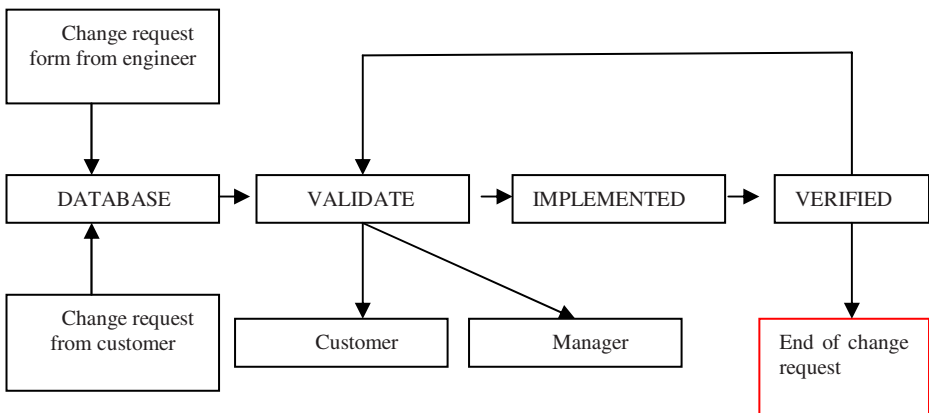


Fig. 4. Company A's requirements change model

- Company A follows a predefined process when changes are required. Changes to the system are done through formal configuration management using a tool called Redbox.
- Staff must fill in change request forms before any processes can begin.
- Company A has in place a process (traceability) that confirms the requirement back to the customer to make sure that the company knows exactly what the customer desires. All requirements that come in are assessed and an outline design document is produced. This document is checked to ensure it corresponds to the customer's requirements. This document is then passed onto a change control board. Final step is the system confirmation which ensures that the workforce clearly understands what changes are required.

- Company A contains multiple stages at which testing can be done. The first kind of testing is ‘unit testing’ which is internal testing. An in-house team specialised in system and regression testing performs this testing. The second type is called “field trials” where they release software into specific locations for specific groups of users. Another type is called “user acceptance testing” where the company performs testing with their customer.
- The reasons for changes are primarily customer driven. The main reason is to enable general enhancements to business process. These can be minor or major enhancements. Minor enhancements are considered standard software maintenance when just small tweaks are needed. Major enhancements are system rewrites or amending large parts of the systems. Reasons for changes can vary from project to project and the majority are minor changes.

4.2.2 Company B’s Requirements Change Model

In this section, we discuss the key points that we have extracted from the interview followed by our interpretation of how Company B manages its requirements changes as shown in Figure 5.

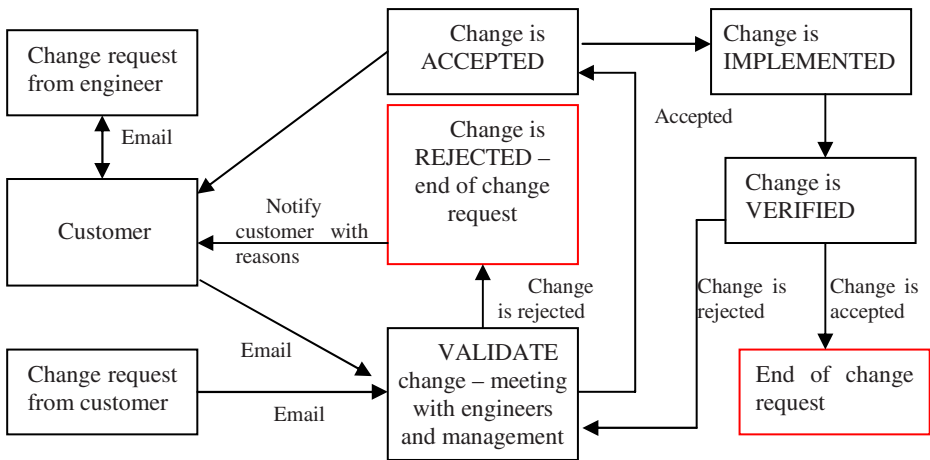


Fig. 5. Company B’s requirements change model

- Company B does not follow any set process or model to manage requirements changes. Requirements changes are done on an informal basis mainly using emails to communicate. Customers email changes if required. These requests are then evaluated to see if the requested changes can be made. Software Engineers can also suggest possible changes to the customers via emails.
- The company does not use a database to store change requests.
- The company does not use change request forms.

- All requests are forwarded to all the members of the management and development teams. Every member offers comments on the validity of the request and identifies ramifications of these changes, if any, on the existing software.
- Each request is handed over to the developer who will then implement the change.
- Verification is done by the person who implements the change. At the end of the project the whole team double-checks the software for any problems or issues.
- The most common reasons for making any changes are functionality enhancements. Bug fixes are also required.
- All changes are treated the same regardless of their size.

4.3 Our Requirements Change Model (RCM)

RCM development was initiated by creating and agreeing its success criteria. Objectives were set to clarify the purpose of the model and outline what the model is expected to describe. These criteria guided development and are later used to help evaluate the RCM.

The first stage in the development of RCM was to set criteria for its success. The motivation for setting these criteria comes from previously conducted empirical studies [36; 41] and by a consideration of the Technology Acceptance Model [8; 9]. The following criteria were used.

- User satisfaction: stakeholders need to be satisfied with the results of the RCM. Stakeholders (e.g. requirements engineers, systems analysts, outsourcing project staff) should be able to use the RCM to achieve specified goals according to their needs and expectations without confusion or ambiguity.
- Ease of use: complex models and standards are rarely adopted by organisations as they require resources, training and effort. The structure of the RCM should be simple, flexible and easy to follow.

In the second stage, in order to address these desired criteria, research questions (see Section 1) were developed. In order to answer research questions, in the third stage, we have extensively reviewed the RE literature and conducted interviews with two RE experts. In the final stage, based on an extensive literature review and findings from the interviews, we have developed a model of Requirements Change Management for a CMMI level 2 specific practice. The model is based on five core elements: request, validate, implement, verify and update as shown in Figure 6. Within each of these elements, we have identified a set of specific activities that need to take place during requirements change management process.

The initial stage is the change “Request”. We have decided to include this element in the RCM as this element was found in the both companies’ requirements change management processes as well as also in Ince change model [40]. The main sources of requests may be either internal or external. The internal requests come from the project management or software maintenance teams within the company. The external requests come from the customers. These internal and external requests are then fed to a requirements change pool. The requirements change pool contains a description of the change, the reasons behind the changes and who has requested the change.

The next stage is to “Validate” the change request. The validation of the change request was cited in our two interviews and also in the Spiral-like change management process and the Ince change model [38; 40]. The first activity in the validate stage is to understand the change request (i.e. what needs to be done – fix, an enhancement or removal). Request analysis is the activity to look at the different ways in which the request can be met, i.e. how much effort is needed to make this change, how much effort is needed to implement this change, the impact of the change, the risk of change and the priority of each change request. In addition, in the validation stage it is also analysed if the change request is:

- consistent with the business goals of the organisations;
- not ambiguous i.e. could it be read in different ways by different people;
- feasible in the context of the budget and schedule available for the system development;
- consistent with other requirements of the system.

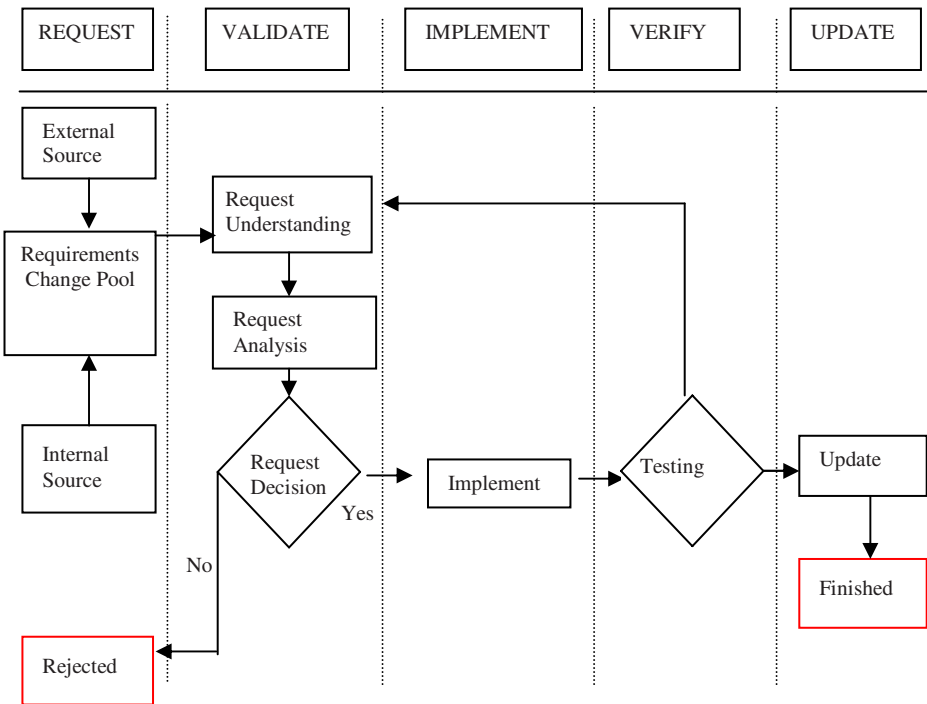


Fig. 6. Requirements change management model

The final activity of the validation process is to make decision if the change request should be accepted, rejected or reanalysed with new evidence.

The third stage is to “Implement” the changes. In this stage all the accepted changes are consolidated and are implemented in the form of end product or software.

The fourth stage is to “Verify” changes where it is ascertained that the software conforms to its specification. The verification of the change request was cited in our two interviews and also in the Olsen’s change management model [39]. In this stage the new product/software is tested in the form of regression testing, field trials or user acceptance. The testing method will depend on the characteristics of the request and the environment of the product/software. If the verification stage is not successful the change request is sent back to the “Validate” stage for further understanding and analysis of the change request.

The final stage is “Update”. Documentation on the software is updated with the changes made. The customers and the project team are all informed about these update so that everyone is working on the current up to date version. The finished step is when the product/software is released with the new requirements included.

5 Model Evaluation

The initial evaluation of the model was performed via an expert review process. The expert panel review process was used in order to seek opinions of two SPI experts about the “ease of learning” and “user satisfaction” [8; 9] of the proposed requirements change model. One SPI expert has 25 years of experience in software development and SPI. The second expert has 6 years of experience in software development and SPI related activities.

In order to seek SPI experts’ opinion about requirements change model, a questionnaire was designed in which some questions were taken from [42-44] and tailored to fit into this research project goals. This questionnaire is divided into two parts, i.e. demographic and model feedback.

Before sending out this questionnaire to the SPI experts, drafts questionnaire were reviewed by two researchers. These researchers were asked to critically evaluate the questions against “ease of learning” and “user satisfaction”. Based on their feedback, some questions were re-written in order to better capture the required data. The questionnaire was tested by two researchers before sending requests to the experts.

- **Ease of Learning:** Both experts rated the model as clear and easy to understand. Neither expert felt that a great deal of prior knowledge of SPI was needed to understand the proposed model. Both experts also felt that the division of the model into five core sections aided them in their understanding. This was encouraging as it showed that our model was concise and comprehensive.
- **User Satisfaction:** Both experts felt that in general our model would be useful within the software industry. Neither of the experts thought that any key part was missing from our requirements change management model. Both experts felt that the requirements change management model is clear and can effectively manage the requirements change process. However, other companies might need to adapt this model in order to fulfil their specific requirements.

Based on the initial evaluation, we are confident that the proposed model can help organisations to implementing requirement change management process according to CMMI level 2 maturity requirement, however, we are also realize the need for further

evaluation of the model to rigorously assess its various elements. We plan to perform this evaluation through multiple case studies in the industrial context.

6 Conclusion

The objective of this research is to develop and empirically assess a model that would help organisation to effectively manage requirements changes. For this purpose, we identified following research questions to be addressed by the reported research:

- RQ1. How can one implement CMMI Level 2 specific practice - manage requirements changes?
- RQ2. What is the perceived “ease of learning” of the outcome of the requirements change management practice implementation?
- RQ3. What is the “perceived usefulness” of the outcome of the requirements change management practice implementation?

In order to address the RQ1, we have developed a requirements change management model based on literature review and two companies’ processes of managing requirements changes. During the literature review, we analysed the published experience reports, case studies and articles to identify a list of characteristics required to effectively manage requirements change process. We have identified three requirements models from the literature and we believe these models can be adapted in order to implement the CMMI level 2 specific practice - requirements change management. Our interviews with two companies’ representative provided us with interesting insights into their requirements change management processes.

In order to address the RQ2 and RQ3, we performed an initial evaluation of the proposed model using the expert review process. We sought the opinions of two SPI experts about the “ease of learning” and “user satisfaction” [8; 9] of the proposed requirements change model. Both experts rated the model as clear and easy to understand. Both also experts felt that in general our model would be useful within the software industry. However, it was noted that some companies might need to adapt this model to their specific requirements.

For further improvement and rigorous evaluation, we plan to conduct multiple case studies in industrial setting to trial the proposed model.

References

1. Leung, H.: Slow change of information system development practice. *Software quality journal* 8(3), 197–210 (1999)
2. SEI: Process Maturity Profile. Software Engineering Institute Carnegie Mellon University (2004)
3. Niazi, M., Wilson, D., Zowghi, D.: Critical Barriers for SPI Implementation: An empirical study. In: *IASTED International Conference on Software Engineering (SE 2004)*, Austria, pp. 389–395 (2004)
4. Niazi, M., Wilson, D., Zowghi, D.: Critical Success Factors for Software Process Improvement: An Empirical Study. *Software Process Improvement and Practice Journal* 11(2), 193–211 (2006)

5. Niazi, M., Wilson, D., Zowghi, D.: Implementing Software Process Improvement Initiatives: An empirical study. In: *The 7th International Conference on Product Focused Software Process Improvement*. LNCS, pp. 222–233 (2006)
6. SEI: Process maturity profile of the software community. Software Engineering Institute (2002)
7. Ngwenyama, O., Nielsen, P.v.: Competing values in software process improvement: An assumption analysis of CMM from an organizational culture perspective. *IEEE Transactions on Software Engineering* 50, 100–112 (2003)
8. Davis, F.D., Bagozzi, R.P., Warshaw, P.R.: User acceptance of computer technology: A comparison of two theoretical models. *Management Science* 35, 982–1003 (1989)
9. Davis, F.D.: Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology. *MIS Quarterly* 13(3), 319–340 (1989)
10. El Emam, K., Madhavji, H.N.: A Field Study of Requirements Engineering Practices in Information Systems Development. In: *Second International Symposium on Requirements Engineering*, pp. 68–80 (1995)
11. Standish-Group: *Chaos: A Recipe for Success*. Standish Group International (1999)
12. Standish-Group: *Chaos - the state of the software industry*. Standish group international technical report, pp. 1–11 (1995)
13. Standish-Group: *Chaos - the state of the software industry* (2003)
14. Hall, T., Beecham, S., Rainer, A.: Requirements Problems in Twelve Software Companies: An Empirical Analysis. *IEE Proceedings - Software*, 153–160 (2002)
15. Kamsties, E., Hormann, K., Schlich, M.: Requirements Engineering in Small and Medium Enterprises. *Requirements Engineering* 3(2), 84–90 (1998)
16. Nikula, U., Fajaniemi, J., Kalviainen, H.: Management View on Current Requirements Engineering Practices in Small and Medium Enterprises. In: *Fifth Australian Workshop on Requirements Engineering*, pp. 81–89 (2000)
17. Nuseibeh, B., Easterbrook, S.: Requirements Engineering: a roadmap. In: *22nd International Conference on Software Engineering*, pp. 35–46 (2000)
18. Siddiqi, J., Chandra, S.: Requirements Engineering: The Emerging Wisdom. *IEEE Software* 13(2), 15–19 (1996)
19. Beecham, S., Hall, T., Rainer, A.: Software Process Problems in Twelve Software Companies: An Empirical Analysis. *Empirical software engineering* 8, 7–42 (2003)
20. Niazi, M.: An empirical study for the improvement of requirements engineering process. In: *The 17th International Conference on Software Engineering and Knowledge Engineering*, Taipei, Taiwan, Republic of China July 14–16, pp. 396–399 (2005)
21. Alexander, I., Stevens, R.: *Writing Better Requirements*. Addison-Wesley, Reading (2002)
22. Hoffmann, H., Lehner, F.: Requirements Engineering as a Success Factor in Software Projects. *IEEE Software*, 58–66 (July/August 2001)
23. Chatzoglou, P., Macaulay, L.: Requirements Capture and Analysis: A Survey of Current Practice. *Requirements Engineering Journal* 1, 75–87 (1996)
24. MacDonell, S., Shepperd, M.: Using Prior-Phase Effort Records for Re-estimation During Software Projects. In: *9th Int. Symp on Software Metrics*, Sydney, Australia, September 3–5, 2003, pp. 73–86 (2003)
25. Barry, E.J., Mukhopadhyay, T., Slaughter, S.A.: Software Project Duration and Effort: An Empirical Study. *Information Technology and Management* 3(1-2), 113–136 (2002)
26. Zowghi, D., Nurmuliani, N.: A study of the impact of requirements volatility on software project performance. In: *Ninth Asia-Pacific Software Engineering Conference*, pp. 3–11 (2002)

27. Pfahl, D., Lebsanft, K.: Using simulation to analyse the impact of software requirement volatility on project performance. *Information and Software Technology Journal* 42, 1001–1008 (2000)
28. Ferreira, S., Collofello, J., Shunk, D., Mackulak, G., Wolfe, P.: Utilization of Process Modeling and Simulation in Understanding the Effects of Requirements Volatility in Software Development. In: *International Workshop on Software Process Simulation and Modeling (ProSim 2003)*, Portland, USA (2003)
29. Stark, G., Skillicorn, A., Ameen, R.: An Examination of the Effects of Requirements Changes on Software Maintenance Releases. *Journal of Software Maintenance: Research and Practice* 11, 293–309 (1999)
30. Zowghi, D., Nurmiliani, N., Powell, S.: The Impact of Requirements Volatility on Software Development Lifecycle. In: *Proceedings of Software Engineering Conference, Australian*, pp. 28–37 (2004)
31. Kitchenham, B.: *Procedures for Performing Systematic Reviews*. Keele University, Technical Report TR/SE0401 (2004)
32. Chrissis, M., Konrad, M., Shrum, S.: *CMMI Guidelines for Process Integration and Product Improvement*. Addison-Wesley, Reading (2003)
33. Creswell, J.: *Research Design: Qualitative, quantitative and mixed methods approaches*. Sage, London (2002)
34. Kotonya, G., Sommerville, I.: *Requirements Engineering Processes and Techniques*. John Wiley, Chichester (1998)
35. Niazi, M., Cox, K., Verner, J.: An empirical study identifying high perceived value requirements engineering practices. In: *Fourteenth International Conference on Information Systems Development (ISD 2005)*, Karlstad University, Sweden, August 15–17 (2005)
36. Niazi, M., Cox, K., Verner, J.: A Measurement Framework for Assessing the Maturity of Requirements Engineering Process. *Software Quality Journal* (in press for publication, 2008)
37. Beecham, S., Hall, T., Rainer, A.: Building a requirements process improvement model. Department of Computer Science, University of Hertfordshire, Technical report No: 378 (2003)
38. Mäkäriäinen, M.: Application management requirements for embedded software. Technical Research Centre of Finland, VTT Publications 286 (1996)
39. Olsen, N.: The software rush hour. *IEEE Software*, 29–37 (September 1993)
40. Ince, D.: *Introduction to software quality assurance and its implementation*. McGraw-Hill, New York (1994)
41. Niazi, M., Wilson, D., Zowghi, D.: A Maturity Model for the Implementation of Software Process Improvement: An empirical study. *Journal of Systems and Software* 74(2), 155–172 (2005)
42. Beecham, S. and Hall, T.: Expert panel questionnaire: Validating a requirements process improvement model (May 2003), http://homepages.feis.herts.ac.uk/~pppgroup/requirements_cmm.htm
43. Rainer, A., Hall, T.: Key success factors for implementing software process improvement: a maturity-based analysis. *Journal of Systems & Software* (62), 71–84 (2002)
44. Niazi, M.: *A Framework for Assisting the Design of Effective Software Process Improvement Implementation Strategies*, PhD thesis, University of Technology Sydney (2004)

Experience Report on the Effect of Software Development Characteristics on Change Distribution

Anita Gupta¹, Reidar Conradi¹, Forrest Shull², Daniela Cruzes², Christopher Ackermann², Harald Rønneberg³, and Einar Landre³

¹ Dep. of Computer and Information Science (IDI), Norwegian University of Science and Technology (NTNU), Trondheim, Norway

{anitaash, conradi}@idi.ntnu.no

² Fraunhofer Center Maryland, College Park, USA

{fshull, dcruzes, cackermann}@fc-md.umd.edu

³ StatoilHydro ASA KTJ/IT, Forus, Stavanger

{haro, einla}@statoilhydro.com

Abstract. This paper reports on an industrial case study in a large Norwegian Oil and Gas company (StatoilHydro ASA) involving a reusable Java-class framework and an application that uses that framework. We analyzed software changes from three releases of the framework and the application. On the basis of our analysis of the data, we found that perfective and corrective changes account for the majority of changes in both the reusable framework and the non-reusable application. Although adaptive changes are more frequent and has longer active time in the reusable framework, it went through less refactoring compared to the non-reusable application. For the non-reusable application we saw preventive changes as more frequent and with longer active time. We also found that designing for reuse seems to lead to fewer changes, as well as we saw a positive effect on doing refactoring.

Keywords: Software reuse, Software quality, Software changes, Case Study.

1 Introduction

Understanding the issues within software evolution and maintenance has been a focus since the 70's. The aim has been to identify the origin of a change, as well as the frequency and cost in terms of effort. Software changes are important because they account for a major part of the costs of the software. At the same time, they are necessary; the ability to alter software quickly and reliably means that new business opportunities can be taken advantage of, and that businesses thereby can remain competitive [1].

Several previous studies have concluded that reusable software components are more stable (less change density) than non-reusable components [20-22]. However, few of these studies have investigated and compared the characteristics of software changes (such as distribution, how long the changes tend to stay in the system, and number of files modified for each change type) for reusable and non-reusable components. In the study described here we investigate whether aspects of software changes, such as their frequency, type, or difficulty, can be better understood based on:

- Characteristics of the process being applied (e.g. whether different change characteristics are seen when designing for reuse vs. designing for a specific context), and
- Characteristics of the product being built (e.g. whether different change characteristics are seen for systems before and after refactoring).

By “change characteristics” here we refer to attributes of the set of software changes made to a system over time, such as the relative frequency of different types of changes, the files of the system affected by the changes, and how the changes were implemented.

The case study described here is on the reuse process in the IT-department of a large Norwegian Oil & Gas company, StatoilHydro ASA¹. We have collected data from software changes for three releases of a reusable class framework called Java Enterprise Framework (JEF), as well as three releases of one application called Digital Cargo Files (DCF) that uses this framework “as-is”, without modification. All data in our study are software changes from the evolution (e.g. development) and maintenance phases for the three releases each of two systems.

The purpose of this study is to compare change characteristics across systems, with respect to the impact of reuse on change types, frequency, active time and localization of the effects of changes on the systems.

We were particularly interested in gaining insight into properties of systems being designed to be reusable, since that was a major focus for the reuse program at StatoilHydro ASA. The results are important in that they characterize and explain the changes to the reusable framework and the non-reusable application.

The paper is structured as follows. Section 2 presents the related work. Section 3 introduces the context in StatoilHydro ASA. Section 4 presents the motivation for the research and the research questions. Furthermore, Section 5 describes the research methodology. Section 6 presents the results and possible explanations of our analysis of software changes extracted from Rational ClearCase. Section 7 looks into the validity threats for our study. Section 8 states our conclusions.

2 Related Work

Lehman [2] carried out the first empirical work on software changes, finding that systems that operate in the real world have to be adapted continuously, otherwise, their changeability decreases rapidly. During the lifetime of software systems, they usually need to be changed as the original requirements may change to reflect changing business, user and customer needs [3]. Other changes occurring in a software system’s environment may emerge from undiscovered errors during system validation that require repair, from the introduction of new hardware.

Changes to software may be categorized into four classes based on the intent of making the change, namely corrective, adaptive, perfective and preventive. In general, corrective refers to fixing bugs, adaptive are related to new environments or platforms, while implementing altered or new requirements, as well as improving performance, can be classified as perfective. Finally, changes made to improve future

¹ ASA stands for “allmennaksjeselskap”, meaning Incorporated.

maintainability can be thought of as preventive [4]. Such taxonomy is useful because it captures the kind of situations that developers face over time. However, differences may exist in the definition of these change classes, which can make the comparison of studies difficult. We have in our study decided to use the definition given by [5]:

- *Adaptive* changes are those related to adapting to new platforms, environments or other applications.
- *Corrective* changes are those related to fixing bugs.
- *Perfective* changes are that that encompass new or changed requirements as well as optimizations.
- *Preventive* changes are those having to do with restructuring and reengineering.

Several studies have investigated the distributions of different changes (e.g. corrective, adaptive, perfective, and preventive) based on change logs of different systems. These studies show that:

- *The classifications of changes are different among different studies.* For example, some studies [6-11] have classified the changes into adaptive, corrective, and perfective; some of them have still a fourth category [9-11]. Other studies have classified changes into adaptive, preventive, and perfective [12-17] and four of these studies have classified changes into a fourth category: corrective [14-17]. One study has classified changes into planned enhancement, requirement modifications, optimization and “other” [18]. Yet another study has classified changes into user support, repair and enhancement [19].
- *Definitions of change types are different among different studies.* For example, perfective change has been defined to include user enhancements, improved documentation, and recoding for computational efficiency [6][7]. It is also defined as encompassing new or changed requirements (expanded system requirements) as well as optimization [12][13][15]. While, [10] has defined the same type as including enhancements, tuning and reengineering.
- *The distributions of changes are different for different systems.* For example, the most frequent changes of the studied system in [6][10][11] are perfective changes. However, perfective changes in the system in [7] are the least frequent ones. In the study conducted by [9][15] the most frequent changes are adaptive changes. While, in [18] the most frequent changes are in the category “other”.

Table 1 shows different studies and the most frequent changes found in the results. We also distinguish systems that were designed to be reused as part of another system. We can see that 64% of the studies have perfective changes as the most frequent ones, 21% have corrective changes, followed closely by 14% that have adaptive changes as the most frequent ones.

Other studies [20-25] have investigated whether the amount of changes varies according to development characteristics without classifying changes into different categories. We are aware of no previous studies that have compared change distributions between reusable software components and non-reusable ones, which we are looking at in this study.

Table 1. Related work

Reusable system?	Studied systems	Most frequent change types
No	System which has been operational for at least 1 year, represents a significant investment of time and effort, and is of fundamental importance to the organization [6].	Perfective changes
No	A case study investigating 2152 change requests collected for 2 years in a Canadian financial institute [9].	Adaptive changes
No	A survey conducted in the MIS department in 9 different business types in Hong Kong [10].	Perfective changes
No	Survey carried out in a computer department of a large Norwegian organisation in 1990-1991 (study1) and 1992-1993 (study2). The computer department studied maintains of more than 70 software applications and include 110 maintainers, distributed on 11 maintenance groups [14].	Perfective changes
No	Study of 10 projects conducted in Flight Dynamic Division (FDD) in NASA's Goddard Space Flight Center. FDD maintains over 100 software systems totaling about 4.5 millions lines of code [11].	Perfective changes
No	Analyzed 654 change and maintenance requests from a large software application (written in SQL) [19]	Corrective changes
No	A survey carried out in financial organizations in Portugal. Data was collected from 20 project managers [15].	Adaptive changes
No	453 non-defect changes from an Ada system developed at the Software Engineering Laboratory (SEL) of the NASA Space Flight Center [18].	Perfective changes
No	Version control and maintenance records from a multi-million line real-time software system [7].	Corrective changes
No	An integrated system for automated surveillance, a reengineering project (Written in C++; version 3 is 41 KLOC) [16].	Perfective changes
No	Three software products, a subset of Linux consisting of 17 kernel modules and 6506 versions, and GCC consisting of 850 KLOC [8].	Corrective changes
Yes	Analyzed 169 change requests (covers any change in the requirements or assets from the time of requirement baseline) for 4 releases of a large telecom system. This system reuses components [12].	Perfective changes
No	Web-based java application, consisting of 239 classes and 127 JSP files. Analysis of fault reports [17].	Perfective changes
Yes	Analyzed 208 change requests (covers any change in the requirements) for three releases of a reusable framework [13].	Perfective changes

3 The StatoilHydro ASA Setting

StatoilHydro ASA is a Norwegian company, and is part of the Oil & Gas industry. It is represented in 40 countries, has a total of about 31,000 employees, and is headquartered in Europe.

The central IT-department of the company is responsible for developing and delivering software meant to give key business areas better flexibility in their operation. It is also responsible for the operation and support of IT-systems. This department consists of approximately 100 developers, located mainly in Norway. Since 2003, a central IT strategy of the O&S (Oil Sales, Trading and Supply) business area has been to explore the potential benefits of reusing software systematically. StatoilHydro ASA has developed a custom framework of reusable components based on J2EE - JEF (Java Enterprise Framework). The actual JEF framework consists of seven separate

components, which can be applied separately or together when developing applications. Table 2 shows the size and release date of the three JEF releases. This JEF framework is currently being reused in two applications at StatoilHydro ASA. In this study we investigated one of these applications, namely DCF (Digital Cargo Files), due to the available data set. DCF is mainly a document storage application: A “cargo file” is a container for working documents related to a deal or cargo, used by all parties in the O&S strategy plan at StatoilHydro ASA. DCF is meant to replace the current means of handling cargo files, which are physical folders containing printouts of documents pertaining to a particular cargo or deal. The DCF application also consists of seven components. Table 3 gives an overview of the size and release date of the three DCF releases.

Although they have different aims, JEF and DCF have certain similarities. These systems operate in the same business domain, were conducted by a fairly stable set of developer from the same IT-department, were built over nearly the same time period, and are of similar size. The maturity level is the same for JEF and DCF. Thus they provide us with a fairly controlled environment for looking at whether process and product considerations impact the change characterization of systems.

Table 2. The size and release date of the three JEF releases

Release 1: 14. June 2005	Release 2: 9. Sept. 2005	Release 3: 8. Nov. 2005
17 KSLOC	19 KSLOC	20 KSLOC

Table 3. The size and release date of the three DCF releases

Release 1: 1. Aug. 2005	Release 2: 14. Nov. 2005	Release 3: 8. May 2006
20 KSLOC	21 KSLOC	25 KSLOC

3.1 Software Change Data in StatoilHydro ASA

When a software change is detected during integration/system testing, a change request or trouble report is written (by test manager, developers etc.) and tracked in the Rational ClearQuest tool. Examples of software changes are:

- add, modify or delete functionalities
- address a possible future problem by improving the design
- adapt to changes from component interfaces
- bug fixing

The test managers assign the change requests and trouble reports to developers. The developers then access the source files in the version control system (Rational ClearCase) to make modifications. When implementing the changes the developers adhere to the following steps:

- (1) Check out the file(s) corresponding to the specific change request.
- (2) Implement the specific software change.
- (3) Check the file back in to Rational ClearCase.
- (4) While checking in the file, they input a change description, a thorough description of what changes were made and a time and date.

Rational ClearCase captures various information about source code changes and the ClearQuest also stores information about changes to requirements and other documents. We extracted the data for JEF and DCF from Rational ClearCase as described in Table 4, with a corresponding example.

Table 4. The data collected from Rational ClearCase

Data	Example
File id	8
System	JEF
Filename	DataAccessException
Number of versions	2
Dates ²	Version 1: 19.04.2005, Version 2: 04.01.2007
Physical size (kilobytes)	1800
Size of a files first version	Non-commented SLOC (source lines of code): 34 Commented SLOC: 58
Size of a files last version	Non-commented SLOC: 34 Commented SLOC: 51
Descriptions of what changes occurred in each file version	Version 1: Component support for accessing data. Version 2: Remove obsolete java source file header entries.
Component to which the file belongs	One of the seven JEF or DCF components

4 Research Questions

The existence of comparable systems in the StatoilHydro ASA environment gave us the ability to examine our major research goal: *The impact of reuse*:

- The reusable framework (JEF) had changes related to all kinds of potential downstream reuse.
- The non-reusable application: DCF had only software changes related to the specific goals of that application (explained in section 3). The DCF application has different development characteristics for release 1 and release 2 and 3:
 - DCF1.0 is relatively unstructured, since it was unclear what the developers were supposed to implement, and how it should be organized. In the beginning the developers did not have a detailed design, and a lot of changes were made regarding functionality and design during the implementation and testing period.
 - DCF 2.0 and 3.0 were based on refactoring. Prior to DCF2.0, when the design and the goals became clearer the developers realized that the code they had developed was complex and hard to maintain. Therefore, they decided to do refactoring to improve the structure and ease the code maintenance.

The research questions we addressed for our goal are:

RQ1: Does the distribution of change types vary for different development characteristics? We hypothesize that the development process being employed would have a

² The date here refers to when the file was checked in after undergoing a change by the developer.

measurable impact on the type and number of changes required to a system. Making a software reusable may help to reduce certain kinds of changes, but may increase the frequency of other kinds of changes. For example, components that need to be reusable may have more adaptive changes, over a longer period of time, as more environments are found that could exploit such components. Since DCF went through a refactoring we also expect the preventive changes to decrease for release 2 and 3, compared to release 1. We have the following related questions:

- *RQ1.1: Does JEF have higher adaptive changes than DCF?*
- *RQ1.2: Is there a decrease in the preventive changes before and after refactoring for DCF?*
- *RQ1.3 Do perfective and corrective changes account for the majority of the changes, with adaptive following closely?*

RQ2: What change types are the longest active for different development characteristics? Our purpose is to investigate what change types (perfective, preventive, corrective and adaptive) are longest active for different systems, which may provide some insight into which types of changes are more difficult or problematic to make. It is important to clarify that the changes that are longest active do not necessarily require more effort; a change may not have been constantly under work the entire time it was open. However, if characteristic patterns are found, this can be useful as the starting point for a conversation with the developers to explore differences. The following are the related research questions for RQ2:

- *RQ2.1: Are adaptive changes longer active in JEF than DCF?*
- *RQ2.2: Are preventive changes longer active before refactoring than after for DCF?*

RQ3: How localized are the effects of different types of changes, for different development characteristics? We hypothesize that a change that needs to modify many files is not well-supported by the architecture, and hence more expensive to fix. Our purpose is to investigate whether development changes can be successful in reducing this metric for a system, and allowing future changes to be more localized. We would like to investigate the following research questions for RQ3:

- *RQ3.1: Is the average number of files modified for adaptive changes higher in JEF than DCF?*
- *RQ3.2: Is the average number of files modified for preventive changes higher before refactoring than after for DCF?*

5 Research Methodology

We analyzed a representative sample of the software changes for both the JEF framework and the DCF application to answer the research questions *RQ1-RQ3*.

Our analysis began from the files checked into Rational ClearCase. In total over all releases, there were 481 files for JEF framework and 1365 for the DCF application, distributed across the seven components in each system. Due to the manual nature of the analysis it was infeasible to analyse all changes to all 1846 files. Therefore we adopted a strategy of analysing a representative subset of the files in each component.

In our data collection we decided to have a threshold value of 10 files. This means that if a component had more than 10 files we would not include all of the files in our dataset, but pick a random subset that was representative of the properties of the largest. A sampling calculator [26] was used to calculate a sufficient sample size. For example component JEFClient had 195 files. Based on the calculated sample size (165), we randomly (using a mathematic function in excel) selected 165 files from the JEFClient to include in the dataset.

In total we used 442 files for the JEF framework and 932 files for the DCF application. Table 5 gives an overview of the actual number of files in Rational ClearCase vs. the number of files we analyzed, and the size (in SLOC, including the non-commented source lines of code) for the collected files.³ In total we analyzed 1105 changes for the JEF framework and 4650 changes for the DCF application. We can see that the number of changes for DCF is higher than for JEF. This can be explained by that DCF development was going on for about 10 months (Table 3), while JEF development was going on for about 6 months (Table 2). Due to longer development period, DCF faced more changes.

Table 5. Description of data set collected from ClearCase

	Actual number of files	Number of files collected	Number of changes collected	Size in SLOC for files collected
DCF: Release 1 (before refactoring)	426	282	2111	15K
DCF: Release 2 and 3 (after refactoring)	939	650	2539	55K
JEF framework	481	442	1105	38K
Total	1846	1374	5755	108K

During the classification and comparison, we noticed that some of the changes descriptions were labelled as “no changes” (meaning no changes were made to the code), and “initial review” (meaning changes resulting from formal code review of the code). The changes in category “code review” are changes we cannot classify, since no description of the change was provided. We grouped “no changes” into the category “other” and “initial review” into the category “code review”. The changes in the category “other” and “code review” are excluded from the analysis for RQ1 – RQ3. Quantitative differences among the change profiles of the systems were used to formulate questions for the development team. These questions were addressed via interviews which elicited possible explanations for these differences.

6 Results

Before investigating our specific research questions, we examined the distribution of data across the change history. The test for normality of our datasets failed, meaning that the data is not normally distributed. Additionally, we investigated the variances

³ However, the SLOC is just for the last version of the collected files. For example, if a file has 6 versions, the SLOC is presented for version 6 only and not for the remaining files. Thus these values should be taken as only an approximate overview of file sizes.

for each change type for JEF and DCF and they turned out to be quite large (e.g. 3555 for DCF and 11937 for JEF for perfective changes) respectively. Hence, we decided not to use T-tests to statistically test our hypotheses, and present the results with histograms. The following is a summary and possible explanation of the results from our analysis of software changes for JEF and DCF.

RQ1: Does the distribution of change types vary for different development characteristics? We plotted our data in a histogram, shown in Fig. 1. From Fig. 1-a) we observed the following for JEF:

- 1) Decreasing perfective, corrective, preventive and adaptive changes over the three releases. The sudden drop in number of perfective changes for JEF between release 1 and release 2 and 3, yields that release 2 and 3 did not have much requirement changes and was based more on third party components. We can also see that there is not a big difference in the number of changes between release 2 and 3.
- 2) The preventive and adaptive changes decrease towards 0 between release 2 and release 3.
- 3) For the 3rd release the dominating changes are perfective and corrective, but the perfective changes are the most frequent ones.

For DCF (Fig. 1-b) we observed that:

- 1) Although the number of changes goes down for DCF between release 1 and 2 (before and after refactoring) for all change types, there is not a tendency that shows that any of these change types are decreasing.
- 2) It seems that corrective changes remain in the 25% of the changes.

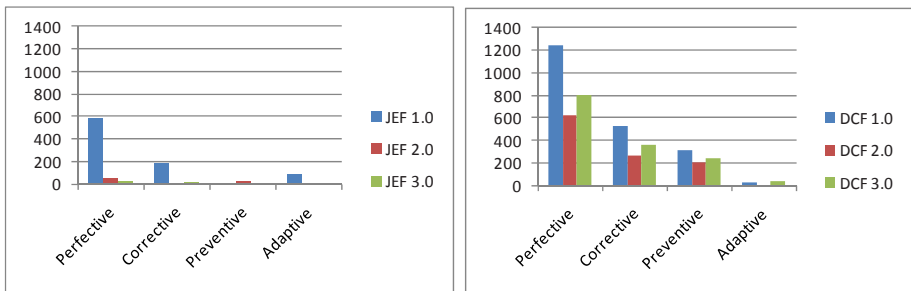


Fig. 1. Number of Changes: a) JEF, b) DCF

Fig. 1 shows that *perfective and corrective changes* account for the majority of changes, for both the reusable JEF framework and the non-reusable DCF application. Our results confirm some of the findings from earlier studies (see Table 1), which shows that perfective and corrective changes are the most frequent ones independent of which kind of development characteristics the applications have. However, for JEF compared to DCF the adaptive changes follow closely. Regarding the perfective changes a contributing factor on DCF was an incomplete and poorly documented design, which required a high number of improvements over time. Important factor for JEF was to develop a common framework to support GUI (Graphical User

Interface) development “up front” (developing without knowing all the functionalities a framework may need). The *least frequent changes* for the non-reusable application are the *adaptive changes*, and for the reusable framework the *least frequent changes* are the *preventive changes*. Contributing factors for the preventive and adaptive changes for DCF were:

- *Preventive changes*: Time pressure and incomplete and poorly documented design lead to some refactoring, since everything was not implemented optimally the first time. However, we can see a decrease in the preventive changes before (release 1) and after (release 2) refactoring.
- *Adaptive changes*: Minor changes were made to the environment/platform, which explains the small amount of *adaptive changes*.

Contributing factors for the preventive and adaptive changes for JEF were:

- *Preventive changes*: JEF did not go through the same time pressure as DCF during development. That resulted in a higher code quality for JEF, and less need for refactoring.
- *Adaptive changes*: StatoilHydro ASA changed their version control system from PVCS to Rational ClearCase in the middle of the project. All the files in the PVCS had a java comment, but when StatoilHydro ASA switched to Rational ClearCase the java comments in all the files were removed. The reason for why these changes are seen as adaptive changes is due to that these files had to be adapted to a different version control system (see section 2 for definition of adaptive changes). The higher frequency (compared to DCF) of adaptive changes can also be explained by the fact that JEF is built over various third party components, and changes in these components will cause changes in the framework.

We can see from Fig. 1 that JEF has a higher amount of adaptive changes than DCF. For JEF we see that adaptive changes accounted for more than usual compared to DCF, but still a fairly low number. This might be some surprising given that we expected JEF to need to be reused in a number of different environments/applications. However, this can partially be explained by the fact that the data we collected from Rational ClearCase includes just one application reusing the JEF framework. There are other application reusing JEF but they are for the time being under development and no data is available.

Answering our research questions:

- *RQ1.1: Does JEF have higher adaptive changes than DCF?* Yes, JEF (total number of changes 94) has higher adaptive changes than DCF (total number of changes 58).
- *RQ1.2: Is there a decrease in the preventive changes before and after refactoring for DCF?* Yes, there is a decrease in the preventive changes before (total number of changes 306) and after (203 for release 2 and 240 for release 3) refactoring for DCF. We can see there is a slightly increase between release 2 and 3 (18%), but still the number of changes are less for release 3 compared to before refactoring.
- *RQ1.3: Do perfective and corrective changes account for the majority of the changes, with adaptive following closely?* Yes, perfective and corrective

changes account for the majority of changes for JEF and DCF, but it is only for JEF that adaptive changes follow closely.

RQ2: What change types are the longest active for different development characteristics? From Fig. 1-a) we saw there was not a big difference in the number of changes between release 2 and 3. Therefore, we decided not to divide the JEF framework into three releases for our analysis of RQ2, since it will not affect the average. This means that for RQ2 we will here compare DCF release 1, 2 and 3 against the whole JEF framework.

By comparing the change types that are longest active for JEF and DCF, we found from Fig. 2-a) that *adaptive* (average of 50,2days) changes are longest active for JEF. This is because StatoilHydro ASA changed their version control system from PVCS to Rational ClearCase in the middle of the project. All the files in the PVCS had a java comment related to this version control system, but when StatoilHydro ASA switched to Rational ClearCase the java comments in all the files were removed. The JEF framework is built over various third party components, and changes in these components will cause changes in the framework. However, we can speculate that adaptive changes were longest active for JEF, because they affected many files. Another reason could be that adaptive changes were given low priority to fix. Thus, these files may have been checked out while developers might have been busy with other tasks with higher priority.

From Fig. 2-b) we can see that *preventive* changes (average of 17,0 days) are longest active for DCF, and the number of days for preventive changes drops (84% in average) between the two first releases of DCF. This is because before refactoring the code was difficult and hard to maintain (release 1), but after the refactoring the code became easier to maintain (release 2).

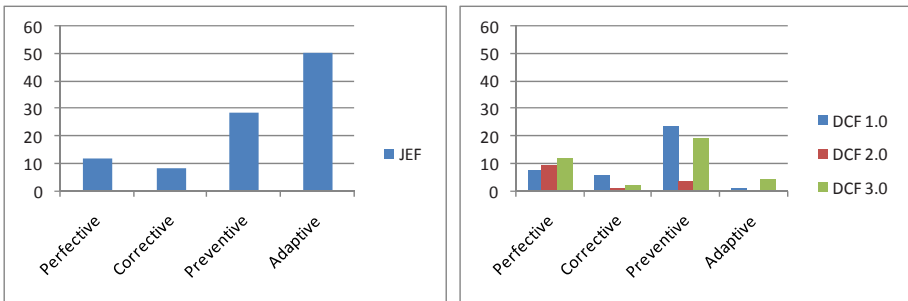


Fig. 2. Average #of days the changes are active: a) JEF, b) DCF

It is important to clarify that the changes that are longest active do not mean that they require more effort, since we do not have the effort data. However, by looking into what change types are active longest we might to some extent be able to say if these changes stays longer in the applications and require more time to fix.

Answering our research questions:

- *RQ2.1: Are adaptive changes longer active in JEF than DCF?* Yes, adaptive changes are longer active for JEF (average of 50,2 days) than DCF (average of 2,5 days).

- *RQ2.2: Are preventive changes longer active before refactoring than after for DCF?* Yes, preventive changes are longer active before refactoring; release 1 has an average of 23, 5 days. While after refactoring; release 2 has an average of 3,8 days, and release 3 has an average of 19, 3 days. We can see there is an increase between release 2 and 3 (80% in average), but still the average number of days are less for release 3 compared to before refactoring.

RQ3: How localized are the effects of different types of changes, for different development characteristics? For RQ3 we will also compare DCF release 1, 2 and 3 against the whole JEF framework. By comparing the average number of files changed for each change type (Fig. 3), we found that DCF has higher average amount of files modified for the preventive changes (14,5). From Fig. 3-a) we can see that JEF has higher amount of files changed for the adaptive changes (5,5).

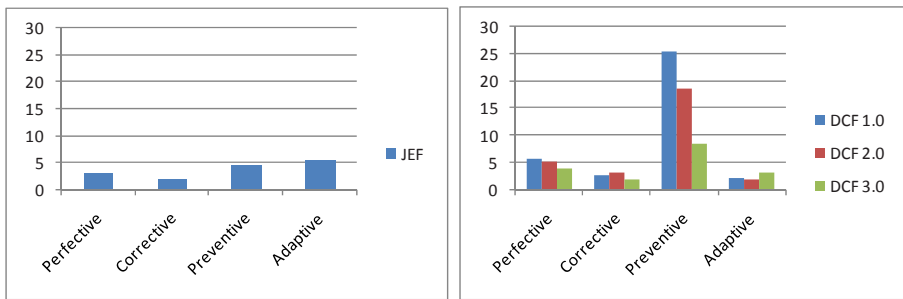


Fig. 3. Average amount of files modified: a) JEF, b) DCF

From Fig. 3-b) we can also see the affect of the refactoring that happened between all the three releases, since the average number of files modified decreases. This decrease in the files for the preventive changes is related to adapting to an open source system framework to improve and ease the code related to handling GUI events. Before refactoring most of the code was developed by the developers and just some parts of the open source system framework were used. This made the code more complex, and difficult to maintain. Due to the high time-pressure the code was developed quickly and was defect-prone. However, during the refactoring the developers adapted more of the open source system framework and the code became much more structured.

Answering our research questions:

- *RQ3.1: Is the average number of files modified for adaptive changes higher in JEF than DCF?* Yes, the average number of files modified for adaptive changes is higher for JEF (5,5 files modified) than DCF (2,4 files modified).
- *RQ3.2: Is the average number of files modified for preventive changes higher before refactoring than after for DCF?* Yes, DCF (before refactoring) has in average 25,5 modified files. While DCF (after refactoring) has in average 18,5 modified files (release 2), and 8,4 modified files (release 3).

RQ2 combined with RQ3, we see the following results for DCF:

- Even though the average number of days the changes are active are high for perfective and preventive changes, the number of files modified (within these two change types) are getting less over the three releases.

7 Threats to Validity

We here discuss possible threats to validity in our case study and the steps we took to guard against them, using the definitions provided by [27]:

Construct Validity: All our data is from the pre- and post-delivery software changes (covering any reported changes) for the three releases of the reusable framework, and for the three releases of the DCF application.

External Validity: The object of study is a framework consisting of only seven components, and only one application. The whole data set of software changes in StatoilHydro ASA has been collected for three releases of the reusable framework, as well as for three releases of the application. So, our results should be relevant and valid for other releases of the framework and other future releases of the application. The entire data set is taken from one company. Application of these results to other environments needs to be considered on a case by case basis, considering factors such as:

- *The nature of the software development:* The DCF application and the JEF framework in our study are based on the object-oriented programming language, namely Java. Additionally, DCF is based on a waterfall process while JEF is based on a combined incremental/waterfall process.
- *The profile of the company and projects:* The profile of the company is an oil and gas company, and hence the projects are related to oil and gas field.
- *The way that software changes are defined and classified:* Our definition of software changes and other definitions used (see section 2), vary among the different studies.
- *The way that software changes are collected and measured:* We have collected software changes related only to the non-commented source lines of code for a reusable framework and a non-reusable application.

Internal Validity: All of the software changes for JEF and DCF were classified manually. Two researchers classified independently all the changes, and then cross-validated the results. This is to enhance the validity of the data collection process. A threat to the internal validity is the number of files we have selected from Rational ClearCase. However, we have 422 files for the JEF framework and 932 files for the DCF application, which should be enough files to draw a valid conclusion. We did a semi-random sampling to ensure the normal distribution between components.

Conclusion Validity: We verified the reasons for differences of software change profiles between the JEF and DCF by interviewing one senior developer (see section 5). Just asking one developer might cause systematic bias. However, we do not consider this possibility to be a threat for our investigation, because the senior developer has worked with both the JEF framework and the DCF application. His insights further supported our results for RQ1-RQ3.

8 Conclusion and Future Work

Few published empirical studies characterize and compare the software changes for a reusable framework with those of a non-reusable application. We have presented the results from a case study for a reusable class framework and one application reusing it in StatoilHydro ASA. We studied the impact that software changes had on different development characteristics (e.g. impact of reuse and impact of refactoring). Our results support previous findings to the effect that perfective and corrective changes accounts for the majority of changes in both reusable and non-reusable software, but it is only for the reusable framework that adaptive changes follow closely. We also observed that DCF faced higher time-to-market pressure, more unstable requirements, and less quality control than the reusable framework.

When it comes to *designing for reuse* it does have an effect on the aspect of the change types. Our results indicate that adaptive changes have longer active time and files related to adaptive changes are more modified in JEF compared to DCF. The increase in adaptive change might be a result of successfully shielding the end user (i.e. DCF developer) from changes from the vendors. Additionally, preventive changes are more common in DCF (due to the refactoring that took place). So, the amount of changes, as well as the effect on the localization of changes will not be similar to the systems not necessarily designed for reuse.

Non-reusable applications usually face more unstable requirements, higher time-to-market pressure, and less quality control than the reusable framework. Therefore, their poorer quality is not surprising. So, making a component reusable will not automatically lead to better code quality. In order to lower the amount of software changes of the reusable component, it is important to define and implement a systematic reuse policy; such as better design [28] and better change management [21].

In addition, we have seen a positive affect for the *refactoring*. A system with poor structure initially has to deal with more frequent preventive changes before refactoring than after. However, our results indicated that there was an increase in preventive changes between release 2 and 3 (after refactoring), but the increase in release 3 was still less than before refactoring.

The lesson learned here is that developing a framework “up front” (developing without knowing all the functionalities a framework may need) is always difficult and challenging, since you do not know all of the requirements that will appear when a reusable framework is being used.

One interesting question raised by StatoilHydro ASA is whether the results of our study could be used as input to improve future reuse initiatives. In addition, we intend (i) to expand our dataset to include future releases of the JEF framework, future releases of the DCF application, and new applications (further reuse of the JEF framework), and (ii) to refine our research questions on the basis of the initial findings presented herein.

Acknowledgement

This work was financed by the Norwegian Research Council for the SEVO project [29] with contract number 159916/V30. We thank StatoilHydro ASA for involving us

in their reuse projects. This work is also partially sponsored by NSF grant CCF0438933, "Flexible High Quality Design for Software."

References

1. Bennett, K.H., et al.: Software Maintenance and Evolution: A Roadmap. In: 22nd Intl. Conf. on Software Engineering, pp. 73–78. IEEE Press, Limerick (2000)
2. Lehman, M.M., et al.: Programs, Life Cycles and Laws of Software Evolution. In: Proc. Special Issue Software Eng., vol. 68(9), pp. 1060–1076. IEEE CS Press, Los Alamitos (1980)
3. Postema, M., et al.: Including Practical Software Evolution in Software Engineering Education. IEEE Computer Society Press, Los Alamitos (2001)
4. Sommerville, I.: Software Engineering. Sixth Edition. Addison-Wesley, Reading (2001)
5. Bennet, P.L.: Software Maintenance Management: A Study of the Maintenance of Computer Application Software in 487 Data Processing Organizations. Addison-Wesley Pub., Reading (1980)
6. Lientz, B.P., et al.: Characteristics of Application Software Maintenance. Communications of the ACM 21(6), 466–471 (1978)
7. Mockus, A., et al.: Identifying Reasons for Software Changes Using Historical Database. In: Proc. IEEE Intl. Conf. on Software Maintenance, pp. 120–130. IEEE CS Press, San Jose (2000)
8. Schach, S.R., et al.: Determining the Distribution of Maintenance Categories: Survey versus Management. Empirical Software Engineering 8, 351–366 (2003)
9. Abran, A., et al.: Analysis of Maintenance Work Categories Through Measurement. In: Proc. Conf on Software Maintenance, pp. 104–113. IEEE CS Press, Sorrento (1991)
10. Yip, S., et al.: A Software Maintenance Survey. In: Proc. 1st Int. Asia- Pacific Software Engineering Conference, Tokyo, pp. 70–79 (1994)
11. Basili, V., et al.: Understanding and Predicting the Process of Software Maintenance Releases. In: 18th Intl. Conf. on Software Engineering, pp. 464–474. IEEE CS Press, Berlin (1996)
12. Mohagheghi, P.: An Empirical Study of Software Change: Origin, Impact, and Functional vs. Non-Functional Requirements. In: Proc. at Intl. Symposium on Empirical Software Engineering, pp. 7–16. IEEE CS Press, Los Angeles (2004)
13. Gupta, A., et al.: An Empirical Study of Software Changes in Statoil ASA – Origin, Priority Level and Relation to Component Size. In: Intl. Conf. on Software Engineering Advances, p. 10. IEEE CS Press, Tahiti (2006)
14. Jørgensen, M.: The Quality of Questionnaire Based Software Maintenance Studies. ACM SIGSOFT – Software Engineering Notes 20(1), 71–73 (1995)
15. Sousa, M., et al.: A Survey on the Software Maintenance Process. In: Intl. Conf. on Software Maintenance, pp. 265–274. IEEE CS Press, Bethesda (1998)
16. Satpathy, M., et al.: Maintenance of Object Oriented Systems through Re-engineering: A Case Study. In: Proceedings of the 10th Intl. Conf. on Software Maintenance, pp. 540–549. IEEE CS Press, Montreal (2002)
17. Lee, M.G., et al.: An Empirical Study of Software Maintenance of a Web-based Java Application. In: Proceedings of the IEEE Intl. Conf. on Software Maintenance, pp. 571–576. IEEE CS Press, Budapest (2005)
18. Evanco, M.: Analyzing Change Effort in Software During Development. In: Proc. 6th Intl. Symposium on Software Metric, Boca Raton, pp. 179–188 (1999)

19. Burch, E., et al.: Modeling Software Maintenance Requests: A Case Study. In: Proceedings of the Intl. Conf. on Software Maintenance, pp. 40–47. IEEE CS Press, Bari (1997)
20. Mohagheghi, P.: An Empirical Study of Software Reuse vs. Defect Density and Stability. In: Proc. 26th Intl. Conf. on Software Engineering, pp. 282–292. IEEE-CS press, Edinburgh (2004)
21. Selby, W.: Enabling Reuse-Based Software Development of Large-Scale Systems. *IEEE Transactions on Software Engineering* 31(6), 495–510 (1995)
22. Gupta, A., et al.: A Case Study of Defect-Density and Change-Density and their Progress over Time. In: 11th European Conf. on Software Maintenance and Reengineering, pp. 7–16. IEEE Computer Society, Amsterdam (2007)
23. Zhang, W., et al.: Reuse without compromising performance: industrial experience from RPG software product line for mobile devices. In: Proc. 9th Intl. Software Product Line Conference, pp. 57–69. Springer, Rennes (2005)
24. Frakes, W.B., et al.: An industrial study of reuse, quality, and productivity. *Journal of System and Software* 57(2), 99–106 (2001)
25. Algestam, H., et al.: Using Components to Increase Maintainability in a Large Telecommunication System. In: Proc 9th Int. Asia- Pacific Software Engineering Conference, pp. 65–73 (2002)
26. Sampling calculator , http://www.macorr.com/ss_calculator.htm
27. Wohlin, C.: *Experimentation in Software Engineering – An Introduction*. Kluwer Academic Publishers, Dordrecht (2002)
28. Succi, G., et al.: Analysis of the Effects of Software Reuse on Customer Satisfaction in an RPG Environment. *IEEE Transactions on Software Engineering* 27(5), 473–479 (2001)
29. Sevo project, <http://www.idi.ntnu.no/grupper/su/sevo/>

Virtual Prototypes in Developing Mobile Software Applications and Devices

Kari Liukkunen¹, Matti Eteläperä², Markku Oivo¹, Juha-Pekka Soininen²,
and Mika Pellikka³

¹ Department of Information Processing Science, University of Oulu
P.O. Box 300, 90014 Oulu, Finland

² VTT, Kaitoväylä 1, 90570, Oulu, Finland

³ CCC Group, Tietotie 2, 90460 Oulunsalo, Finland

kari.liukkunen@oulu.fi, matti.etelapera@vtt.fi,
markku.oivo@oulu.fi, juha-pekka.soininen@vtt.fi,
mika.pellikka@ccc.fi

Abstract. The goal of this paper is to study how software based virtual prototypes and hardware simulation tools can be combined. By combining these tools and techniques we can shorten the time to market with parallel concurrent design and more importantly, we can provide a real-time simulation environment for virtual prototypes. Application designers get access to a simulated realistic real-time mobile device well before the first prototypes are available from the device manufacturer. The research work was done in two cases. In the first case the virtual prototypes were used to illustrate and help to select new mobile application concepts and to test new applications usability. In the second case the virtual prototypes were used for modelling the product platforms, e.g. the computer system and the simulation of the complete system including both hardware and software. Our approach facilitates early simulation and testing of the final user experience and system behaviour in cases where they are heavily dependent on the characteristics and performance of the underlying computer platform.

Keywords: Usability requirements, user interfaces, user experiences, virtual prototypes, mobile software applications, virtual platform.

1 Introduction

In order to meet the usability requirements in the development of interactive software for handheld devices, such as mobile phone applications, the user-centred design approach is an appealing approach. There are tools that enable the virtual simulation of applications for handheld devices (terminals). These tools offer a virtual design space that is used to design the application and produce the virtual prototypes that can be used to simulate the application logic and user interface. There are also tools that allow the simulation of the target hardware platforms for the terminals for example for building software platforms for mobile devices. These tools are normally used in

isolation by different designer groups. However, our goal was to integrate these tools and their use for early and realistic simulation purposes. Rapid capacity increase of simulator workstations and the simulation tools have made it possible for the first time.

Paper prototyping or story boarding enables the rapid concept designing of certain parts of software's user interface, but with virtual prototyping and simulation tools it is possible to produce fully interactive simulations. Virtual prototypes enable the rapid concept design of software applications already before the implementation work has started. Virtual prototypes can therefore be used to pilot software application concepts and support the concept selection process. When usability and user experience are considered the real performance provided by the execution platform needs to be taken into account. Otherwise, the usability results are based on false hopes of ideal machines. Then the specification can lead to enormous implementation problems, where the software developers are facing an impossible mission. In the worst case, the result may be the change of specification and redesign.

Performance modeling and simulation is currently used approach to analyse the embedded or mobile system, but it has also its limitations. Real-time (or almost real-time) performance is needed in order to provide realistic responses to the user for usability studies. However, the final performance depends very often on the software and hardware architecture design decisions on a very detailed level. The final performance bottlenecks can be results from example poorly designed search algorithms, inadequate bus or memory interfaces or unfortunate interactions of different system functions. Modelling the system on such detailed level is too much time consuming, too expensive and typically results too slow simulation performance to be practical.

In case both hardware and software are new, the system architect and designers can optimise the final system instead of software or hardware. From product development point of view the benefits are naturally related to better product, but only if we can do the software and hardware development in parallel and if we can look the complete system characteristics during the design (Fig. 1).

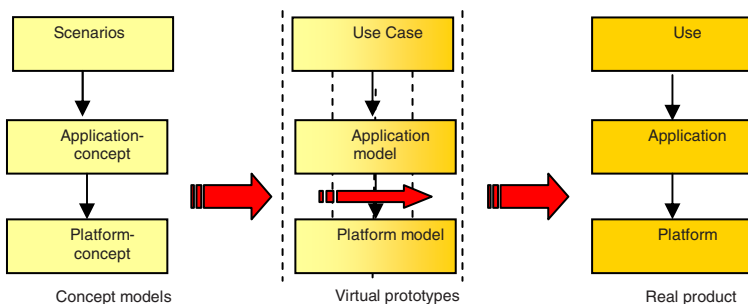


Fig. 1. Virtual prototypes in application and platform modeling

In embedded system domain, the development of hardware and software has been sequential. Typically embedded software has been developed for some existing embedded hardware. Two factors are changing this. First, the increase pressure to

shorten the design time, which also means that the hardware has to renew more rapidly. Secondly, the hardware is becoming more flexible due to the reconfigurability. In practice the functionality of the most of the embedded systems today could be implemented using SoPC (system on programmable chip) devices where it is possible to change the hardware system also on run-time.

Complexity of underlying hardware has also increased and it is making the management of performance of complete system extremely difficult. For example, if we look into a modern mobile terminal platform such as OMAP2 architecture, we see a complex computer with general purpose processor ARM11 running Symbian or Linux real-time operating systems, a very complex DSP processor, programmable accelerators for video and graphics, few buses, a complex memory organisations and an interconnect that tries to feed up all units with data. So, it is complex embedded computer with a complex set of software services on the top of it. And it is targeted for a battery operated device. The next generation platforms will be even more complex with packet-switched networks and multiple processors.

With battery operated devices and current system complexities it is clear that traditional general purpose computer design ideas needs to be rethought. Hardware architecture design must be guided by the intended use cases and applications. The system resources must be optimised for them in order to avoid the waste of time, energy and money. Similarly in the software architecture and software design, the decisions must be based on the knowledge on what actually happens in the system, how system resources are used, and how the system would respond to the user.

The approach taken in this is to combine software based virtual prototypes and hardware simulation tools. By combining these tools and techniques we can shorten the time to market with parallel design of usability, software system and underlying computer hardware as shown in Fig. 2. Even more importantly, we can provide a real-time simulation environment for virtual prototypes containing both models of applications and execution platforms. Then the application designers can get realistic environment were they can design and test not only the logic of the UI, but also the timing, delays and many other crucial time related issues in UI design. Application designers get access to a simulated realistic real-time mobile device well before the first prototypes are available from the device manufacturer. The first models can be very abstract conceptual models that are then refined during the design until the real

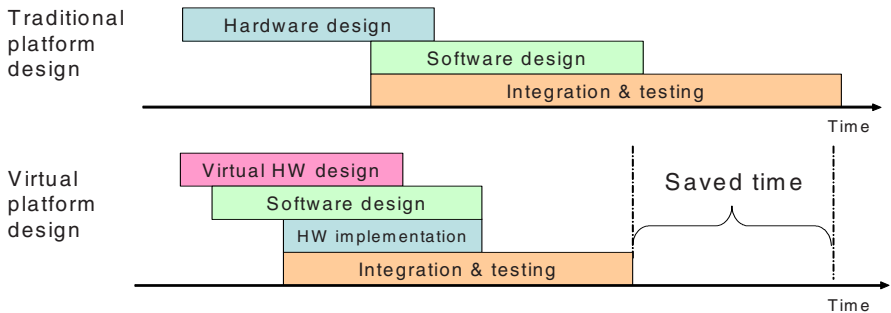


Fig. 2. Virtual versus traditional platform design. Simultaneous hardware and software design is one of the benefits of virtual platform based design

product is ready as shown in Fig. 2. Both the application software developers and execution platform developers can have very rapid feedback on how the changes made in either area affect to the complete system and its usability characteristics.

Simulation tools used in this research were CCC Group's Cybelius Maestro, which is suitable for simulating application's user interface and CoWare's Electronic System Level design tool set, especially SystemC simulation tool, suitable for simulating target platforms. These tools were used in different parts of the development process, starting from collecting ideas and ending to software testing using the virtual hardware platform. In this project, it was impossible to have only one case and follow it through the development process. We have to use three different cases. In this paper we later describe these cases and our experiences.

2 Related Work

Modern computer software is characterized by continuous change, by very tight time-lines, and by an emphatic need for customer/user satisfaction. In many cases, time-to-market is the most important management requirement. If a market window is missed, the software project itself may be meaningless. Virtual design has the potential to help in these critical problem areas. Tseng et al. [18] explain that virtual design is proposed to replace hardware prototypes with computational (virtual) prototypes of systems and the processes that they may undergo. By replacing hardware with computational prototypes, the potential is tremendous for greatly reducing product development time, manufacturing facility ramp-up time, and product development cost.

2.1 Virtual Prototyping

Virtual prototyping can be viewed either as a technology term or as a process description. The process description of virtual prototyping is as follows [13]:

“Virtual prototyping is a process in which a product or a product concept, its behavior and usage are simulated as realistically as possible by combining different prototyping and simulation models and techniques.”

When virtual prototyping is viewed as technology term, the focus is usually on the virtual prototype and its realization. Based on the definition given by Haug et al. [7] a virtual prototype could be described as follows:

“A virtual prototype is a simulation of a target object which aims at an adequate degree of realism that is comparable with the physical and logical functionality and the appearance of the possible real object, and that is achieved by combining different prototyping and simulation models and techniques.”

According to Kiljander [14] the UI prototyping methods include Mathematical models, Scenarios, 2D & 3D drawings and computer aided design (CAD) models, Storyboards, Paper prototypes, Computer simulations, Virtual reality prototypes, Hard models, and Hardware prototypes. Ulrich & Eppinger [19] have classified prototypes along two dimensions – physical/analytical and comprehensive/focused. Kiljander [14] has ordered the different UI prototyping methods to this classification according to the methods' level of fidelity i.e. interactivity or concreteness they support (Fig. 3).

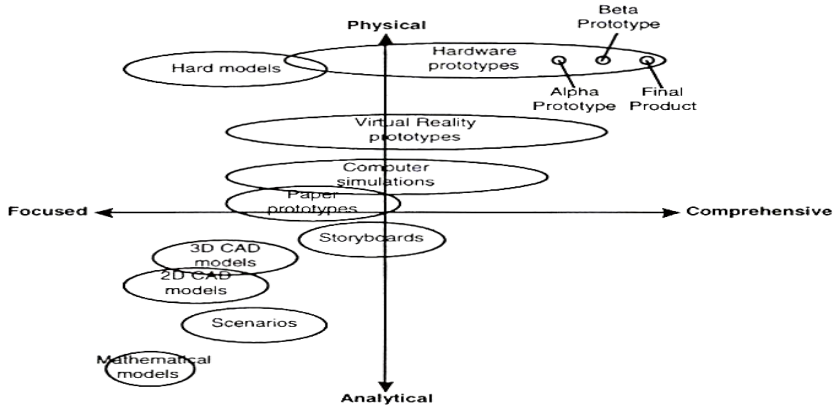


Fig. 3. Classification of Prototypes with UI Prototyping Methods [14]

2.2 System Simulation

There are three basic approaches, if the characteristics of executing hardware are taken into account during the development of software applications [11]. First, we can measure the effects of hardware if the hardware exists. This is naturally very accurate method. The accuracy and the reliability of results depend on the maturity of the software we are developing. The second approach is to use some hardware prototype or hardware emulator that mimics the final hardware. The accuracy is degraded, because the used hardware is not exactly as the final hardware and it introduces errors. The third alternative is to model the hardware and simulate its behavior in a computer. The simulation model is called a virtual platform and if we simulate the execution of application software or even the complete software system, the complete system model is a virtual prototype of a product.

The traditional technique for the analysis of software-hardware system is co-simulation. The first co-simulation environments were commercialized during early 1990s and because the need for co-simulation was initiated from hardware design the simulation models were very detailed and slow. In the co-simulation the software can be executed using real processors, emulated processors, RTL-simulation models of the processors, instruction-set simulators of the processors [8, 1], or more abstract functional simulation models [5]. Similar abstraction levels are available for the other hardware parts and recently the development has been from detailed hardware descriptions towards more abstract transactional models. In the processor-hardware interface bit-accurate models, bus-accurate functional models and abstract communication channels have been used [2].

When co-simulation is used for system-level design, the problem is the trade-off between modeling effort, accuracy and performance. Complex software systems cannot be simulated using a clock-accurate processor or bus models. There are three basic approaches for solving the performance problems. The first is to increase the abstraction level of the hardware simulation by, for example, using concurrent processes that communicate using channels [4, 6]. The second is to use host-based execution of the software instead of trying to execute machine code instructions [20]. The

third is to replace the final code with a more abstract workload model that only generates events for the hardware architecture instead of implementing the complete functionality of the software [15]. The state-of-the-art tools currently are based on the use of high-level modeling and simulation languages such as SystemC [21]. SystemC is based on C++ and shares the same syntax and advanced features such as dynamic memory allocation and object oriented design partitioning. It is a very flexible language and it is possible to build custom platform simulators of different abstraction levels using it [22].

3 Research Approach

In this research we have carried out three cases to study how virtual prototypes can be used for piloting and testing of mobile software applications, how these virtual prototypes can support usability planning and collecting the user experiences of a Web-based product and how virtual platform can be used to test software applications (Fig. 4). Cases included iterative construction of virtual prototype solutions for two real-world applications (City of Oulu and Ouman). In these two cases UI-simulation was used to collect mobile application ideas for the city of Oulu. Solution concepts were then created from these ideas. Concepts were built in the form of virtual prototypes. In the second case study the goal was to gather usability requirements of the Web application for a company called Ouman. The City of Oulu case was categorized as “Concept design process” and Ouman case as “Requirements gathering”. In the third (VTT) case research scope was to demonstrate the capabilities of virtual platform design methodology on software development. Full commercial implementation was left out of the scope of this research.

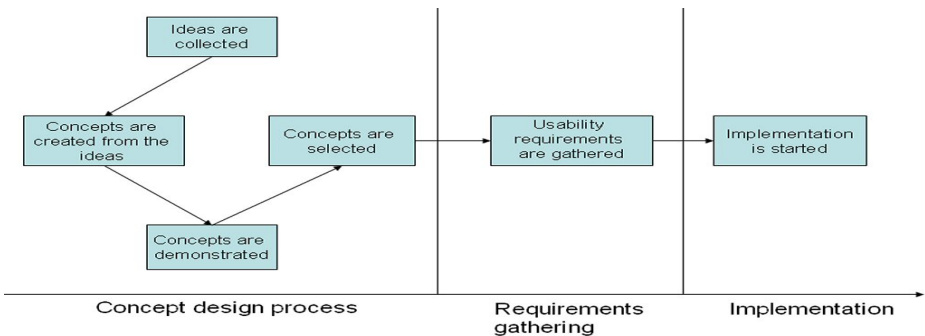


Fig. 4. Focus of the City of Oulu and Ouman research cases

3.1 UI Simulation Approach

Virtual design is becoming more important also because the functions of a product are implemented more and more through software. For example, in mobile phones the size of software has risen from some kilobytes in analogue phones to several megabytes in the latest smart phones. The weighting of software is shifting from lower

level system to the UI parts. In some consumer products UI software can account for over 50% of the entire software. [13, 14]. This shift can be better understood looking at it from a user's perspective. Mayhew points out that as far as users are concerned the UI is the product, because their entire experience with the product is their experience with its UI. [16].

The definition of usability from ISO 9241-11 is: "The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use." [10, 2]. Jokela [12] describes that the ISO 9241-11 definition is not only a formal standard, but is also becoming the de facto standard. He also points out that one essential feature concerning usability is that it is not an absolute product property, but it always exists in relation to the users of the product.

Virtual prototypes are created in a concept design process. It is possible to see the international standard ISO 13407 as a concept design process. It identifies five UCD activities (processes), four of which deal with the substance of UCD while one is about the planning of "human-centred design". The processes of UCD are illustrated in Fig. 5.

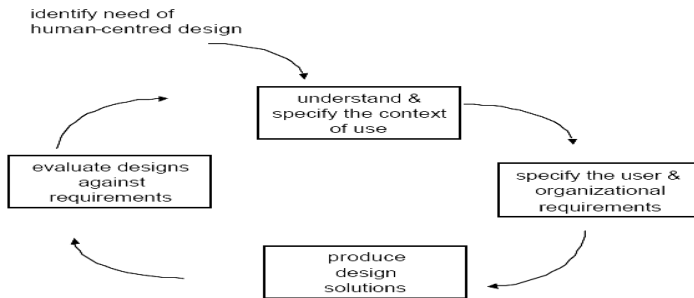


Fig. 5. Processes for user-centered design in ISO 13407 [9]

To be exact, concepts are created as a result of the "produce design solutions" process, but performing every process of the standard is essential for getting relevant results. Accurate product concepts would be hard to create without understanding the context and environment of use.

In this research the focus is on concept design of Web applications UIs and collecting usability data concerning the UIs before the real application actually exists. This can be achieved by illustrating the concepts with functional virtual prototypes, and by performing usability tests to users with these virtual prototypes. These activities are described in the ISO 13407 standard as "produce design solutions" and "evaluate designs against requirements" processes.

3.2 HW Simulation Approach

Our approach is based on the idea that application software is mapped on the platform model of computer system and the resulting system model is then simulated (Fig. 6.). The SW is written in C and compiled to target processor. The platform model is

created using Coware ESL tools and SystemC models. The model should be created detailed enough to provide performance data for usability analysis, but abstract enough to have adequate real-time performance for interactive use of applications. In the CoWare toolset a simulation engine is a SystemC simulator. In case we identify any performance or functionality problems then either the application code or platform hardware model can be changed or the whole system re-simulated.

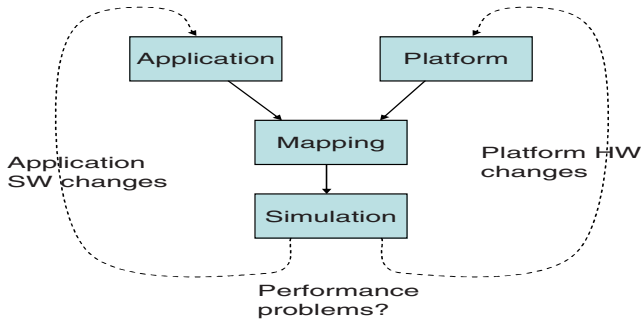


Fig. 6. Basic application-platform simulation approach

The resulting system model is actually a virtual prototype of a product. It consists of a HW model, i.e. the execution platform, the HW-dependent SW parts, e.g. operating system such as Symbian or Linux, control SW code and platform services, etc., and application code as shown in Fig. 7. Modeled execution platform and simulator offers full visibility to all parts of hardware so it is possible to monitor all parts of system. This is a superior feature to any implementation or emulator, since it allows the designer to see what happens inside a computer, which is typically impossible. The modern embedded systems computers are integrated components containing several processors, dedicated memory organizations and complex interconnect that are

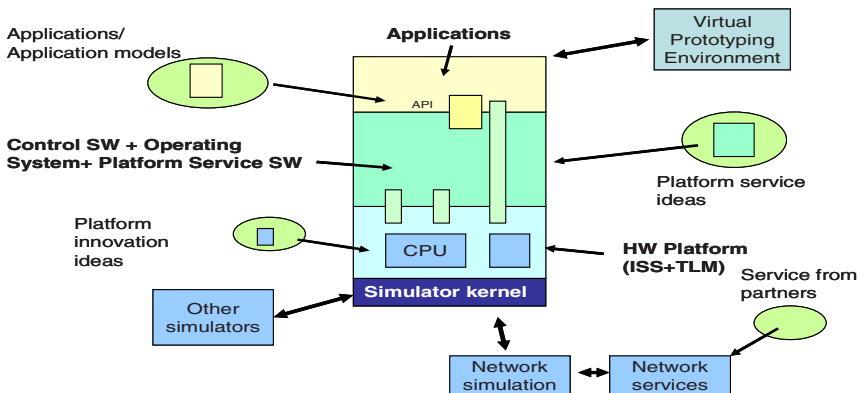


Fig. 7. Virtual prototype of a mobile terminal

impossible to access from outside of the component. It is also possible to separate the computer system part and SW development environment and to encapsulate it into distributable package for third party application developers.

4 Empirical Experiences

In University of Oulu case Cybelius Maestro tool was used to pilot software application concepts and to support the concept selection process (City of Oulu case). It was also used to gather user experiences collected concerning the selected concept (Ouman case). VTT created a case example of a mobile stereoscopic video recorder platform. The aim was to demonstrate the capabilities of virtual platform design methodology on software development. Also the feasibility of such approach and the work effort needed to make a virtual platform were quantified.

4.1 City of Oulu – Rapid Concept Design Process of Mobile Applications

For the ROOSTER-project, the City of Oulu mobilized their different working sectors; construction, cleaning, sports, education and health care in order to find new ideas for mobile phone applications. Meetings were arranged with these working sectors. The idea was to collect ideas for possible future mobile applications that could help citizens of Oulu in using their facilities as well as the workers themselves in their everyday work. The central technologies that were concerned in this research case were the Radio-Frequency Identification (RFID) tag and mobile phone. About 30 potential ideas were gathered from the meetings in total. The goal was to identify few potential ideas that would eventually be implemented as real applications.

Simulations from the collected ideas were implemented in a “quick and dirty” way because in this case no usability requirements were gathered. Simulations from these two ideas were further used for demonstration purposes in following meetings.

Maestro’s Physical User Interface (PUI) model’s front view consisted of several Key components and one view component. For example the pictures of an ID card, a RFID-tag, a Globe indicating the Internet and a computer indicating a server were in fact own components. These images were needed, because actions and data flow between these components and the phone might otherwise be hard for the user to comprehend. The created concept of the system that retrieves a map of the building the person enters in is illustrated in Fig. 8.

When the user clicks left mouse button on top of the RFID-Tag the mobile phone moves on top of it (Fig. 8.). Phone connecting to the Internet is indicated with a green arrow from the phone to the Internet to indicate the information flow. The same indication is also used to describe dataflow from the Internet to server and from Server back to the mobile phone. Indication of touching the RFID tag was implemented with a PUI-component that sets a smaller mobile phone visible on top of the RFID-tag image. After few seconds from pressing the RFID-tag, the mobile phone returns to the centre. Finally in this simulation the map of the building was opened to the phone view.



Fig. 8. A created system concept for City of Oulu case in its start and end position. RFID-Tag is touched with the phone and information is retrieved from the Internet.

The simulation style presented in Fig. 8. was used also in the implementation of the other concepts in the City of Oulu case. The development time of one simulation was approximately one day for a developer who is familiar with Maestro. In order to use Maestro efficiently the developer needs to have Java programming skill as well as skills to use some graphics software, e.g. GIMP or PhotoShop. Based on these empirical experiences, it can be said that with experienced developer, simulation tool suits extremely well in rapid design of mobile application concepts. This way mobile application can be piloted already in concept design phase, thus making the concept selection more efficient.

4.2 Ouman – User Experiences of Web-Based Application Simulation

The goal was to create a Web application representing Ouman's Web-based heat regulation system called EH-Net. A design decision was made that the simulation would represent a full scale Web server, and usability issues would be collected from users using the simulation. Because of the time constraints in the project it was decided that functionality of the simulation would be restricted to those parts of the system that concerned the user goals. Simulations were made in form of applets so that it was possible to use them remotely with Web browsers through the Internet.

Usability requirements gathering was started by arranging a stakeholder meeting. User groups along with their technical background and responsibilities were identified. The tasks and goals of these user groups were then studied. Ouman provided the "Design guidelines" and "Style guides" that were used in interaction design of the simulation. Ouman also defined usability requirements for the system. Also description of the environment of use and restrictions of use for different groups were defined.

In Maestro tool the PUI model is defined with front - and back views, where an image of the device for example a specific mobile phone model can be inserted along with various key components including Key, Liquid Crystal Display (LCD) and Led. Keys are used for example to simulate the mobile phone's key presses, whereas LCD is the view area of the device. Led's can be used to indicate different states of the device for example power on/off. Applet representing a Web server was done using only the LCD component. Maestro GUI components have also a feature called touch screen that enables the use of simulations without externally defined key triggers.

Every component contains specific set of methods that can be called from the simulation at runtime. The only prerequisite is that the touch screen feature is enabled from the component properties. For example in Ouman simulation dropdown menus functionality was implemented by grouping the functionalities of TextField, List and Button component. List is set invisible when entering the state and visible after the Button is pressed. List is set invisible again after a list item is selected from the list. Finally the text from the selected list item is set to the TextField component.

Three iterations were implemented to the simulation before launching the final larger scale usability test session. All the needed graphics were implemented already in the first iteration; second and third iteration contained only logical fixes. The third iteration contained only cosmetic fixes and therefore it was considered to be good enough quality for starting the larger scale usability testing.

It is important to notice that the test done here is not usability testing in the sense it is generally understood. Usability testing usually involves a controlled experiment and such an environment was not implemented in this research. The testing done in this research was more like usability appraisal, and it was done remotely. Nevertheless, real usability issues were found. The test group consisted of six persons with some experience in software development and user interfaces. Nielsen explains that the best results from usability testing come when testing with 5 users and running as many small tests as is affordable. Nielsen's has also described that 5 testers can find over 75% of the usability problems. With 6 testers the amount increases already to nearly 90% [17.]. Based on this information the test group's size was sufficient.

Usability issues were gathered by creating an applet from the Web application simulation and storing the HTML-file containing the applet to a Web server. The applet could then be run by test users' using their PC's Web browsers. Data received from the questionnaire was qualitative. Some of the answers could not be counted valid usability issues. This is because some usability flaws found by the user were actually simulation tool and simulation specific problems. Only usability issues that could be identified to concern the real system were collected.

The implementation work of the simulations grows exponentially when the focus is on the usability of a product. There are numerous features that need to be implemented in order to create a simulation that can offer adequate degree of realism that is comparable with the physical and logical functionality and the appearance of the possible real object. In this case it took a full month to implement the first version of the simulation that could be used to gather user experiences and usability issues. Nevertheless, empirical data from this case shows that virtual prototypes can be used to gather valid usability issues and thus they can provide support for planning the usability of a product.

4.3 Virtual Platform Model of a Mobile Device

The aim of the case was to demonstrate the capabilities of virtual platform design methodology on software development. Also the feasibility of such approach and the work effort needed to make a virtual platform were quantified [23].

A SystemC based toolset from CoWare was used in this work for developing the virtual platform hardware model. The SystemC language [24] has a dual role in ESL (Electronic System Level) design as being both a hardware description language and a

system description language. CoWare is one of the few tool vendors which currently provide virtual platform technology. The tools allow the user to build a whole platform from scratch by using models of commercial IP blocks or by building synthesizable processors models and any other custom IP blocks. The models used by the CoWare tools are SystemC based, so models of various abstraction levels can be incorporated in the platform. The platform designer can create a virtual platform of his design to be distributed to software developers. Software designers then receive a virtual platform package which includes a compiler, a debugger and the platform simulator. As software designers give feedback of the system issues back to the platform designer, they will receive an updated version of the virtual platform package later on.

The stereoscopic video recorder which was chosen as the case example is a device which multiplexes and encodes video streams from two independent cameras representing the human visual system. This creates a video stream which captures a true three dimensional view. The approach is viable in future mobile phones for example due to the rapid development camera technology and compact autostereoscopic liquid crystal displays. Both encoding and decoding modes were implemented in the virtual platform for approach feasibility studies and architecture exploration purposes. Performance of these platforms and the performance of the simulator were measured. During the architecture exploration, we also approximated the time taken by each step in creating and modifying the platform. The amount of time consumed was also approximated for the consecutive changes made to the platform after the initial learning curve had been confronted. This allowed us to quantify the effort in learning the method and tools used.

An ARM9 processor family based platform was chosen, because it is one of the most popular processor types for embedded multimedia devices. Designing the hardware platform block diagram was straightforward and intuitive. Different platform variations were tested for the 3D recorder, such as running the instructions from the fast SRAM memories or the external memories over the bus.

The software development for the virtual platform proved to be very straight forward. The initial port of the stereoscopic codec was developed in a PC environment with a gcc cross compiling environment. An operating system was not used in the virtual platform. Instead the stereoscopic video encoding software was a standalone ARM-binary, because this was considered to be enough for performance evaluation of the most critical parts of the system. System boot up codes were naturally required to setup the system prior to the execution of the video encoder in addition to the actual codec work. Ffmpeg / libavcodec open source encoding/decoding software was chosen as the starting point for the stereoscopic encoding and decoding task. The codec is customizable, but is not fully optimized for ARM9. Modifications to the original code were made.

The hardware modeling was done by one research and the software was created concurrently with the hardware by a research scientist with background in video encoding software. As with all new tools, virtual platform involves a significant learning curve, so the results for time consumed in model changes are displayed in two categories. The first one is the amount of work for the first time user making a building or changing the platform. The time to build a new hardware platform from scratch was measured to be eight weeks for a novice user. The total time to build a working

platform with all the software and peripherals included was 16 weeks. This amount includes the concurrent software and hardware development. After building the first platform and encountering the initial obstacles, it was possible to build a working virtual platform model from scratch in a week.

The CoWare tools could perform virtual platform simulation with two modes: fast and full. The fast mode was suited for functional verification, but did not give accurate timing data for performance approximation. On the other hand the full mode was accurate, but approximately ten times slower than the fast mode. The fast simulation mode reached 11.8 MIPS performance on a 3,2GHz Intel Xeon workstation. This is 9.6 times slower than real time. This gap is becoming smaller as simulator workstations and the simulation tools improve.

5 Conclusions and Future Work

Virtual prototypes can be used to gather usability requirements already in early phase of product development life cycle reducing the amount of errors that are implemented. Paper prototyping or story boarding enables the rapid concept designing of certain parts of software's user interface, but with virtual prototyping and simulation tools it is possible to produce fully interactive simulations. With interactive simulations it is possible to find even more relevant usability issues from a software product. Virtual prototypes enable the rapid concept design of software applications user interface already before the implementation work has started. Virtual prototypes can therefore be used to pilot software application concepts and support the concept selection process. The application designers can get realistic environment where they can design and test not only the logic of the UI, but also the timing, delays and many other crucial time related issues in UI design. Application designers get access to a simulated realistic real-time mobile device well before the first prototypes are available from the device manufacturer. The tools can be used for piloting and testing of mobile software applications and platforms and how these virtual prototypes can support planning and collecting the user experiences.

Virtual platform modeling is a feasible approach for both software and hardware development. The tools used in this work also proved to be suitable for architecture exploration and performance evaluation. The functionality of the platform can be modeled very accurately with virtual platforms and they provide an intuitive way of developing and testing software for both functional and performance estimation. The functional verification simulations used in our work were still 9.6 times slower than real time, but this gap is becoming smaller as simulator workstations and the simulation tools improve.

Table 1. Simulation development times in research cases

UI simulation	1 day
Application simulation	4 weeks
Working virtual platform model	1 week

Our experiences (Table 1.) with integrated prototyping tools indicate that it is possible to shorten development time considerably. However, it was difficult to find benchmarking data for comparing our results and hence further empirical research is required to quantify the improvements.. Also, tight integration of UI (e.g. Maestro) and platform (e.g. CoWare) simulation tools would be needed to enable optimal use of both simulation techniques. This kind of tight tool integration would be a promising and interesting future research area.

Acknowledgments

The research work presented in this paper was done in ROOSTER research project at the Department of Information Processing Science, University of Oulu and at VTT. Project was financially supported by the National Technology Agency of Finland (TEKES) and industrial partners CCC, Nokia, Elektrobit, F-Secure, TietoEnator, Jaakko Pöyry Oy, Pöyry Telecom Oy, Ouman, Embe Systems Oy and City of Oulu.

References

1. Austin, T., Larson, E., Ernst, D.: SimpleScalar: An Infrastructure for Computer System Modeling. *Computer* 35(2), 59–67 (2002)
2. Benini, L., Bertozzi, D., Bruni, D., Drago, N., Fummi, F., Poncino, M.: SystemC Cosimulation and Emulation of Multiprocessor SoC Designs. *Computer* 36(4), 53–59 (2003)
3. Bevan, N.: International standards for HCI and usability. *International Journal of Human-Computer Studies* 55 (4), 533–552 (2001)
4. Buck, J., Ha, S., Lee, E., Messerschmitt, D.: Ptolemy: A Framework for Simulating and Prototyping Heterogeneous Systems. *International Journal of Computer Simulation* 4, 152–182 (1994)
5. Chandra, S., Moona, R.: Retargetable Functional Simulator Using High Level Processor Models. In: *Proceedings of 13th International Conference on VLSI Design, 2000, Calcutta, India, January 3–7, 2000*, pp. 424–429. IEEE Computer Society Press, Los Alamitos (2000)
6. Grötter, T., Liao, S., Martin, G., Swan, S.: *System Design with SystemC*, p. 217. Kluwer Academic Publishers, Boston (2002)
7. Haug, E.J., Kuhl, J.G., Tsai, F.F.: *Virtual Prototyping for Mechanical System Concurrent Engineering*. In: Haug, E.J. (ed.) *Concurrent Engineering: Tools and Technologies for Mechanical System Design*, pp. 851–879. Springer, Heidelberg (1993)
8. Hughes, C., Pai, V., Ranganathan, P., Adve, S.: Rsim: simulating shared-memory multiprocessors with ILP processors. *Computer* 35(2), 40–49 (2002)
9. ISO/IEC 13407: *Human-Centered Design Processes for Interactive Systems*.1999: ISO/IEC 13407: 1999 (E) (1999)
10. ISO/IEC 9241-11: *Ergonomic requirements for office work with visual display terminals (VDTs). Part 11 - Guidelines for Specifying and Measuring Usability*.1998: ISO/IEC 9241-11: 1998 (E) (1998)
11. Jain, R.: *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modeling*, p. 685. John Wiley & Sons, Inc, New York (1991)

12. Jokela, T.: Making User-Centred Design Common Sense: Striving for an Unambiguous and Communicative UCD Process Model. In: ACM International Conference Proceeding Series, vol. 31, pp. 19–26 (2002)
13. Kerttula, M.: Virtual Design. A Framework for the Development of Personal Electronic Products. VTT, Finland (2006)
14. Kiljander, H.: User Interface Prototyping of Handportable Communication Products. Academic Licentiate Thesis, p. 122. Helsinki University of Technology, Espoo, Finland (1997)
15. Lahiri, K., Raghunathan, A., Dey, S.: Performance Analysis of Systems with Multi-Channel Communication Architectures. In: Proceedings of 13th International Conference on VLSI Design, Calcutta, India, January 3–7, 2000, pp. 530–537. IEEE Computer Society Press, Los Alamitos (2000)
16. Mayhew, D.J.: The Usability Engineering Lifecycle, a practitioner's handbook for user interface design, 4th edn. Morgan Kaufmann Publishers, Inc., San Francisco (1999)
17. Nielsen, J.: Why You Only Need to Test With 5 Users. [Web-document] (2000) [Referenced 1.6.2007], <http://www.useit.com/alertbox/20000319.html>
18. Tseng, M.M., Jianxin, J., Chuan-Jun, S.: A framework of virtual design for product customization. Emerging Technologies and Factory Automation Proceedings 9(12), 7–14 (1997)
19. Ulrich, K.T., Eppinger, S.D.: Product Design and Development. McGraw-Hill, Inc., New York (1995)
20. Živojnović, V., Meyr, H.: Compiled SW/HW Cosimulation. In: Proceedings of 33rd Design Automation Conference, Las Vegas, NV, USA, June 3–7, 1996, pp. 690–695. ACM Press, New York (1996)
21. See for example: <http://www.systemc.org>
22. Kreku, J., Kauppi, T., Soininen, J.-P.: Evaluation of platform architecture performance using abstract instruction-level workload models. In: International Symposium on System-on-Chip, Tampere, Finland (2004)
23. Eteläperä, M., Vätjus-Anttila, J., Soininen, J.-P.: Architecture Exploration of 3D Video Recorder Using Virtual Platform Models. In: 10th EUROMICRO CONFERENCE on DIGITAL SYSTEM DESIGN Architectures, Methods and Tools (2007)
24. See: <http://www.systemc.org>

Comparing Assessment Methodologies for Free/Open Source Software: OpenBRR and QSOS*

Jean-Christophe Deprez and Simon Alexandre

Centre d'Excellence en Technologies de l'Information et de la Communication (CETIC),
Charleroi, Belgium
{Jean-Christophe.Deprez, Simon.Alexandre}@cetic.be

Abstract. Many organizations using Free/Open Source Software (F/OSS) are dealing with the major problem of selecting the most appropriate software product corresponding to their needs. Most of these companies are currently selecting F/OSS projects using ad-hoc techniques. However, in the last couple of years, two methodologies for assessing F/OSS project have emerge, namely QSOS and OpenBRR. The objective of this work is, through a detailed and rigorous assessment methodology comparison, to allow companies to have a better understanding of these two assessment methodologies content and limitation. This work compares both methodologies on several aspects, among others, their overall approaches, their scoring procedures and their evaluation criteria.

Keywords: assessment methodologies, open source, free software.

1 Introduction

Many organizations have started to integrate Free (*libre*) Open-Source Software (F/OSS¹) in their products, systems and infrastructures. Furthermore, software solutions that rely on F/OSS components become more frequently available to customers. In turn, organizations want assurance regarding the quality of F/OSS projects before integrating them in their solutions.

Having identified this need, several methodologies help select appropriate F/OSS projects have surfaced in the past couple of years. Two prominent methodologies are the Qualification and Selection Open Source (QSOS) backed by Atos Origin and Open Business Readiness Rating (OpenBRR) created by Carnegie Mellon West and Intel. Although fairly light weight, these two methodologies help shed lights on the seriousness of F/OSS projects.

The contribution of this paper is to compare the two assessment methodologies, OpenBRR and QSOS. In the end, it helps identify which of the two methodologies better fits one's context. Our comparison is performed based on the description of the methodologies and not on their empirical application. Based on our findings, our future work will aim at creating a new methodology, namely, the QUALOSS

* This work is partly funded by QUALOSS (#33547), a research project funded under the FP6 programme of the European Commission.

¹ F/OSS stands for Free *libre* Open Source Software.

methodology that takes advantages of the strong points of QSOS and OpenBRR while eliminating the weaknesses of both.

The rest of this paper is structured as follows. Section 2 presents the two methodologies. Section 3 introduces our comparison approach. Section 4 compares OpenBRR and QSOS. Section 5 reviews the advantages and weaknesses of both methodologies. Section 6 discusses the related work and Section 7 concludes with our future work.

2 Description of QSOS and OpenBRR

Both assessment methodologies help their users reach a similar goal, that is, select among a list of similar F/OSS projects for the one best suited to their context. The two following subsections present QSOS and OpenBRR respectively.

2.1 QSOS

This section presents version 1.6 of QSOS, which appears in [1]. The top part of the QSOS methodologies consists of the following series of steps:

- *Start from a list of F/OSS projects whose products seems to fit with the overall requirements (set by the product user or product integrator)*
- Evaluate each F/OSS project along the list of evaluation criteria given by QSOS. This step assigns an absolute score to each evaluation criterion.
- *The evaluator can adjust the importance of each criterion based on the particular context. This is done by assigning a weight and threshold to each criterion.*
- The scores obtained using 2 are weighted based on 3 so as to get the relative score corresponding to the given context. The outcome of this step is a ranking that determines the qualification or elimination F/OSS projects.
- QSOS then suggests trying the top F/OSS projects that meet qualification criteria. This number can vary between 2-5 F/OSS products depending on the criticality of the decision.

QSOS also provides a tree-hierarchy of evaluation criteria shown in Figure 1 along with a procedure to score each leaf criterion. QSOS splits its evaluation template in two sections: a generic section and a specific section. The generic section includes criteria that apply to all software products while the criteria of the specific section include an expected list the functionality and therefore varies according to software product family such as groupware, cms, database, etc. Due to space consideration, we only present evaluation criteria of the generic section. It is worth emphasizing that the templates listing functionality of software product families is specific to the web version of QSOS (at www.qsos.org) but it is not described in [1].

Figure 1 shows the tree hierarchy of evaluation criteria in QSOS's generic section. We note that this hierarchy comes from the paper version (.pdf) of QSOS 1.6 and not the web version of the template, which slightly differs. In particular, the web version lists the item "Independence of development" under the "Industrialized Solution" category whereas the paper version includes it "Intrinsic Durability". Second, the web version elevated "Exploitability" as a top-level category where as the paper version includes it as a sub-category of "Industrialized Solution". Finally, the paper version

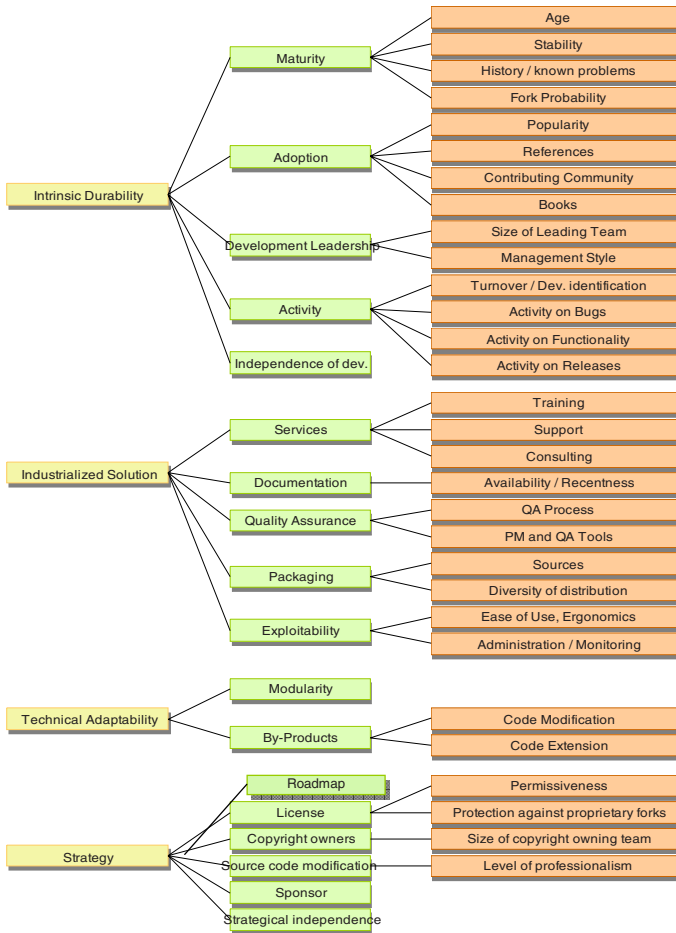


Fig. 1. Generic criteria from QSOS version 1.6

contains a section on “Service Providing” that list a few criteria to evaluate the ability of a service provider to help with the integration of a FLOSS component.

Along the hierarchy of Figure 1, QSOS gives a procedure to score each leaf criteria. A valid score is 0, 1 or 2. Table 1 shows a sample of the scoring procedure for three leaf criteria. We point to [1] for the description of the whole scoring procedure.

2.2 OpenBRR

OpenBRR [2] asks to follow these high level steps:

1. Perform a pre-screening (Quick Assessment). This step starts with a long list of FLOSS projects whose software product fits the overall requirement and ends with a few viable candidates for the final selection. This initial filtering is based on the criticality of the product or system that will integrate the FLOSS component as well as a handful of viability criteria determined based on the context.

Table 1. QSOS scoring procedure for three leaf criteria

Criteria	Score = 0	Score = 1	Score = 2
<i>Age</i>	Less than 3 month old	Between 3 month old and 3 year old	More than 3 year old
<i>Training</i>	No offer of training identified	Offer exists but is restricted geographically and to one language or is provided by a single contractor	Rich offer provided by several contractors, in several languages and split into modules of gradual levels
<i>Source Code Quality</i>	Not very readable code or of poor quality, incoherence in coding styles	Readable but not really commented in details	Readable and commented code implementing classic design patterns with a coherent and applied coding policy

2. Tailor the evaluation template (Target Usage Assessment): This step consists of reviewing and selecting the appropriate evaluation criteria from the hierarchy proposed by OpenBRR. One may also decide to add new criteria and procedure to score these criteria. The outcome is a customized version of the evaluation template fit to the context, which is usually defined by an organization, the kind of F/OSS software product to select, the criticality of the product in which the F/OSS component will be integrated, etc.
3. Data Collection and Processing: This step starts from the list of the F/OSS projects that passed the viability checks of point 1 above. It consists in collecting the raw data needed to score the evaluation criteria selected during point 2 and to apply the weight factor associated to each metrics determine during point 2 as well.
4. Data Translation: Aggregate the relative metric scores so as to obtain the score of each category. Each category score is then adjusted based on its weight factor determined during point 2. The final business readiness rating can then be published.

OpenBRR proposes a set of evaluation criteria for points 1 and 2 in our above list. In relation to point 1, the quick assessment, OpenBRR suggests looking at the following 8 issues and eventually adapting the list to better fit a given context:

(1) licensing, (2) standard compliance, (3) referenceable adopters, (4) availability of support, (5) implementation language(s), (6) third party reviews, (7) books, and (8) review by industry analysts such as Gartner.

For these criteria, OpenBRR let its user determine the scoring procedure and eventually what is a make or break situation. For example, if an organization only considers a F/OSS project when it is published under a license compatible with BSD then no use wasting time evaluating F/OSS components with a stronger copyleft bend.

Concerning point 2 above, the target usage assessment, OpenBRR proposes the template of evaluation criteria shown in Figure 2. In addition, OpenBRR gives a scoring procedure to evaluate the leaves of Figure 2. The procedure assigns a score between 1 and 5 where 1 is unacceptable and 5 is excellent. Due to space consideration, we only describe the procedure for three criteria in Table 2.

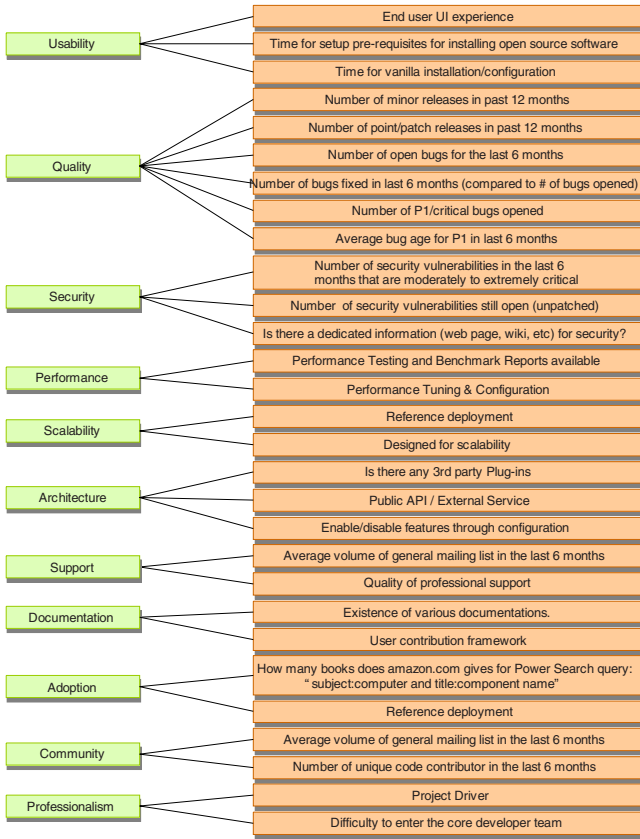


Fig. 2. OpenBRR hierarchy of evaluation criteria

Table 2. OpenBRR scoring procedure for three leaf criteria

Criteria	1	2	3	4	5
<i>Time for vanilla installation</i>	> 4 hours	1-4 hours	30min to 1 hours	10-30 minutes	< 10 minutes
<i>User Contribution Framework</i>	Users cannot contribute		Users are allowed to contribute		Users are allowed to contribute and contribution are edited / filtered by experts
<i>Reference Deployment</i>	No		Yes		Yes, with publication of user's size

3 Comparison Approach

Our comparison approaches is divided in several comparison exercises:

1. Comparison of the overall steps of the methodologies.
2. Analysis of the scoring procedures.

3. Coverage analysis of the evaluation criteria and adequacy of the hierarchies of criteria

The comparison of the overall steps first highlights the similarities and differences, including those related to how each methodology is intended to be applied in the field.

The analysis of the scoring procedures addresses the following three questions.

1. Is the range of scores adequate?
2. Is the scoring procedure of each evaluation criteria unambiguous?
3. Is the raw data required by the procedure in order to compute its result likely to be available in the field?

The coverage analysis compares the evaluation criteria between QSOS and OpenBRR in order to determine which are similar vs. those only present in only one of the methodologies. In addition, we also quickly assess the accuracy of the terminology used to express criteria and categories of criteria in each methodology.

4 Comparing QSOS, OpenBRR

In this section, we compare QSOS and OpenBRR based on the comparison approach introduced in Section 3.

4.1 Comparison of the Overall Approaches

This section compares the overall approaches by highlighting first, their similarities and second, their differences.

Both methodologies are similar in the following aspects:

- Each methodology proposes a predefined set of criteria for evaluating FLOSS projects. The set of criteria is categorized into a tree hierarchy of 2 levels for OpenBRR or of 3 levels for QSOS.
- The evaluation consists of scoring the various criteria based on a standard scoring procedure. During the evaluation of a given FLOSS project, this step results in assigning score to each criterion. We refer to this score as absolute.
- Users can adjust the importance of each criterion according to their context by varying the weight assigned to each criterion. In particular, during an evaluation, the absolute scores are weighted based on their importance to the current evaluation context. We refer to the weighted absolute scores as relative scores.
- A decision can be taken based on the resulting relative scores.

However, the 2 methodologies do differ in the order in which they apply the points above. The current order reflects that of QSOS while OpenBRR suggests inverting point 2 and 3 so that users first select criteria relevant to their context and therefore avoid scoring useless ones. In addition, OpenBRR allows the creation of new criteria as well as the tailoring of the scoring procedure for criteria.

These variations result from the difference in how each methodology is to be applied in the field.

QSOS believes that the absolute scores obtained when applying the scoring procedures are universal. Hence, the scoring procedure for a particular version of a FLOSS

project only takes place once. Others can then comment on the evaluation if a score seems unfair. However, once it is agreed on, the absolute scores of the given version of a F/OSS projects are universal and eventually made available to everyone. The only way to adjust the final outcome of an evaluation is to adapt the weights assigned to evaluation criteria.

OpenBRR on the other hand is a standard methodology but it assumes that every user instantiates it in a slight different way. Hence, the evaluation of a particular F/OSS project would result in slightly different scores depending on the context in which the evaluation is performed. The result of an evaluation is therefore not meant for reuse, even the absolute scores obtained for the evaluation criteria. In the case of OpenBRR, this seems a wise decision since the user is free to eliminate and add evaluation criteria as well as to modify and create new scoring procedures for new or existing criteria. In conclusion, OpenBRR provides a standard methodology that is expected to be tailored in practice.

As a result of the difference between the application and reuse strategies of QSOS and OpenBRR, it is easy to find a growing repository of F/OSS product evaluations for QSOS but not for OpenBRR.

The important question is: “Can a methodology be universal, at least partially such as QSOS, so that intermediate score (such as the absolute score of QSOS) can be re-used and shared?”

First, QSOS can apply such a strategy because its scoring procedure only allows a range of three scores 0, 1, and 2. Hence, the variance of scores between different evaluator is reduced. We delay further discussion on the range of scoring procedures for our next comparison exercise in the next subsection. Second, we believe that providing a one size-fits-all scoring procedure is not adequate for every criterion. For example, QSOS attributes a score based on the number of books published. Some F/OSS products address niche markets and the publication of books is very unlikely however, various technical articles in professional magazine may very well be of interest and a good substitution to the book published criterion. Although QSOS could allow such a tailoring in the scoring procedure, we have not seen it in practice, at least in evaluation sheet accessible via <http://www.qsos.org/sheets/index.html>.

Finally, it is worth noting that QSOS defines the scope of an evaluation based on the particular version of a F/OSS product while this information is not clear for OpenBRR. Hence, we may suppose that OpenBRR intend to leave that issue open so that it is possible for one to apply decide if the scope is a whole project or just a specific version of a particular F/OSS product. Leaving this decision to the users raises the following concern: When applying scoring procedures it is very important to clearly define the scope of the dataset so not to include data related to other versions. In some cases, this is not always easy or feasible. For example, the number of post on forums or the number of book published may not be about the particular version being evaluated. Hence, this may make the scoring procedure ambiguous.

4.2 Comparison of the Scoring Procedures

Both methodologies provide a scoring procedure in order to transform raw data into a score assigned to evaluation criteria. Table 1 and 2 respectively show a sample of the scoring procedures of QSOS and OpenBRR. Below we compare the complete scoring

procedures based on the 3 checks mention in section 3, in particular, the score range, the scoring procedure clarity/ambiguity, and the data availability.

Score Ranges

QSOS proposes a procedure whose result assigns a discreet score between 0 and 2, that is, 0, 1, or 2 to each evaluation criteria. Unfortunately, this range seems too restrictive to appreciate fully the information, at least, for certain criteria. We feel that a minimum of 4 levels would be required. With just 3 levels, the middle score may have true mid position but it may embed a positive or negative tilt.

OpenBRR has a procedure that assigns a discreet score between 1 and 5. In this context, a 5-level score is adequate. It is clear that 1 and 2 are negative while 4 and 5 are positive. Allowing a neutral score of 3 is also acceptable. However, we observe that in 14 of the 28 evaluation criteria, scoring rules do not use all 5 levels; in 13 cases, 3 of the 5 levels are used, in particular, 1, 3, and 5 skipping 2 and 4 and in the remaining case, 4 levels are used: 1, 3, 4, and 5. In turn, for more than half of the evaluation criteria, the procedure is no better than the 3 levels offered by QSOS.

Clarity and ambiguity of scoring procedures

Both scoring procedures lack clarity in certain of their scoring rules. We determined that the scoring rule of a criterion was ambiguous if the wording was unclear or if it could be interpreted differently by different people or in different contexts or also, if we identified a gap in the description of the scoring procedure, for example, the description given is clear for each score but there are many real-life situations not accounted for where it would be hard to settle on an actual score.

Our analysis finds the following:

QSOS contains a total of 41 evaluation criteria and 22 of them are found to be ambiguous. The 22 scoring rules we found ambiguous are for the following criteria: *stability, fork probability, popularity, references, contributing community, management style, activity on bugs, on functionality, on release, training, support, consulting, documentation availability, PM and QA tools, ergonomics, admin/monitoring, modularity, code modification, source code modification – level of professionalism, source code quality, intrinsic complexity, and technical documentation.*

The scoring procedure for *source code quality* is given in Table 1.

OpenBRR has 28 evaluation criteria. In most cases, evaluation criteria have much more specific meanings hence this leads to scoring rules that are much more precise. Nonetheless, we found 7 ambiguous cases: *end-user UI experience, is there a dedicated information for security?, performance testing and benchmarks, performance tuning and configuration, design for scalability, quality of professional support, and difficulty to enter the core development team.*

As already mentioned in Section 4.1, beside clarity, we can also question scoring rules on their range of applicability to the world of software products and components. However, we must be careful on the value of this comparison. OpenBRR recognizes that its rules may not be applicable to all situations hence allows tailoring by the evaluator. On the other hand, QSOS aims at providing scoring rules that compute universal scores hence it is more important for QSOS to propose generic rules. Incidentally, ambiguous rules usually seem more generic and it is thus the likely reason why more than half of QSOS's rules were found ambiguous. Furthermore, QSOS is capable to achieve a certain level of universality in its rules because its scores only vary between three discreet outcomes. Having a fourth or fifth outcome would make

it much harder for rules to stay generic. Unfortunately, as we pointed out earlier, a three point scale is likely not enough to truly help in good decision making.

Likelihood of data availability

In addition to the clarity of a rule, it is also important that the data requested be available otherwise the lack of data also put the applicability of the methodology at stake. When determining that some data is unavailable, it may mean that the data will be really hard to find but it can also suggest that the data is not readily available. That is, the raw data is not available and obtaining it would require posting a question on a forum hence would depend on the friendliness of community members and whether the members who answered actually know the correct data.

For QSOS, we found that 5 criteria have scoring rules requesting data unlikely to be available, in particular for the following criteria, *history/known problem*, *fork probability*, *management style*, *developer identification/turnover*, *independence of developments*.

For OpenBRR, we determined that 9 criteria asked for data that would likely be unavailable: *time to setup pre-requisites*, *time for vanilla installation/configuration*, *number of security vulnerabilities in last 6 months*, *number of security vulnerabilities still open*, *performance testing and benchmarks*, *performance tuning and configuration*, *reference deployment*, *designed for scalability*, *difficulty to enter the development team*.

4.3 Coverage of the Evaluation Criteria and Quality of Wording

This section first studies the similarities and differences among the evaluation criteria of QSOS and OpenBRR. This exercise is done for the leaf criteria of both methodologies. Second, we analyze the adequacy of the terminology used by both methodologies.

We start from the QSOS hierarchy and compare every leaf criteria with those of OpenBRR, including the viability criteria used in the quick assessment set, that is, the first pre-filtering step of OpenBRR.

The possible results of a pair wise comparison between two criteria A and B are:

- The two criteria are equivalent ($A = B$)
- One criterion is a more generic than the other ($A < B$ (A is a special case of B) or $A > B$ (A is a more general case of B)),
- The two criteria have some similarity relationship of a fuzzy nature ($A \sim B$), for example, A may influence B or vice versa.
- The two characteristics have nothing in common

We must emphasize that our coverage analysis actually does not compare the criteria based on the semantic of their wording but rather compares them based on the semantic of their scoring rules.

In the comparison table shown in Table 3, the left column enumerate all QSOS characteristics and then, for every leaf characteristic of QSOS, we indicate whether OpenBRR has corresponding characteristic with one of the relationship signs identified above ($=$, $<$, $>$, or \sim). This is shown by the relationship sign preceding the characteristics in the OpenBRR columns.

Table 3. Coverage analysis between QSOS and OpenBRR leaf criteria

QSOS			OpenBRR
Intrinsic Durability	Maturity	Age	NONE
		Stability	~ all Quality sub-criteria
		History, known problems (= Management Ability)	NONE
		Fork probability	NONE
	Adoption	Popularity	= Referenceable Adopters (from Quick Assessment)
		References (= level of mission criticality of references)	NONE
		Contributing community (= volume and diversity of community contribution)	> Community .. Average volume on general mailing list in the last 6 months > Community .. Number of unique code contributor in the last 6 months
		Books (number of books published about products)	= Adoption .. How many Books ...
	Development leadership	Leading Team (= Size of leading team)	NONE
		Management style (= level of democracy of management)	~ Professionalism .. Project Driver ~ Professionalism .. Difficulty to enter core developer team
	Activity	Developers identification, turnover	~ Professionalism .. Difficulty to enter core developer team
		Activity on bugs	= Quality .. Number of open bugs, .. number of fixed bugs, and ..average bug age in the last 6 months + .. number of P1/critical bugs opened
		Activity on functionalities	NONE

Table 3. (continued)

QSOS		OpenBRR	
	Activity on releases	= Quality .. number of minor releases and .. number of point/patch releases in past 12 months	
	Independence of development	~ Professionalism .. Project Driver	
Industrialized Solution	Services	Training (Diversity in geographical, cultural and gradual aspects)	NONE
		Support (Level of commitment assigned to support)	~ Support .. Quality of professional support
		Consulting (Diversity in geographical and cultural aspects)	NONE
	Documentation (Availability and recency of documentation)		~ Documentation .. Existence of various kinds of documentation
	Quality Assurance	Quality Assurance Process	~ Performance testing and benchmark reports available
		PM and QA Tools	NONE
	Packaging	Sources	NONE
		*nix packaging	NONE
	Exploitability	Ease of use, ergonomics	> Usability .. time for vanilla installation/configuration
Administration/Monitoring (Availability of functionality for administration and monitoring)		NONE	
Technical adaptability	Modularity (Software modularity)	~ Scalability .. Design for scalability ~ Architecture .. Are they any third party plug-ins? ~ Architecture .. Public API / External Service	

Table 3. (continued)

QSOS			OpenBRR
	By-Products	Code modification (Ease of build-ability)	NONE
		Code extension (Extensibility or plug-ability)	= Architecture .. Are they any third party plug-ins? AND Architecture .. Public API / External Service
Strategy	License	Permissiveness	~ Licensing/Legal (in the quick assessment)
		Protection against proprietary forks	~ Licensing/Legal (in the quick assessment)
	Copyright owners (Size of copyright owning team)		NONE
	Modification of source code (Level of professionalism of procedure for proposition of modification.)		~ Professionalism .. Difficulty to enter core developer team
	Roadmap (availability + precision of the roadmap)		NONE
	Sponsor (Driving force behind product)		= Professionalism .. Project Driver
	Strategical independence		~ Professionalism .. Project Driver
Services Providing	Maintainability	Quality of Source Code (Volume of comment and use of design pattern)	~ Scalability .. design for scalability ~ Performance .. Tuning & Configuration (on user's end) ~ Architecture .. Are there any third party plug-ins? ~ Architecture .. public API / External service
		Technological dispersion (number of prog.lang. used)	~ Implementation language (in the quick assessment)
		Intrinsic complexity (Complexity of algorithms)	NONE
		Technical documentation (Design and arch doc + others)	~ Documentation .. Existence of various kinds of documentation

Table 3. (continued)

QSOS		OpenBRR
	Code Mastery	Direct availability (Number of experts available within a consulting company)
		Indirect availability (Number of experts available in partner companies of serv. prov.)

From Table 3, we observe that 16 QSOS criteria are not covered by OpenBRR. Conversely, we can also derive the OpenBRR criteria not covered by QSOS using Table 3 and Figure 2. In particular, the 7 following criteria are not covered by QSOS: end user UI experience, time for setup pre-requisites for installing open source software, all 3 criteria under security, reference deployment, user contribution framework. In addition, there are also 5 criteria from the quick assessment step of Open BRR not covered by QSOS: standard compliance, availability of a supporting or stable organization, implementation languages, third party reviews and industry analyst.

Beside our coverage analysis, we also add a few comments regarding the wording used by both methodologies for their criteria as well as for the higher-level nodes in their tree hierarchies.

We find that QSOS uses a very appropriate nomenclature for the higher level nodes in its tree hierarchies. However, the leaf criteria are usually summarized in very imprecise words. This forces the investigation of the scoring rules to understand accurately the meaning of criteria. For example, the criterion References under Intrinsic Durability .. Adoption is rather unclear. Once reading the scoring rules, we find that it measures the number of cases where users use a FLOSS product in mission critical solutions. Hence, the wording mission criticality of references would be more accurate without being too lengthy. This kind of re-wording could take place for several QSOS criteria.

For OpenBRR, this is the exact opposite. The wording of metrics is accurate and extensive although sometimes quite lengthy. Many criteria could therefore be re-worded in shorter phrases without losing clarity. Concerning, the top node in the tree hierarchy, we find that the terms used are often very broad and inaccurate. For example, Quality is much too broad and a term such as stability or reliability would be more appropriate.

5 Advantages and Disadvantages of QSOS and OpenBRR

This section reviews the advantage and disadvantages of both methodologies. Besides helping decide which methodology to use, this comparison exercise will be our starting point to create a new methodology that preserves most advantages of both methodologies while getting rid of the disadvantages.

	Advantages	Disadvantages
QSOS	<ul style="list-style-type: none"> • Open repository of evaluation scores for various F/OSS projects (this pushes evaluators to collaborate on evaluation and to facilitate cross validation) • Extensive list of criteria • Interesting innovating nomenclature for the tree hierarchy • QSOS methodology is versioned and evaluation mention the QSOS version used 	<ul style="list-style-type: none"> • Ambiguous scoring rules for more than half of the criteria • Scoring procedure with 3-level scale may make decision making harder • Universality of scoring rules is not possible for many criteria
OpenBRR	<ul style="list-style-type: none"> • Allows for tailoring hence better fit one's evaluation context • Clearer scoring procedure with fewer ambiguities • 5-level scoring scale for about half of the criteria • Ask evaluator to perform a quick assessment step to reduce the evaluation effort 	<ul style="list-style-type: none"> • No open repository of evaluation (due to possible tailoring) • Does not exploit the 5-level scales for more than half of the criteria • Terminology is broad and imprecise for the top nodes in the hierarchy • OpenBRR does not seem to be versioned. However, this may be left to the evaluator

In addition, we find that both methodologies have a particular important weakness. They do not require evaluators to capture the location of the raw data used to obtain the evaluation scores. This makes it hard to refute or argue the correctness of an evaluation. However, we did find that in practice, several QSOS evaluation sheets list URL's where raw data used for evaluation are mentioned.

6 Related Work

Prior to QSOS [1] and OpenBRR [2], other F/OSS evaluation methodologies were proposed, notably, two of them called Open Source Maturity Model respectively created by Golden from Navica [3] and by Frans-Willem Duijnhouwer from CapGemini [4]. In addition David Wheeler also proposed a very high level methodology to quickly evaluate F/OSS projects. These three efforts were used as a stepping stone by OpenBRR. On the other hand, OpenBRR and QSOS were created in parallel and to the best of our knowledge we are the first effort comparing F/OSS assessment methodologies.

An orthogonal body of research studies whether F/OSS development allows reaching software component of higher quality; an example of such efforts is found in [5]. These research endeavors have the objectives to determine how F/OSS development differs from the traditional methods used in the proprietary world and also to identify whether these differences impact the quality of code and of software products. On the

other hand, the evaluation methodologies compared in this work do not argue that F/OSS is better than proprietary. Rather, they give a mean to evaluate and to compare among several F/OSS alternatives without taking a stand on whether F/OSS or proprietary yields better quality.

On a more general note, the European Commission is currently funding several research projects related to open source and quality, namely, QUALOSS [6], FLOSS-METRICS [7], SQO-OSS [8], and QUALIPSO [9]. The first project listed is led by the authors of this article.

7 Future Work

Based on the comparison exercises presented in this paper, our future goal is to derive a new methodology for evaluating F/OSS projects. As OpenBRR learned from previous works, the QUALOSS project aims to bring F/OSS assessment to a higher level of objectivity, completeness, and clarity.

The goal is to create a methodology applicable at different level of thoroughness. A first light level will closely resemble QSOS and OpenBRR in principle with most of the advantages and without the shortcomings.

References

1. Method for Qualification and Selection of Open Source software (QSOS) version 1.6 © Atos Origin (April 2006), <http://qsos.org/>
2. Business Readiness Rating for Open Source © OpenBRR.org, BRR 2005 – Request fro Comment 1 (2005), <http://www.openbrr.org>
3. Golden, B.: Open Source Maturity Model © Navica, <http://www.navicasoft.com/pages/osmmoverview.htm>
4. Widdows, C., Duijnhouwer, F.-W.: Open Source Maturity Model © CapGemini (August 2003), <http://www.SeriouslyOpen.org>
5. Aberdour, M.: Achieving Quality in Open-Source Software. *IEEE Software* 24(1), 58–64 (2007)
6. QUALOSS (2008), <http://www.qualoss.org/>
7. FlossMETRICS (2008), <http://flossmetrics.org/>
8. SQO-OSS (2008), <http://www.sqo-oss.eu/>
9. QUALIPSO (2008), <http://www.qualipso.org/>

Predicting Software Fault Proneness Model Using Neural Network

Yogesh Singh, Arvinder Kaur, and Ruchika Malhotra

University School of Information Technology, Guru Gobind Singh Indraprastha University,
Kashmere Gate, Delhi-110006, India
ys66@rediffmail.com, arvinderkaurtakkar@yahoo.com,
ruchikamalhotra2004@yahoo.com

Abstract. Importance of construction of models for predicting software quality attributes is increasing leading to usage of artificial intelligence techniques such as Artificial Neural Network (ANN). The goal of this paper is to empirically compare traditional strategies such as Logistic Regression (LR) and ANN to assess software quality. The study used data collected from public domain NASA data set. We find the effect of software metrics on fault proneness. The fault proneness models were predicted using LR regression and ANN methods. The performance of the two methods was compared by Receiver Operating Characteristic (ROC) analysis. The areas under the ROC curves are 0.78 and 0.745 for the LR and ANN model, respectively. The predicted model shows that software metrics are related to fault proneness. The models predict faulty classes with more than 70 percent accuracy. The study showed that ANN method can also be used in constructing software quality models and more similar studies should further investigate the issue. Based on these results, it is reasonable to claim that such a model could help for planning and executing testing by focusing resources on fault-prone parts of the design and code.

Keywords: empirical validation, metrics, software quality, artificial neural network.

1 Introduction

Software metrics [2, 6, 7, 11, 12, 16, 20, 22, 27-29, 30] provide ways to evaluate the quality of software and their use in earlier phases of software development can help organizations in assessing large software development quickly, at a low cost [1]. There have been empirical studies evaluating the impact of software metrics on software quality and constructing models that utilize them in predicting quality attributes of the system, such as [1, 4, 6, 8-10, 13-15, 19, 21-22, 24-25, 28-29, 31-32, 35-36, 38]. Most of these prediction models are built using statistical techniques. ANN have seen an explosion of interest over the years, and are being successfully applied across a range of problem domains, in areas as diverse as finance, medicine, engineering, geology and physics. Indeed, anywhere that there are problems of prediction, classification or control, neural networks are being introduced. ANN can be used as a predictive model because it is very sophisticated modeling technique capable of modeling

complex functions. In [25], Khoshgoftaar et al. presented a case study of real time avionics software to predict the testability of each module from static measurements of source code. They found that ANN is a promising technique for building predictive models, because they are able to model nonlinear relationships.

Thus LR and ANN approaches are inherently different, raising the question whether one approach has better performance than the other. To investigate this question, the performance of LR and ANN methods was compared in the study for predicting software fault proneness. The public domain NASA data set is used in this study to empirically evaluate the relationship of software metrics with fault proneness.

The study is divided into following parts:

- (i) Software fault proneness model is constructed using multivariate analysis to predict fault proneness of classes using LR and ANN technique.
- (ii) The performance of the models is evaluated using ROC analysis.

The paper is organized as follows: Section 2 summarizes the related work. Section 3 summarizes the metrics studied and describes sources from which data is collected. Section 4 presents the research methodology followed in this paper. The results of the study are given in section 5. The model is evaluated in section 6. Section 7 presents threats to validity of the models and conclusions of the research are presented in section 8.

2 Related Work

Khoshgoftaar et al. [25] introduced the use of the neural networks as a tool for predicting software quality. In [25], they presented a large telecommunications system, classifying modules as fault prone or not fault prone. They compared the ANN model with a non-parametric discriminant model, and found the ANN model had better predictive accuracy.

Huang et al. [24] proposed a neuro-fuzzy constructive cost model for software cost estimation. In another research, Cartwright [9] compared four prediction techniques: regression, rule induction, nearest neighbor, and neural nets. Other recent studies include using machine learning algorithms [31-32].

3 Research Background

In this section we present the summary of metrics studied in this paper (Section 3.1) and empirical data collection (Section 3.2).

3.1 Dependent and Independent Variables

The binary dependent variable in our study is fault proneness. The goal of our study is to empirically explore the relationship between software metrics and fault proneness at the class level. Fault proneness is defined as the probability of fault detection in a class. We use LR and ANN methods to predict probability of fault proneness. The metrics are summarized in Table 1.

Table 1. Metrics Studied

Metric	Source
Cyclomatic complexity	Mc Cabe [30]
Design complexity	
Lines Of Code (LOC)	
Branch count	Miscellaneous
Call pairs	
Maintenance severity	
Edge count	
Node count	
Design density	

3.2 Empirical Data Collection

This study makes use of public domain data set KC4 from the NASA Metrics Data Program. The data in KC4 was collected from a ground-based subscription server consisting of 25 KLOC of Perl source code [34]. This system consists of 126 classes and provides method-level static metrics. At the method level, 21 software product metrics based on product's complexity, size and vocabulary are given.

The metrics having constant or missing values were removed from the analysis. The metrics having less than 6 data points were also removed from the analysis.

4 Research Methodology

In this section the steps taken to analyze software metrics for classes taken for analysis are described. The procedure used to analyze the data collected for each measure is described in following stages (i) outlier analysis (ii) LR and ANN modeling (iii) model evaluation.

4.1 Outlier Analysis

Data points, which are located in an empty part of the sample space, are called outliers. Outlier analysis is done to find data points that are over influential and removing them is essential. Univariate and multivariate outliers were found in our study. To identify multivariate outlier we calculate for each data point the Mahalanobis Jack-knife distance. Details on outlier analysis can be found in [4, 23].

The input metrics were normalized using min-max normalization. Min-max normalization performs a linear transformation on the original data [17]. Suppose that \min_A and \max_A are the minimum and maximum values of an attribute A. It maps value v of A to v' in the range 0 to 1 using the formula:

$$v' = \frac{v - \min A}{\max A - \min A} \quad (1)$$

4.2 Logistic Regression (LR) Modeling

LR is the most widely used technique [1] in literature used to predict dependent variable from set of independent variables (a detailed description is given by ([1], [4] and [23])). Binary LR is used to construct models when the dependent variable is binary as in our case. In our study, the dependent variable is fault proneness and the independent variable is metrics. LR is of two types: (i) Univariate LR (ii) Multivariate LR.

Univariate LR is a statistical method that formulates a mathematical model depicting relationship among dependent variable and each independent variable. This technique is used to test hypotheses.

Multivariate LR is used to construct a prediction model for the fault-proneness of classes. In this method metrics are used in combination. The multivariate LR formula can be defined as follows:

$$prob(X_1, X_2, \dots, X_n) = \frac{e^{(A_0 + A_1 X_1 + \dots + A_n X_n)}}{1 + e^{(A_0 + A_1 X_1 + \dots + A_n X_n)}} \quad (2)$$

where $X_i, i = 1, 2, \dots, n$, are the independent variables. *Prob* is the probability of detecting faults in a class. Univariate logistic formula is a special case of multivariate LR formula and can be defined as:

$$prob(X_1, X_2, \dots, X_n) = \frac{e^{(A_0 + A_1 X)}}{1 + e^{(A_0 + A_1 X)}} \quad (3)$$

In LR two stepwise selection methods forward selection and backward elimination can be used [4]. In forward stepwise procedure, stepwise variable entry examines the variables in the block at each step for entry. The backward elimination method includes all the independent variables in the model. Variables are deleted one at a time from the model until a stopping criterion is fulfilled. We have used backward elimination method using metrics selected in univariate analysis. Details of LR method can be found in [1].

4.3 Artificial Neural Network Modeling

The network used in this work belongs to Multilayer Feed Forward networks and is referred to as M-H-Q network with M source nodes, H nodes in hidden layer and Q nodes in the output layer [36]. The input nodes are connected to every node of the hidden layer but are not directly connected to the output node. Thus the network does not have any lateral or shortcut connection.

ANN repetitively adjusts different weights so that the difference between desired output from the network and actual output from ANN is minimized. The network learns by finding a vector of connection weights that minimizes the sum of squared errors on the training data set. The summary of ANN used in this study is shown in Table 2. The ANN was trained by standard error back propagation algorithm at a learning rate of 0.005, having the minimum square error as the training stopping criterion.

The input layer has one unit for each input variable. Each input value in the data set is normalized within the interval [0, 1] using min-max normalization (see Section 4.1). Given an n by m matrix of multivariate data, Principal component analysis [26] can reduce the number of columns. We performed Principal component analysis on the input metrics to produce domain metrics [36]. In our study n represents the number of classes for which OO metrics have been collected. Using Principal component analysis, the n by m matrix is reduced to n by p matrix (where p<m).

We use one hidden layer as what can be achieved in function approximation with more than one hidden layer can also be achieved by one hidden layer [25]. There is one unit in the output layer. The output unit with value greater than a threshold (cutoff point) indicates the class selected by the network is fault prone otherwise it is not.

Table 2. ANN Summary

Architecture	
Layers	3
Input Units	4
Hidden Units	5
Output Units	1
<i>OTraining</i>	
Transfer Function	Tansig
Algorithm	Back Propagation
Training Function	TrainBR

Due to the nonlinear nature of ANN, the statistical tests for parameter significance that are used in LR cannot be applied here. Instead we used ROC analysis [18] to heuristically assess the importance of input variables for the classification result.

4.4 Evaluating the Performance of the Model

The common measures to assess the quality of predicted model in our study are:

- The sensitivity and specificity of the model is calculated to predict the correctness of the model. The percentage of classes correctly predicted to be fault prone is known as sensitivity of the model. The percentage of non-occurrences correctly predicted i.e. classes predicted not to be fault prone is called specificity of the model.
- Yourdon’s *J* coefficient [14]: *J* coefficient is defined as:

$$J = s + f - 1 \tag{4}$$

The *J* coefficient can vary from -1 to +1 with plus 1 being perfect accuracy and -1 being the worst accuracy.

- Proportion correct is defined as: It is defined as ratio of number of classes correctly classified as fault prone (and not fault prone) and total number of classes.

- Receiver Operating Characteristic (ROC) analysis: The LR model outputs and the ANN outputs were evaluated for performance using ROC analysis. ROC curve, which is defined as a plot of sensitivity on the y-coordinate versus its 1-specificity on the x coordinate, is an effective method of evaluating the quality or performance of predicted models [14]. While constructing ROC curves, one selects many cutoff points between 0 and 1 in our case, and calculates sensitivity and specificity at each cut off point. We report sensitivity and specificity of the predicted models for threshold selected by ROC (mostly taken as classifier output [14]).

Area Under the ROC Curve (AUC) is a combined measure of sensitivity and specificity [14]. In order to compute the accuracy of the predicted models, we use the area under ROC curve. The standard error for ROC curves was determined according to the method proposed by Hanley and McNeil [18].

- In order to predict accuracy of model it should be applied on different data sets. We therefore performed k-cross validation of models [37]. The data set is randomly divided into k subsets. Each time one of the k subsets is used as the test set and the other k-1 subsets are used to form a training set. Therefore, we get the fault proneness for all the k classes.

5 Analysis Results

In this section we described the analyses performed to find the relationship between software metrics and fault proneness of the classes. We first employed LR [23] method, which is widely used to predict quality models. We then employed ANN technique to predict the fault proneness of the classes. This method is rarely applied in this area.

5.1 Logistic Regression (LR) Analysis

In this subsection we find the relationship of independent variables (metrics) with dependent variable (fault proneness). Table 3 shows the Coefficient (B) and Significance (p-value) of metrics included in the model.

The model is applied to all system classes to compare predicted and actual fault proneness (or non fault proneness). A threshold of $P_0=0.5$ is chosen using ROC analysis (Table 4). Classes with predicted probability above 0.5 are classified to be fault prone and below this threshold are classified as to be not fault prone. This threshold was selected to balance the number of actual and predicted faults. Out of 61 classes actually fault prone, 47 classes were predicted to be fault prone. The sensitivity of the model is 77%. Similarly 47 out of 64 classes were predicted not to be fault prone. Thus specificity of the model is 73.4%. This shows that the model correctness is good.

Table 3. Multivariate Analysis for LR Model

Variable	Call pairs	Design density	Edge count	Constant
B	0.011	0.014	0.239	0.554
p-value	0.185	-2.350	0.009	0.531

Table 4. Predicted Correctness of LR Model

Observed	Predicted		
	0.00	1.00	Percent Correct
0.00	47	17	73.4%
1.00	14	47	77%

Table 5. Predicted Correctness of ANN Model

Observed	Predicted		
	.00	1.00	Percent Correct
.00	42	22	65.4%
1.00	11	50	80.3%

5.2 Artificial Neural Network (ANN) Method

The results of the model predicted are shown in Table 5. Out of 61 classes actually fault prone, 50 classes were predicted to be fault prone (Table 5). The sensitivity of the model is 80.3%. Similarly 42 out of 64 classes were predicted not to be fault prone. Thus specificity of the model is 65.4%. This shows that the sensitivity of the model is high as compared to model predicted using LR approach but specificity is slighter less as compared to LR model.

6 Model Evaluation

In this section we present the results of cross validation of LR and ANN models and also perform ROC analysis to compare these approaches.

6.1 Cross Validation of Models Using ROC Analysis

The accuracy of models predicted is somewhat optimistic since the models are applied on same data set from which they are derived from. To predict accuracy of model it should be applied on different data sets thus we performed 10-cross validation of LR and ANN models following the procedure given in Section 4. For the 10-cross validation, the classes were randomly divided into 10 equal parts of approximately. We summarized the results of cross validation of predicted models via the LR and ANN approaches in Table 6.

In Figure 1, the ROC curves for LR and ANN models are presented. The ROC curve for the LR model is shown in Figure 1(a), AUC was 0.78 (SE 0.040), providing 75.4% of sensitivity and 71.8% of specificity. Whereas, the area under the ROC curve for ANN model was 0.745 (SE 0.044), with sensitivity 75.4% and specificity 65.4%.

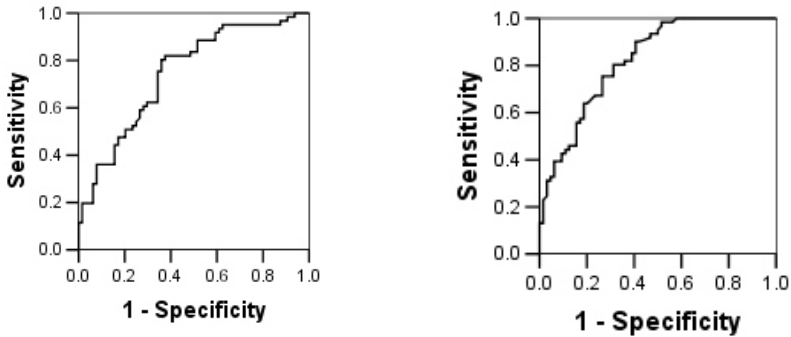


Fig. 1. ROC curve for (a) LR and (b) ANN models

As shown in Table 6, the results of cross validation of ANN model were almost similar as compared to cross validation results of LR model.

Table 6. Results of 10-cross validation of Models

Technique	Sensitivity	Specificity	Proportion correct	J Coefficient	AUC
LR	75.4	71.8	73.8	0.48	0.78 (SE 0.040)
ANN	75.4	65.4	73.8	0.49	0.745 (SE 0.044)

7 Threats to Validity

The study has a number of limitations that are not unique to our study but are common with most of the empirical studies in the literature. However, it is necessary to repeat them here. The degree to which the results of our study can be generalized to other research settings is questionable. The reason is that the systems developed are medium-sized. In this study severity of faults is not taken into account. There can be number of faults which can leave the system in various states e.g. a failure that is caused by a fault may lead to a system crash or an inability to open a file. The former failure is more severe than latter, although the types of fault are not the same.

Though these results provide guidance for future research on the use of LR and ANN methods to find the impact of software metrics on fault proneness, further validations are necessary with different systems to draw stronger conclusions.

8 Conclusions

We conducted an empirical analysis of the software metrics. The main goal of our study was to examine and compare LR and ANN methods in order to find the impact

of software metrics on fault proneness. Thus we employed LR and ANN methods to assess the applicability of the software metrics to predict fault proneness. This is the primary contribution of our study. The performance of the fault proneness models were evaluated using ROC analysis, since few studies have used this method in past.

The AUC for LR model was 0.78 (SE 0.040), providing of sensitivity 75.4% and 71.4% of specificity. The AUC for ANN model was 0.745 (SE 0.044), with sensitivity 75.4% and specificity 65.4%. The models predicted using both LR and ANN method yielded good AUC using ROC analysis. This study confirms that construction of ANN is feasible, adaptable to systems, and useful in predicting fault prone classes.

While research continues, practitioners and researchers may apply ANN method for constructing models to predict faulty classes.

As in all empirical studies the relationship we established is valid only for certain population of systems. In this case, we can roughly characterize this population as “medium-sized systems.”

More similar type of studies must be carried out with large data sets to get an accurate measure of performance outside the development population. We further plan to replicate our study to predict models based on other artificial intelligence techniques.

References

1. Aggarwal, K.K., Singh, Y., Kaur, A., Malhotra, R.: Investigating the Effect of Coupling Metrics on Fault Proneness in Object-Oriented Systems. *Software Quality Professional* 8(4), 4–16 (2006)
2. Aggarwal, K.K., Singh, Y., Kaur, A., Malhotra, R.: Software Reuse Metrics for Object-Oriented Systems. In: *Third ACIS Int'l Conference on Software Engineering Research, Management and Applications (SERA 2005)*, pp. 48–55. IEEE Computer Society, Los Alamitos (2005)
3. Barnett, V., Price, T.: *Outliers in Statistical Data*. John Wiley & Sons, Chichester (1995)
4. Basili, V., Briand, L., Melo, W.: A Validation of Object-Oriented Design Metrics as Quality Indicators. *IEEE Transactions on Software Engineering* 22(10), 751–761 (1996)
5. Belsley, D., Kuh, E., Welsch, R.: *Regression Diagnostics: Identifying Influential Data and Sources of Collinearity*. John Wiley & Sons, Chichester (1980)
6. Briand, L., Daly, W., Wust, J.: Unified Framework for Cohesion Measurement in Object-Oriented Systems. *Empirical Software Engineering* 3, 65–117 (1998)
7. Briand, L., Daly, W., Wust, J.: A Unified Framework for Coupling Measurement in Object-Oriented Systems. *IEEE Transactions on software Engineering* 25, 91–121 (1999)
8. Briand, L., Daly, W., Wust, J.: Exploring the relationships between design measures and software quality. *Journal of Systems and Software* 5, 245–273 (2000)
9. Cartwright, M., Kadoda, G.: Comparing software prediction techniques using simulation. *IEEE Transactions of Software Engineering* 27(1), 1014–1022 (2001)
10. Cartwright, M., Shepperd, M.: An Empirical Investigation of an Object-Oriented Software System. *IEEE Transactions of Software Engineering* 26(8), 786–796 (1999)
11. Chidamber, S., Kemerer, C.: A metrics Suite for Object-Oriented Design. *IEEE Trans. Software Engineering* SE-20(6), 476–493 (1994)
12. Chidamber, S., Kemerer, C.: Towards a Metrics Suite for Object Oriented design. In: *Proc. Conference on Object-Oriented Programming: Systems, Languages and Applications (OOPSLA 1991)*. Published in *SIGPLAN Notices*, vol. 26(11), pp. 197–211 (1991)

13. Chidamber, S., Darcy, D., Kemerer, C.: Managerial use of Metrics for Object-Oriented Software: An Exploratory Analysis. *IEEE Transactions on Software Engineering* 24(8), 629–639 (1998)
14. El Emam, K., Benlarbi, S., Goel, N., Rai, S.: The Confounding Effect of Class Size on the Validity of Object-Oriented Metrics. *IEEE Transactions on Software Engineering* 27(7), 630–650 (2001)
15. Gyimothy, T., Ferenc, R., Siket, I.: Empirical validation of object-oriented metrics on open source software for fault prediction. *IEEE Trans. Software Engineering* 31(10), 897–910 (2005)
16. Halstead, M.H.: *Elements of Software Science*. North Holland, New York (1997)
17. Han, J., Kamber, M.: *Data Mining: Concepts and Techniques*. Harchort India Private Limited (2001)
18. Hanley, J., McNeil, B.: The meaning and use of the area under a Receiver Operating Characteristic ROC curve. *Radiology* 143, 29–36 (1982)
19. Harrison, R., Counsell, S.J., Nithi, R.V.: An Evaluation of MOOD set of Object-Oriented Software Metrics. *IEEE Trans SE-24*(6), 491–496 (1998)
20. Henderson-Sellers, B.: *Object-Oriented Metrics, Measures of Complexity*. Prentice-Hall, Englewood Cliffs (1996)
21. Henry, S., Kafura, D.: Software structure metrics based on information flow. *IEEE Transactions on Software Engineering SE* 7(5), 510–518 (1981)
22. Hitz, M., Montazeri, B.: Measuring Coupling and Cohesion in Object-Oriented Systems. In: *Proc. Int. Symposium on Applied Corporate Computing*, Monterrey, Mexico (1995)
23. Hosmer, D., Lemeshow, S.: *Applied Logistic regression*. John Wiley and Sons, Chichester (1989)
24. Huang, X., Capretz, L.F., Ren, J., Ho, D.: A neuro-fuzzy model for software cost estimation. In: *International Conference on Quality Software*, p. 126 (2003)
25. Khoshgafaar, T.M., Allen, E.D., Hudepohl, J.P., Aud, S.J.: Application of neural networks to software quality modeling of a very large telecommunications system. *IEEE Transactions on Neural Networks* 8(4), 902–990 (1997)
26. Kothari, C.R.: *Research Methodology. Methods and Techniques*, New Age International Limited (2004)
27. Lorenz, M., Kidd, J.: *Object-Oriented Software Metrics*. Prentice-Hall, Englewood Cliffs (1994)
28. Lee, Y., Liang, B., Wu, S., Wang, F.: Measuring the Coupling and Cohesion of an Object-Oriented program based on Information flow (1995)
29. Li, W., Henry, S.: Object-Oriented Metrics that Predict Maintainability. *Journal of Systems and Software* 23(2), 111–122 (1993)
30. McCabe, T.J.: A Complexity Measure. *IEEE Transactions on Software Engineering SE* 2(4), 308–320 (1976)
31. Menzies, T., DiStefano, J., Orrego, A., Chapman, R.: Assessing Predictors of Software Defects. In: *Proc. Workshop Predictive Software Models* (2004)
32. Menzies, T.: Data Mining Static Code Attributes to Learn Defect Predictors. *IEEE Transactions on Software Engineering* 32(11), 771–784 (2006)
33. Myers, G.J.: *Composite/Structured Design*, Von Nostrand, Reinhold, New York (1978)
34. ASA/WVU IV&V Facility, Metrics Data Program, <http://mdp.ivv.nasa.gov>
35. Olague, H., Etkorn, L., Gholston, S., Quattlebaum, S.: Empirical Validation of Three Software Metrics Suites to Predict Fault-Proneness of Object-Oriented Classes Developed Using Highly Iterative or Agile Software Development Processes. *IEEE Transactions on software Engineering* 33(8), 402–419 (2007)

36. Singh, Y., Kaur, A., Malhotra, R.: Application of Logistic Regression and Artificial Neural Network for Predicting Software Quality Models. In: International Conference on Software Engineering Research and Practice (SERP 2007), Las Vegas, USA, June 25-26 (2007)
37. Stone, M.: Cross-validatory choice and assessment of statistical predictions. *J. Royal Stat. Soc.* 36, 111–147 (1974)
38. Yuming, Z., Hareton, L.: Empirical analysis of Object-Oriented Design Metrics for predicting high severity faults. *IEEE Transactions on Software Engineering* 32(10), 771–784 (2006)

Automating the Measurement of Functional Size of Conceptual Models in an MDA Environment*

Beatriz Marín, Oscar Pastor, and Giovanni Giachetti

Department of Information Systems and Computation,
Technical University of Valencia,
Camino de Vera s/n,
46022 Valencia, Spain
{bmarin, opastor, ggiachetti }@dsic.upv.es

Abstract. The manual measurement of functional size is generally very time-consuming and has many precision errors. For this reason, it is necessary to automate the measurement process to obtain a solution that can be applied in a MDA industrial development. The OO-Method COSMIC Function Points (OOmCFP) is a measurement procedure that has been designed to measure the functional size of object-oriented applications generated from their conceptual models by means of model transformations. This work presents the definition of the mechanisms that are necessary to automate the OOmCFP procedure. This work also presents the OOmCFP tool that implements the OOmCFP procedure. Since this tool measures the functional size of industrial applications generated in MDA environments from their conceptual models, it is not necessary to perform the measurement task on the final code. The OOmCFP tool incorporates the benefits that the COSMIC measurement method provides. These benefits are demonstrated through a comparative analysis.

Keywords: Conceptual modeling, Object orientation, Functional size measurement, COSMIC, MDA, Tool.

1 Introduction

The Model-Driven Architecture (MDA) approach [17] separates application and business logic from the platform technology, allowing code generation by means of model transformations. In MDA contexts, conceptual models are used as input to the process of code generation. Thus, the conceptual models must have enough semantic formalization in order to specify all the functionality of the final application and also to avoid different interpretations for the same model.

The OO-Method approach [18] [20] is an object-oriented method that provides the required semantic formalization to define complete and unambiguous conceptual models, allowing the automatic generation of software products [19] using an

* This work has been developed with the support of MEC under the project SESAMO TIN2007-62894 and co financed by FEDER.

MDA-based technology. This method has been implemented in a suite of industrial tools by CARE Technologies [5].

The adoption of MDA-based technology has presented new challenges, such as measuring the size of the products that are generated from their conceptual models. This is important because the size of the conceptual model allows that the cost of the application that is automatically generated will be estimated correctly. The Function Point Analysis (FPA) proposal ([9] [10]), together with its adaptations of this measurement method [1] [2] [15] [23] [24], is used to do this. However, these FPA-based approaches have limitations for the measurement of conceptual models used in MDA environments [4] [8] [14]. For instance, FPA-based approaches only allow the measurement of the functionalities from the viewpoint of the human user, ignoring all the functionality that the human user does not see, which should be built for the correct operation of the application.

To overcome the limitations of the initial design of the FPA measurement method, the COSMIC measurement method was defined [3] [13]. COSMIC allows the measurement from different points of view: from the human user viewpoint (like FPA); from the developer viewpoint (including all the functionalities that should be built); and from the viewpoint of any user of the conceptual model. Currently, there are some approaches that apply COSMIC for the purpose of estimating the functional size of future software applications from conceptual models, such as Poels' proposal [22] and Diab's proposal [7]. Both of these FSM procedures were defined establishing a mapping between the COSMIC concepts and their primitives; however their proposals are not compliant with MDA principles.

For industrial MDA development, it is essential to do the measurements quickly and in a precise way because the functional size determines the cost of the generated applications. Therefore, a tool that allows the automatic measurement of conceptual models used in MDA environments is needed to avoid the excessive time and the precision errors involved in a manual measurement process.

This paper introduces the OOmCFP proposal from a practical perspective. This is a procedure to measure the functional size of OO-Method conceptual models based on COSMIC, focusing on the OOmCFP tool, which automates the measurement of OO-Method conceptual models through the implementation of the OOmCFP proposal. The OOmCFP tool includes all the benefits that are related to the COSMIC approach. It allows a better measurement of the conceptual models involved in the OO-Method MDA industrial approach and makes the practical application of the OOmCFP approach possible.

The rest of the paper is organized as follows: section 2 and section 3 present the main concepts of the COSMIC method and the OO-Method approach, respectively. Section 4 presents the OOmCFP measurement procedure and an example of the measurement of an OO-Method conceptual model using the OOmCFP proposal. Section 5 presents the tool that automates the OOmCFP proposal and a comparative analysis of the results obtained in the measurement of conceptual models of real applications. Finally, section 6 presents a discussion on the results achieved as well as suggestions for further work.

2 The COSMIC Functional Size Measurement Method

The COSMIC functional size measurement method can be used to measure any type of software. The application of this measurement method includes three phases: the measurement strategy, the mapping of concepts, and the measurement of the identified concepts.

In the *measurement strategy phase*, the *purpose* and the *scope* of the measurement exercise must be defined. Next, the *functional users*, which are types of users that send (or receive) data to (from) the functional process of the application to be measured must be identified. Finally, the *level of granularity* of the description of the piece of software to be measured is also identified.

In the *mapping phase*, the *functional processes* (the elementary components of a set of functional user requirements) must be identified. Next, the *data groups* must be identified. A data group is a set of data attributes that are distinct, non empty, non ordered, non redundant, and that participate in a functional process. The identification of the *data attributes* of a data group is optional.

In the *measurement phase*, the *data movements* (Entry, Exit, Read and Write) for every functional process must be identified. When all the data movements of the functional process are identified, the measurement function must be applied: this is a mathematical function that assigns 1 CFP to each data movement of the functional process. Then, after all the functional processes are measured, the measurement results are aggregated to obtain the functional size of the piece of software that has been measured.

Figure 1 shows the COSMIC metamodel, which illustrates the information that should be represented by the software artefact to be measured.

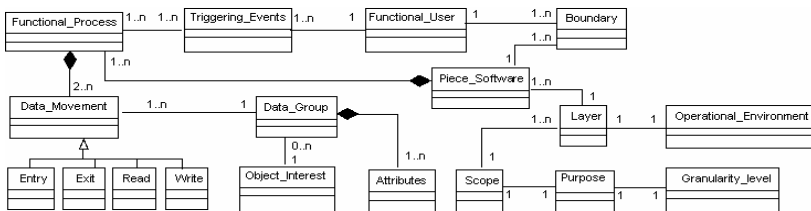


Fig. 1. Metamodel of COSMIC.

3 The OO-Method Approach

OO-Method is a method that allows the automatic generation of software from conceptual models. It has a formal definition supported by OASIS [21], which is an object-oriented, formal specification language for Information Systems. This method is supported by the compiler of OO-Method conceptual models that is implemented in the OlivaNova Suite [5].

The OO-Method model compiler generates applications according to a three-tier software architecture: a tier for the client component, which contains the graphical user interface-related software components; a tier for the server component, which

contains the business rules and the connections to the database; and a tier for the database component, which contains the persistence aspects of the applications.

The software production process in OO-Method is represented by three models:

- The Requirements Model, which specifies the system requirements using a set of techniques such as the Mission Statement, the Functions Refinement Tree, the Use Case Model, and the Sequence Diagrams Model.
- The Conceptual Model, which captures the static and dynamic properties of the functional requirements of the system by means of an Object Model, a Dynamic Model, and a Functional Model. The conceptual model also allows the specification of the user interfaces in an abstract way through the Presentation Model. With all of these models, the conceptual model has all the details needed for the automatic generation of the software application. The complete definition of the elements of the conceptual model of OO-Method is described in detail in [19].
- The Execution Model, which allows the transition from the problem space (represented by the conceptual model) to the solution space (the corresponding software product). This model fixes the mappings between conceptual primitives and their corresponding software representations in a target software development environment.

For the purpose of this work, we only need to focus on the Conceptual Model, which is the artifact from which we want to measure the functional size through the corresponding measurement process.

4 OOmCFP: A Measurement Procedure for the OO-Method Conceptual Model

OOmCFP (OO-Method COSMIC Function Points) is a measurement procedure that was developed for measuring the functional size of the OO-Method applications that are based on the MDA approach [16]. In the OOmCFP procedure, the *entity* to be measured is an OO-Method conceptual model, and the *attribute* to be measured is the functional size, which is defined by the ISO/IEC 14143-1 standard as the size of software derived by quantifying the functional user requirements [12].

The OOmCFP was defined in accordance with the COSMIC measurement manual version 3.0 [3]. We selected this functional size method for the design of OOmCFP for the simplicity with which it quantifies functional size without being limited by maximum values, as occurs in other standards (IFPG FPA, NESMA FPA or MARK II FPA). Given that the OOmCFP procedure was designed in accordance with COSMIC, a mapping between the concepts used in COSMIC and the concepts used in the OO-Method conceptual model has been defined (Table 1). It is important to note that the mapping has been done in only one direction since only some of the elements of the conceptual model are relevant to the measurement of the functional size when COSMIC is used.

Table 1. Results obtained from the mapping between COSMIC and OO-Method

OOmCFP

Purpose: To Measure the functional size of the OO-Method conceptual models to estimate the cost of the applications specifically generated by the OlivaNova Suite.

Scope: The OO-Method conceptual model, which has all the functionality details from which the final software application will be built.

Granularity Level: Low level, since all the details in the OO-Method conceptual model are needed to generate the applications.

Layers: The Client component, the Server component, and the Database component of an OO-Method application since each component is generated for a specific software environment.

Pieces of Software: The Client component, the Server component, and the Database component of an OO-Method application since every layer has at least one piece of software.

Functional Users:

- Human users are functional users of the client component of an OO-Method application since data is sent (or receive) to (from) this component.

- The Client component of an OO-Method application is a functional user of the Server component of the application. This user is called *client functional user*.

- The Server component of an OO-Method application is a functional user for both the Client component and for the Database component of the OO-Method application. This user is called *server functional user*.

Boundaries: The OO-Method applications have three boundaries that separates the users from the layers: one boundary between the human user and the Client component; one boundary between the client functional user and the Server component; and one boundary between the server functional user and the Database component – see Figure 2.

Triggering Events:

- The human functional user carries out triggering events that occur in the real world.

- The client functional user carries out triggering events that occur in the interaction units of the presentation model of the OO-Method conceptual model.

- The server functional user carries out the triggering events that occur in the server component of the software.

Functional Processes: Direct successors of the menu of the presentation model of OO-Method conceptual model. Every child represents a single functional process, either a selection of a given class population (a Population Interaction Unit (PIU)) or an execution of a service (a Service Interaction Unit (SIU)). These interaction units can be combined into more complex interaction units (as a Master Detail Interaction Unit (MDIU)).

Data Movements: The data movements that can occur in the OO-Method applications are shown in Figure 2. Note that the *write* and *read* data movements only can occur between the server functional user and the database component of an OO-Method application.

Data Groups: The classes of the object model of the OO-Method conceptual model, which are used in the functional process.

Data Attributes: The set of attributes of each class that is identified as a Data Group.

Once the mapping between COSMIC and OO-Method has been defined, the measurement rules of the OOmCFP must also be defined. These rules are the rules that assign a numerical value to the data movements that take place between the functional users and the software components of an OO-Method application. The data movements that can occur in the OO-Method applications are shown in Figure 2.

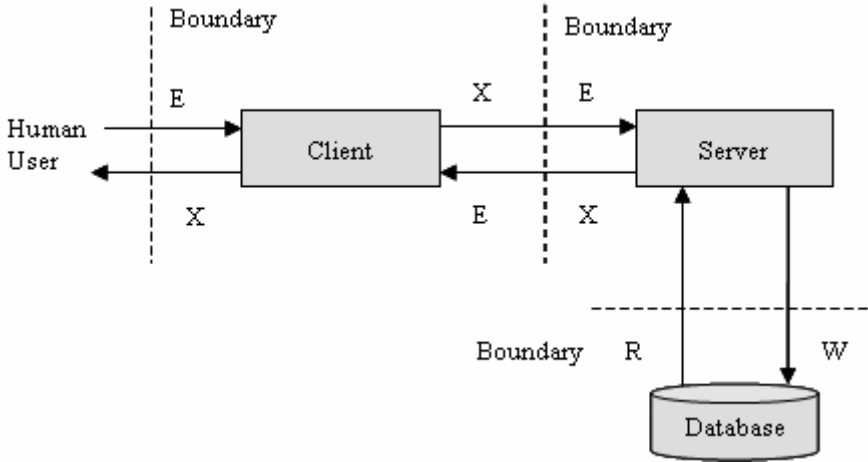


Fig. 2. Data movements that can occur in the functional processes of an OO-Method application

Given that the applications generated from the OO-Method conceptual model has a three-tier architecture, three types of entry (E) data movements can occur in a functional process: from the human user to the client component of the application; from the client component of the application to the server component of the application; and from the server component of the application to the client component of the application (see Figure 2). A set of measurement rules has been defined for each type of entry data movements.

Three types of exit (X) data movements can occur in a functional process: from the client component of the application to the human user; from the client component of the application to the server component of the application; and from the server component of the application to the client component of the application. Figure 2 shows the exit data movements. A set of measurement rules has been defined for each type of exit data movements.

Only one type of read (R) data movements can occur in OO-Method applications: only the server component of the software can read the persistence storage (see Figure 2). A set of measurement rules has been defined for this type of data movements.

Only one type of write (W) data movements can occur in OO-Method applications: only the sever component of the software can write to the persistence storage (Figure 2). A set of measurement rules for the write data movements has been defined.

According to the COSMIC functional size measurement method, each data movement will be assigned one size unit, which is referred to as 1 CFP. To measure the functional size of a functional process, the functional size of all the data movements of the functional process should be added – see formula (1).

$$\text{SizeFunctionalProcess} = \sum_{i=1}^n \text{DataMovement}_i \quad (1)$$

Once all the functional processes are measured with formula (1), then all the measurements should be added to obtain the functional size of the layer that contains these functional processes – see formula (2).

$$\text{SizeLayer} = \sum_{i=1}^n \text{SizeFunctionalProcess}_i \quad (2)$$

To measure of the generated software applications from the developer’s viewpoint, it is necessary to add the functional size of every layer. This calculation is represented in formula (3).

$$\text{SizeOOMethodApplication} = \sum_{i=1}^n \text{SizeLayer}_i \quad (3)$$

Finally, with the three formulas, it is possible to measure the functional size of the OO-Method software applications that are generated from their conceptual model in an MDA environment. The measurement rules include all the functionalities needed by the application for its correct operation; in other words, it includes all the functionalities from the developer’s viewpoint.

In terms of the validation of the OOmCFP procedure, since the validation of COSMIC (from the perspective of the measurement theory) has been carried out successfully using the DISTANCE framework [6], the theoretical validation of the OOmCFP procedure can be inferred. Moreover, an expert has validated the conformity of the OOmCFP procedure with the COSMIC version 3.0.

4.1 A Measurement Example

Figure 3 shows an example of an OO-Method conceptual model that allows the automatic generation of a fully working application. This application allows the creation and deletion of invoices with their details, and also allows the creation of the customers associated to the invoice. The administrator of the application, which is represented by the class Admin, can execute the services of the application.

The populations: PIU_Admin, PIU_Customer, PIU_Invoice¹, and the master detail MDIU_Invoice are identified as functional processes by applying the mapping rules presented in Table 1.

¹ PIU is the OO-Method acronym for “Population Interaction Unit”. A PIU represents an entry-point for the application, through the presentation of a set of instances of a class. An instance can be selected, and the corresponding set of actions and/or navigations specified in the Presentation Model are offered to the user. More details can be found in [19].

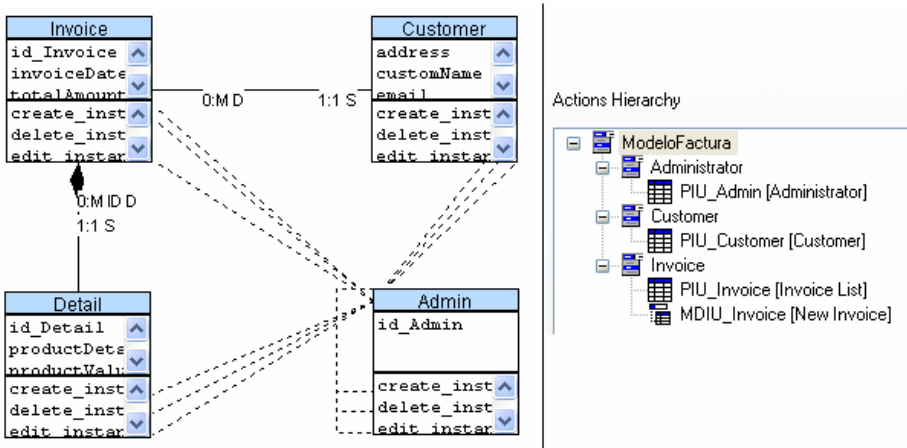


Fig. 3. Example of an OO-Method conceptual model. Left: Object model. Right: Presentation Model

After identifying the functional processes, the OOmCFP measurement rules are applied to identify the data movements that occur in each functional process. The measurement rules applied to the example are presented in the following table.

Table 2. Measurement rules of OOmCFP applied to the example

Component	Measurement Rule
Client	Rule 3. 1 entry data movement for each different class that corresponds to an argument of a SIU that participates in a functional process.
Client	Rule 10. 1 entry data movement for each different class that contributes with attributes to the display set of a PIU or IIU that participates in a functional process.
Client	Rule 15. 1 exit data movement for all the attributes that are shown in a display set of a PIU or IIU that participates in a functional process.
Client	Rule 22. 1 exit data movement for the set of data-valued arguments of a SIU that participates in a functional process.
Client	Rule 23. 1 exit data movement for each different class that corresponds to an argument of a SIU that participates in a functional process.
Server	Rule 7. 1 entry data movement for the set of data-valued arguments of a SIU that participates in a functional process.
Server	Rule 8. 1 entry data movement for each different class that corresponds to an argument of a SIU that participates in a functional process.
Server	Rule 25. 1 exit data movement for each different class that contributes with attributes to the display set of a PIU or IIU that participates in a functional process.
Server	Rule 30. 1 read data movement for each different class that contributes with attributes to the display set of a PIU or IIU that participates in a functional process.
Server	Rule 31. 1 read data movement for each different class that is used in the derivation formula of the derived attributes of the display set of a PIU or IIU that participates in a functional process.
Server	Rule 35. 1 read data movement for each different class that is used in the default value formula of an object-valued argument of a service that is related to a SIU that participates in a functional process.

Table 2. (continued)

Component	Measurement Rule
Server	Rule 36. 1 read data movement for each different class that is used in the valuation formula of the event that is related to a SIU that participates in a functional process.
Server	Rule 38. 1 read data movement for each different class that is used in the formula of the transaction, the operation or the global service that is related to a SIU that participates in a functional process.
Server	Rule 50. 1 write data movement for the class that contains a destroy event or transaction that is related to a SIU that participates in a functional process.
Server	Rule 51. 1 write data movement for the class that contains a creation event or transaction that is related to a SIU that participates in a functional process.
Server	Rule 52. 1 write data movement for the class that contains an event that has valuations and that is related to a SIU that participates in a functional process.

With the measurements rules presented above, we have identified the data movements that occur in the functional processes. The following table shows the data movements that are identified for each functional process in the client component and in the server component of the OO-Method application.

Table 3. Data movements of the functional processes that occur in the client component and in the server component of the OO-Method application

Functional Process	Client Component				Server component			
	Entry	Exit	Read	Write	Entry	Exit	Read	Write
PIU_Admin	8	8	0	0	6	2	2	4
PIU_Customer	8	8	0	0	6	2	2	5
PIU_Invoice	2	1	0	0	0	2	3	0
MDIU_Invoice	8	11	0	0	8	2	4	6

Once all of the data movements are identified, the formulas defined in OOmCFP are applied to obtain the functional size for each functional process. Thus, Table 4 presents the functional size of the functional process in the client component of the software and in the server component of the OO-Method application.

Table 4. Data movements of the functional processes that occur in the software components of the application

Functional Process	Client Component	Server Component
PIU_Admin	16	14
PIU_Customer	16	15
PIU_Invoice	3	5
MDIU_Invoice	19	20

Next, by applying formula (2), we can obtain the functional size of each piece of software: the functional size of the client component of the application is 54 cfp; and the functional size of the server component of the application is 54 cfp.

Finally, we obtain the functional size of the OO-Method application by applying formula (3). The resultant functional size is 108 cfp.

It is important to note that the manual measurement of this small example took 70 minutes. Keeping in mind that the model has only four classes, the manual measurement of real applications that may contain 100 or more classes would require at least 116 hours. Therefore, it is very important to automate the measurement procedure to be able to measure efficiently conceptual models of real applications. By automating the measurement procedure many possible human errors could be avoided. The next section presents the development of the tool that automates the OOmCFP procedure.

5 The Automation of the OOmCFP Procedure

Since the automation of the measurement of conceptual models with the OOmCFP proposal is essential, a tool must be developed to implement the measurement rules defined in OOmCFP and to aggregate the results according to the formulas presented in Section 4. The OOmCFP tool has been developed using Visual Studio .Net 2003 with the language C#.

This tool must have a flexible architecture that allows adaptation to the evolution of conceptual models. It must also be agile in the measurement process.

To provide this flexible architecture, the OOmCFP tool was developed with a set of layers that allows easy incorporation of new measurement rules or changes in the existing measurement rules.

The first layer of the OOmCFP tool consists of the pre-charge of the OO-Method conceptual model that is generated from the Olivanova Suite [5] in an XML file. In this layer, the functional elements are organized in a hierarchical way, according to the functional processes identified for the client component and the server component.

In the second layer of the OOmCFP tool, each element that participates in each functional process is identified, and the measurement of the data movements that occur in each functional process are performed through the rules defined in OOmCFP. To reduce the coupling of the measurement of the elements, each rule is grouped by element and is implemented in an independent way. The result of the analysis of each element is stored in the same element.

The third layer of the OOmCFP tool consists of the aggregation of the values of each element according to the formulas defined in the OOmCFP. Thus, the tool obtains the functional size of each functional process, the functional size of each component of the application, and the functional size of the complete application.

The last layer consists of the generation of a final measurement report of the measurement in an XML file. This XML file can be transformed into other formats using XSLT. By default, the OOmCFP tool transforms the XML file in an HTML page.

Since the longest processing time and run time occur in the identification and measurement step of functional elements, we have implemented a cache mechanism to provide agility to the counting process. This reduces the high amount of time required to analyze elements that have already been analyzed. Thus, when a new functional element is identified, the cache mechanism verifies whether or not it already exists. If so, the value of the measurement is recovered.

To avoid overflow, the related elements are stored in an auxiliary array (Figure 4). Once the analysis of the first element is finished, the analysis of the elements stored in the auxiliary array continues sequentially. If the related elements are also related to other elements, these elements are added at the end of the auxiliary array, eliminating the loop of iterations and avoiding overflow.

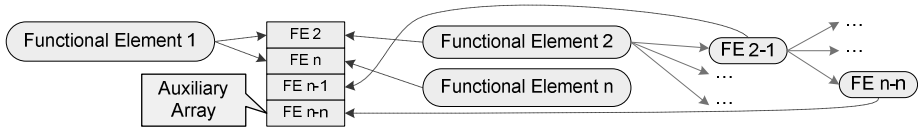


Fig. 4. Schema of the solution to avoid overflow problems

The architecture of the OOmCFP tool provides an efficient measurement process. Therefore, the measurement of conceptual models that generate real applications is done in a few seconds, thus expediting the process of estimating the cost of the final application.

The precision of the measurement is defined as the closeness of agreement between quantity values obtained by replicated measurements of a quantity under specified conditions [11]. In general, it is not possible to ensure precision in manual measurements since people can make mistakes in the identification of the functional process, the application of the measurement rules, or even the application of the formulas. In contrast, when a tool performs the measurement, it can ensure the precision of the measures because it is an automated measurement where a precise procedure will always produce the same result in any measurement task. Consequently, the OOmCFP tool avoids the errors of the manual measurements and assures the precision of the measurements.

5.1 Using the OOmCFP Tool

The steps for using the OOmCFP tool are the following:

The first step is to load the XML file that contains the OO-Method conceptual model and to specify the path where the report will be saved.

The second step is to show a summary of the model that will be measured and the path of the report.

The third step is to show the number of functional processes that have been measured and the function points for every layer of the application. In this step, the report with all the results of the measurement is saved in the path indicated in the first step. Figure 5 shows the report generated by the OOmCFP tool.

We have verified how the OOmCFP tool works in practice using some predefined OO-Method conceptual models.

5.2 A Comparative Analysis of COSMIC and FPA

A preliminary comparative analysis has been carried out with respect to the functional size of five conceptual models used in real applications. These conceptual models belong to the Management Information System domain.

OO-Method COSMIC Function Points Count Results

Data obtained from a model produced on 24/10/2007

For the application: [AgenciaFotograficaAdmin20071024](#) and for the following View: [V_View](#)

SUMMARY

Total COSMIC Function Points Count for the application: 1309

Total COSMIC Function Points Count for the client component:760

Total COSMIC Function Points Count for the server component:549

Client Component						Server Component					
Total of Functional Process: 19 Function Points Count:760						Total of Functional Process: 19 Function Points Count:549					
Name	Nº of Entry	Nº of Exit	Nº of Read	Nº of Write	Total Function Points	Name	Nº of Entry	Nº of Exit	Nº of Read	Nº of Write	Total Function Points
PIU_Admin	4	4	0	0	8	PIU_Admin	3	1	1	2	7
PIU_Editorial	7	17	0	0	24	PIU_Editorial	4	4	4	3	15
PIU_Solicitud	11	24	0	0	35	PIU_Solicitud	7	4	5	4	20
PIU_Nivel	8	27	0	0	35	PIU_Nivel	6	3	3	4	16
PIU_Fotografo	26	54	0	0	80	PIU_Fotografo	17	11	13	8	49
PIU_FotografoxNivel	27	55	0	0	82	PIU_FotografoxNivel	18	11	13	8	50
PIU_FotografoxNivelObj	27	55	0	0	82	PIU_FotografoxNivelObj	18	11	13	8	50
PIU_SolicitudReportaje	7	7	0	0	14	PIU_SolicitudReportaje	5	2	3	1	11
PIU_Tema	5	11	0	0	16	PIU_Tema	4	2	2	3	11
PIU_Reportaje	14	27	0	0	41	PIU_Reportaje	10	4	6	1	21
PIU_Exclusiva	41	84	0	0	125	PIU_Exclusiva	26	19	20	12	77
PIU_ExcluProceso	3	4	0	0	7	PIU_ExcluProceso	2	2	3	0	7
PIU_ExcluEntregada	13	27	0	0	40	PIU_ExcluEntregada	8	6	6	4	24
MDIU_Albaran	17	17	0	0	34	MDIU_Albaran	9	8	23	1	41
MDIU_AlbaranExclusiva	17	17	0	0	34	MDIU_AlbaranExclusiva	9	8	23	1	41
MDIU_Factura	28	27	0	0	55	MDIU_Factura	15	12	47	2	76
PIU_UsuarioDepCom	5	11	0	0	16	PIU_UsuarioDepCom	4	2	2	3	11
PIU_UsuarioDepProd	5	11	0	0	16	PIU_UsuarioDepProd	4	2	2	3	11
PIU_UsuarioDepTec	5	11	0	0	16	PIU_UsuarioDepTec	4	2	2	3	11

Fig. 5. Report generated by the OOmCFP tool

The analysis compares the functional sizes obtained for the OO-Method conceptual models using the COSMIC and FPA techniques. OOmCFP is used as the COSMIC measurement procedure, and OOmFP [1] is used as the FPA measurement procedure.

Before the analysis was carried out, we formulated the following hypothesis:

H1: The functional size of the measurement of an OO-Method conceptual model using a COSMIC-based approach is bigger than the measurement using a FPA-based approach.

Table 5 shows the results obtained. The *Model* column shows an identifier for each model; the *Classes* column shows the number of classes associated to each model; the *OOmCFP* column shows the number of function points obtained with the OOmCFP procedure; the *OOmFP* column shows the number of function points obtained with the OOmFP procedure; and the last column *Dif* shows the difference between the results obtained with the OOmCFP procedure and those obtained with the OOmFP procedure.

Table 5. Functional size of OO-Method conceptual models measured with the OOmCFP approach and the OOmFP approach

Model	Classes	OOmCFP	OOmFP	Dif
M1	9	836	463	373
M2	17	357	308	49
M3	30	2019	1158	861
M4	83	4326	2811	1515
M5	193	14649	6267	8382

As Table 5 shows, the functional size obtained was bigger when the OOmCFP approach was used. Therefore, as expected, hypothesis H1 is true since more aspects are taken into account when COSMIC is used as the measurement strategy. This demonstrates that when COSMIC is used, a better measurement of the functionality generated from the conceptual models in MDA environments is obtained.

Table 5 also shows that the differences obtained in the comparative analysis do not follow a common pattern, because the measurement approaches analyzed use different conceptual elements to quantify the functional size of the applications.

An important final consideration is that the complexity of the conceptual model is not measured by either the COSMIC approach or by the IFPUG approach. However, we believe that the conceptual elements that COSMIC uses in the measurement of the functional size can be used to define metrics to measure the complexity of the conceptual models.

6 Conclusions and Further Work

In this paper, we have introduced OOmCFP, which is an FSM procedure for object-oriented applications generated in MDA environments. OOmCFP allows the measurement of the functional size in the conceptual models that will be transformed in the generated applications. Therefore, we consider that OOmCFP specifies the functional requirements for the development of a tool to automate the measurement of the functional size of applications generated in MDA environments.

We have presented the OOmCFP tool, which automates the OOmCFP procedure. To develop the OOmCFP tool, a set of aspects has been taken into account. These aspects are related to the performance of the functional measurement process and implementation aspects. The performance aspects are focused on the reduction of the measurement time. The implementation aspects consider a correct execution of the measurement process avoiding the overflows that can be produced by the measurement of large OO-Method conceptual models.

A measurement example demonstrates how the OOmCFP tool is more efficient than the measurements that are performed manually. This example also shows how the OOmCFP tool can obtain precise measurements for the OO-Method conceptual models.

A comparative analysis shows how a COSMIC-based approach, like OOmCFP, provides a better measurement of the conceptual models that are used in an MDA environment. This is because, in contrast to other FSM standards like IFPG FPA, NESMA FPA or MARK II FPA, it allows the functional size measurement of multi-layer applications (like the applications modeled with the OO-Method approach) from different viewpoints.

Further work will include empirical studies of the reproducibility and the repeatability of OOmCFP. It also include the analysis of the rules presented in this work in order to take into account different points of view (such as the human user viewpoint) for the measurement of applications generated in MDA environments.

References

1. Abrahão, S., Pastor, O.: Estimating the Applications Functional Size from Object-Oriented Conceptual Models. In: International Function Point User Group Annual Conference (IFPUG 2001), Las Vegas, USA (2001)
2. Abrahão, S., Pastor, O.: Measuring the functional size of web applications. *International Journal of Web Engineering and Technology (IJWET)* 1(1), 5–16 (2003)
3. Abran, A., Desharnais, J., Lesterhuis, A., Londeix, B., Meli, R., Morris, P., Oligny, S., O'Neil, M., Rollo, T., Rule, G., Santillo, L., Symons, C., Toivonen, H.: The COSMIC Functional Size Measurement Method, version 3.0 In GELOG, <http://www.gelog.etsmtl.ca>
4. Abran, A., Pierre, N.: Function Points: A Study of Their Measurement Processes and Scale Transformations. *Journal Systems and Software* 25(2), 171–184 (1994)
5. CARE Technologies, <http://www.care-t.com>
6. Condori-Fernández, N.: Un procedimiento de medición de tamaño funcional a partir de especificaciones de requisitos, Doctoral thesis, Universidad Politécnica de Valencia, Valencia, España (2007)
7. Diab, H., Koukane, F., Frappier, M., St-Denis, R.: μ ROSE: Automated Measurement of COS-MIC-FFP for Rational Rose Real Time. *Information and Software Technology* 47(3), 151–166 (2005)
8. Giachetti, G., Marín, B., Condori-Fernández, N., Molina, J.C.: Updating OO-Method Function Points. In: 6th IEEE International Conference on the Quality of Information and Communications Technology (QUATIC 2007), Lisboa, Portugal, pp. 55–64 (2007)
9. IFPUG: International Function Point Users Group, <http://www.ifpug.org>
10. IFPUG, Function Point Counting Practices Manual Release 4.1, International Function Point Users Group, Westerville, Ohio, USA (1999)
11. ISO, International vocabulary of basic and general terms in metrology (VIM), International Organization for Standardization, Geneva, Switzerland (2004)
12. ISO, ISO/IEC 14143-1, Information Technology – Software Measurement – Functional Size Measurement – Part 1: Definition of Concepts (1998)
13. ISO, ISO/IEC 19761, Software Engineering – CFF – A Functional Size Measurement Method (2003)
14. Kitchenham, B.: Counterpoint: The Problem with Function Points. *IEEE Software Status Report* 14(2), 29–31 (1997)
15. Lehne, A.: Experience Report: Function Points Counting of Object-Oriented Analysis and Design based on the OOram method. In: Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA 1997), Atlanta, Georgia (October 1997)
16. Marín, B., Condori-Fernández, N., Pastor, O., Abran, A.: Measuring the Functional Size of Conceptual Models in a MDA Environment. In: The 20th International Conference on Advanced Information Systems Engineering (CAiSE 2008), Montpellier, France (accepted, 2008)
17. OMG: Web site of MDA, <http://www.omg.org/mda/>
18. OO-Method Group Web Site, <http://oomethod.dsic.upv.es>
19. Pastor, O., Molina, J.C.: *Model-Driven Architecture in Practice*. Springer, Heidelberg (2007)
20. Pastor, O., Gómez, J., Insfrán, E., Pelechano, V.: The OO-Method Approach for Information Systems Modelling: From Object-Oriented Conceptual Modeling to Automated Programming. *Information Systems* 26 (2001)

21. Pastor, O., Hayes, F., Bear, S.: OASIS: An Object-Oriented Specification Language. In: Loucopoulos, P. (ed.) CAiSE 1992. LNCS, vol. 593, pp. 348–363. Springer, Heidelberg (1992)
22. Poels, G.: Functional Size Measurement of Multi-Layer Object-Oriented Conceptual Models. In: Proceedings of 9th International Object-Oriented Information Systems Conference, Geneva, Switzerland, pp. 334–345 (2003)
23. Tavares, H., Carvalho, A., Castro, J.: Medicao de Pontos por Funcao a partir da Especificacao de Requisitos. In: Workshop on Requirements Engineering, Universidad Politécnic de Valencia, Spain, November 2002, pp. 278–298 (2002)
24. Uemura, T., Kusumoto, S., Inoue, K.: Function Point Measurement Tool for UML Design Specification. In: 5th International Software Metrics Symposium, IEEE METRICS, Florida, USA, pp. 62–71 (1999)

How Does a Measurement Programme Evolve in Software Organizations?

Lasse Harjumaa, Jouni Markkula, and Markku Oivo

University of Oulu, Department of Information Processing Science

P.O. Box 3000

FIN-90014 OULUN YLIOPISTO

{lasse.harjumaa, jouni.markkula, markku.oivo}@oulu.fi

Abstract. Establishing a software measurement programme within an organization is not a straightforward task. Previous literature surveys have focused on software process improvement in general and software measurement has been analysed in case studies. This literature survey collects the data from separate cases and presents the critical success factors that are specific to software measurement programmes. We present a categorization of the success factors based on organizational roles that are involved in measurement. Furthermore, the most essential elements of success in different phases of the life cycle of the measurement programme are analysed. It seems that the role of upper management is crucial when starting measurement and the individual developers' impact increases in the later phases. Utilization of the measurement data and improvement of the measurement and development processes requires active management support again.

Keywords: Software measurement, software metrics, software process.

1 Introduction

Software measurement is an area that covers a wide variety of activities in software engineering. Defining data that is needed, implementing tools and procedures to collect the data and analysing the data are some of the tasks that are needed. Traditionally, the term "metrics" has been used to characterise source code in a quantifiable manner. Metrics are also used to estimate quality, schedule and resource requirements of software development efforts. Metrics and measurement are essential in order to manage the software development work efficiently, and metrics also enable benchmarking and exact representing of functional requirements. [1]

The main objective of measuring software development is to support managerial decision making. Producing quality software that is delivered to customers on time is a challenging task, and predicting effort that is needed to produce the software is even more challenging. Incorrect estimates of code quality or development lifecycle may cause significant financial losses to the producer organization. Accurate metrics data that is at all times available to managers is extremely important in software process management and improvement [2].

Even though the software metrics and measurement has been actively studied for decades, it still seems that many software organizations do not actively collect and analyze metrics data. Fenton and Neil [3] state that much of the metrics research is irrelevant to practitioners because of irrelevant scope or irrelevant content. While academia is mostly interested in studying very fine-grained and code-related metrics, industry would need more process improvement-related metrics that are easy to collect and use.

Existing software process literature surveys [4] have focused on software process improvement in general and software measurement has been analysed mostly in case studies. This paper is based on a wide literature survey and presents the critical success factors that are specific to software measurement programmes. This paper aims at contributing to the measurement research by clarifying the requirements for a successful measurement programme introduction and evolution. A chronological view to measurement efforts in different phases of the framework implementation is provided. In addition, a categorization outlining requirements from different organizational perspectives is presented in order to efficiently manage the design and implementation of a measurement programme.

The rest of the paper is organized as follows: Section two introduces the research approach that has been utilized in this review. Section three summarizes the related work, general-level success factors of a measurement programme, and introduces the categorization that has been established based on the factors found in literature. In section four, the success of measurement is reflected to the categorization. Furthermore, a time scale of the factors and related organizational roles is sketched. Section five discusses the limitations and implications of this review, and finally section six concludes the work.

2 Research Setting

In this paper, we will analyze the requirements for successful measurement program implementations. We aim at finding the essential elements of the measurement success at different organizational levels. Furthermore, the study aims at describing how the involvement of people in different organizational roles changes while the measurement programme evolves within the organization.

The study consists of a two-phase literature review and a meta-analysis of the recognized software measurement success factors. For finding and categorizing the success factors, a set of well-known and frequently cited research articles is surveyed. The factors listed in different reports are quite widely agreed, and a limited selection of articles was considered to be adequate in order to get a comprehensive view on the substance.

The literature review is continued by searching real industry case studies on measurement programme implementations. Overlapping material was avoided, i.e. articles that were used in success factor categorization, were not used for this analysis. Issues that seemed to arise most often in these experience reports were categorized according to the success factor classification. In addition, involvement of people working in different organizational levels will be studied, and the intensity of involvement in

different phases is evaluated in order to find out the how measurement efforts should be targeted.

Cases for the analysis were selected from the online databases of IEEE, ACM, Elsevier and Springer. Articles had to satisfy the following search criteria: 1) Abstract section of an article must indicate terms “software” and either “measurement” or “metrics” and 2) the abstract section must indicate that a “case study” is reported. The Springer database does not support searching abstracts, so the search was based on paper title. This search resulted in total of 251 articles in the four databases. After checking the titles and abstracts of the articles, 21 articles were chosen for closer look. Especially the result set from the IEEE database includes a number of articles that were presenting measurement techniques and individual metrics when compared to our approach of looking for practical experiences in implementing and sustaining measurement programmes. Naturally, some papers concerned other fields than software engineering. Articles that were either technical (no practical experiences) or not software related were omitted. If the same case was identified in two databases, it was taken into account only once. Articles that concerned the whole measurement programme rather than single metrics were preferred. Both conference articles and journal papers were accepted. The year span of the selected articles was 1993-2006. Table 1 summarizes the search results of the material.

Table 1. Search results from scientific publications databases

	ACM	Elsevier	IEEE	Springer	Total
Total number of articles	53	32	137	29	251
Examined articles	5	2	8	6	21

Thus, the second phase of the literature review is based on 21 case studies reporting both successful and failed measurement and metrics implementations.

3 Success Factors for a Measurement Programme

The literature concerning measurement programme implementations gives quite concurred view of the requisites for successful metrics adoption. This review provides a synthesis of the most crucial success factors and outlines a grouping based on typical organizational roles that are involved in measurement.

3.1 Success Factors

When analysing successes and failures of metrics programmes, one must understand that implementation of such a programme is not a simple activity of utilizing a fixed method or tool within the organization. Implementation and adoption of a measurement framework does not guarantee its routine use in an organization. Resources and long-term commitment are required to gain the maximum benefits of the measurement [5].

Table 2. Success factors of a metrics programme

Success factor	Reference	Hall & Fenton 1997 [7]	Iversen & Mathiassen 2003 [8]	Pfleeger 1993 [6]	Briand et al. 1996 [9]	Dekkers 1999 [10]	Herbsleb & Grinter 1998 [11]	Kitchenham et al. 2006 [12]	Niessink & van Vliet 2001 [13]	Gopal et al. 2005 [5]
Upper management support		x		x						x
Adequate resources		x	x		x	x				x
Establishing reward system			x			x				
Incremental implementation		x	x	x	x					
Constant improvement of the metrics programme		x	x						x	
Aligning with business goals					x				x	
Added value to organization									x	
Measurement data is used at organizational level		x	x	x			x		x	x
Capability to change						x			x	x
Commitment from project managers		x								
Process transparent to developers		x	x	x						x
Well-planned framework		x	x	x	x	x				
Use of existing metrics materials		x		x						
Use of external gurus		x								
Process ownership		x			x					
Usefulness of metrics data		x		x			x		x	
Measurement data is used in project management		x	x	x		x	x		x	x
Feedback to developers		x	x	x		x			x	
Arranging training		x								
Ensuring integrity of data			x			x				
Developer involvement during implementation				x		x				
Providing feedback for improvements			x			x	x		x	x
Data accuracy		x	x			x				
Automated data collection		x		x				x		
Effortless process				x				x		

The adaptation of a measurement framework cannot be judged merely as a success or a failure. Even though the actual data collecting activities may be implemented properly and seem to work well, the data may be useless if the management is not motivated to utilize it in decision-making [6]. Furthermore, objectives of the measurement are defined differently in different organizations.

Literature indicates that the high-level success factors in metrics programme implementations are quite similar in different cases, although the actual implementation is always unique. However, one must notice that research on metrics programmes is typically reported on "case basis", which means success factors have been drawn from a single organization. Another stream of research is to compare metrics programmes in two or more organizations, e.g., [7]. Table 2 summarizes the typical success factors reported in reviewed case studies. The table does not include all the material found in the survey. Instead, it provides a comprehensive overview of the well-known and widely agreed success factors.

The definition of success factors for a measurement programme is highly context dependent. For example, introducing measurement can lead into establishment of the measurement capability or the measurement results can provide useful information and better understanding of the underlying process, development work practices and product quality. We have taken a value perspective; a measurement programme is considered to be successful when it provides added value to the organization and ROI of the measurement programme is clearly positive. Measurement does not always need to lead to SPI type improvements and it is important to distinguish between improvement goals and measurement goals. Even though they are often related, they are logically different. Thus, SPI success factors are not necessarily exactly same as measurement success factors.

Introducing metrics always involves change. Integrating a metrics programme into old practices or refusing any alterations may decrease the probability of achieve the benefits of metrics [13]. This change should be planned. Several studies show that measurement practices should be introduced in deliberate way, including [7, 8, 10]. A dedicated team or a metrics project with clear objectives provides good support for metrics introduction [7]. Setting up the procedures for collecting and analysing the metrics is on the responsibility of the management. Mandating use of metrics without making necessary adjustments to underlying working procedures will most probably cause the metrics programme to fail [5].

Organization's capability to introduce changes into its working procedures and adopt new practices seems to be a critical success factor according to several case studies [5, 6, 8, 10, 14, 15]. In addition to collecting metrics, an organization needs to react to the measurement results. Thus, organizational flexibility helps to achieve benefits from the measurement. Usually the upper management initiates strategic organizational changes while middle management can introduce minor changes. For example, Gopal et al. [5] state that structural changes may be mandatory when introducing a measurement programme. Data collection always affects on underlying working methods, at least in form of new data collection tools. Furthermore, when utilizing the measurement data, it is expected that development processes are transformed into more efficient shape.

Measurement should be started with a simple set of metrics [8, 9]. Attempting to collect every possible piece of data at once will most likely decrease the quality of

data and become burdensome for the personnel. A good starting point is to collect and analyse the data that is already available in some form [8]. The scope and amount of the metrics can be increased later [7, 8].

A case study by Iversen and Mathiassen [8] implies that the better the employees recognize the purpose and importance of the data, the better is the quality of the data. Tying data collection and reporting with bonus system may improve the data quality [8] but is normally not a good approach in process improvement. However, data should never be used directly against developers and those who report the data. Confidentiality of measures is important. Measures of individuals should automatically be available only for those who the data is directly related to. Wider availability has to be agreed with data owners and relevant stakeholders. Results of the measurement can be published, but with care and agreements with the data owners. Metrics provide valuable feedback for developers and teams when they are based on accurate data [6, 8]. Furthermore, measurements should generate as little extra work as possible for the developers [6].

The single most important issue is the use of data [16]. If the data is not used, the whole metrics program will probably turn into a burden that does not help in process improvement. For example, Frederiksen and Mathiassen [14] provide practical suggestions for validating the usefulness of metrics from managerial viewpoint. Furthermore, if the data reveals deficiencies, the organization must take corrective actions based on the measurement [6, 7]. Also individual developers should be aware that the data they provide is appreciated and really used. Usefulness and practical utilization of the metrics data traditionally means that the results of the measurement should be made visible. In addition, measurement should help a software development company to achieve financial or other benefits over its competitors. Thus, measurement should generate value for the organization [13].

Niessink and van Vliet [19] present five different cases of measurement program implementations and analyse the reasons why some of them were more successful than others. They list several success factors that help in achieving organizational goals, thus creating additional value to the organization. The factors they have identified are 1) measurement data is used for reporting purposes 2) measurement data can be used to monitor performance of the organization 3) an organization can learn from the measurement data 4) measurement can help in achieving performance improvement 5) an assessment if the organization meets a set of norms, or benchmarking, can be checked from measurement data (“organizational health”) and 6) measurements help in determining how well organizational goals are achieved and identifying if new directions should be taken (“navigation”) [19].

Support for the measurement has to be concrete. When initializing the measurement programme, it must be ensured that measurements are based on real needs, only meaningful data is collected and proper tools are provided for those who collect, store and analyze the data. Involved people should also be provided adequately time to report their data accurately. Methods like Goal-Question-Metric (GQM) and its further development GQM+strategies have been developed and successfully used to tackle these issues [6, 12, 16, 17, 18, 27]. GQM approach can be used to establish a goal-driven measurement system, starting with definition of organizational goals, measurement goals, and then posing questions to address the goals, and finally identifying appropriate metrics that provide answers to the questions [18].

3.2 Factor Grouping

The list of agreed success factors is quite long and there is certainly some overlapping between separate items. In order to understand the basic elements of success, we will categorize the factors into a more readable form. We will look at the success factors from three different viewpoints. This classification is based on the assumption that effort and involvement that is required from people working in different roles within an organization to achieve effective measurement programme is different. For example, top management is unlikely involved in data collection, and individual developers cannot provide financial resources for running the metrics programme. Thus, we will use the following categorization for classifying the factors for successful measurement.

1) Top management. The upper management is responsible for identifying changes in business environment, setting corresponding business goals for the organization and directing the organization according to the current strategy.

2) Middle management. Managers of specific divisions or processes are responsible for making more fine-grained tactical decision in order to follow the organizational strategy. They put the organization-level business goals into practice by setting concrete goals for products and projects.

3) Developers. Individual designers and coders form the operational level that actually creates the products and projects. They also produce the data that is needed for measurement. Interestingly, it is developer related actions and work products that are measured and developers themselves are usually mainly responsible for collecting the measurement data.

In addition to organization-level categorization, the factors can be grouped into five high-level maxim that capture the essence of the measurement implementation. These core topics can be named as a) commitment, which is required from everybody that is involved, b) planning, that is necessary in defining and managing the process, c) data utilization, which means that people know that the measurement data is used to guide and manage the organization, d) training & knowledge, which ensure that everybody understand the meaning of different metrics and knows how to report and interpret the data, and finally e) tool support to make the measurement process effortless to operate. We have based the categorization on the success factors found in literature to structure the main high-level elements of success. Figure 1 illustrates these categorizations. On the left side, all the success factors are grouped from the organizational viewpoint. Some items are listed more than once, as they relate to more than one organizational level (“measurement data is used”, for example). On the right side, the grouping into five top-level values is presented with links to related original success factors.

There are several issues that need to be paid special attention in all three organizational levels. Two-way feedback, commitment and real use of the data are factors that cross the levels. Ensuring these necessities will also increase understanding of the measurement and transparency of the programme. Training, proper understanding of metrics and tool support for metrics collection and analysis are mostly connected to the operational level, while more abstract issues, commitment and planning are important from the management point of view. Data utilization concerns basically everybody in the organization.

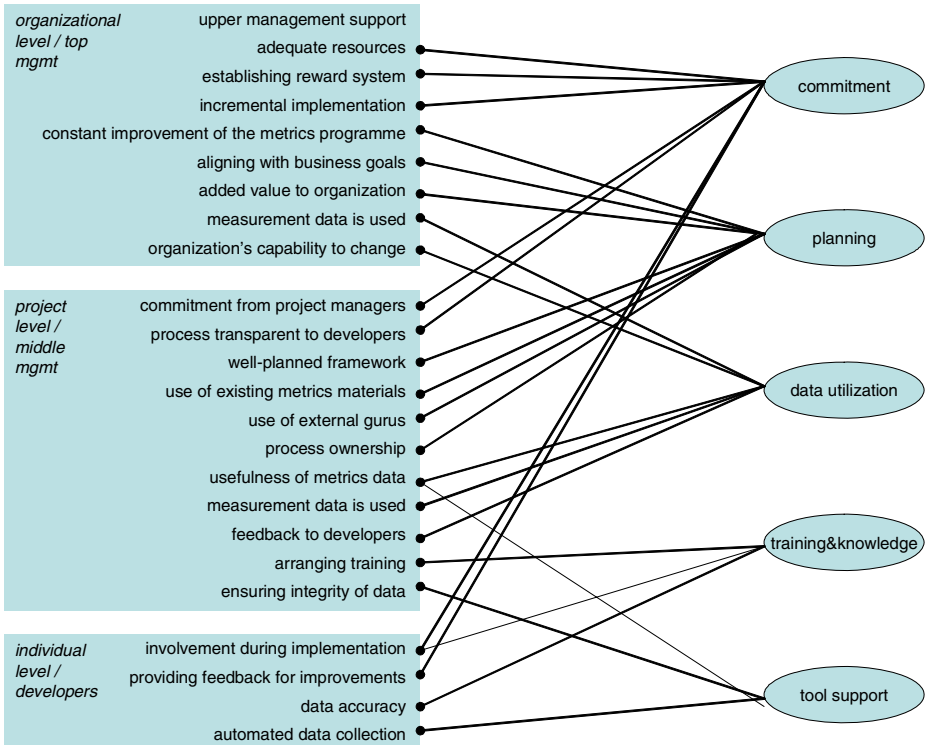


Fig. 1. Categorization of software measurement success factors

To summarize, measurement involves several other stakeholders at different organizational levels. Project managers are the main persons responsible for keeping the measurement in operation and the work of individual developers is most likely to change when the metrics collection is started.

4 Involvement in Different Phases of Measurement Implementation

We suggest that not all organizational roles are similarly involved in the measurement programme implementation in different phases. Examination of the timeline from the measurement programme initiation to its operation helps to understand, who should act and when in order to succeed.

4.1 Viewpoints to Measurement Case Studies

Selected articles were categorized according to the five high level success factor categories. Attention was paid to the most important contributions of the articles by searching the main theme and the most emphasized lessons learnt from each representation. Typically each study emphasized a specific viewpoint to the software

measurement field and it was easy to categorize the article accordingly. Some of the papers have been set into more than one category.

Planning is considered very important and many of studies focused on providing guidance for the planning phase. For example, Latum et al. [16] and Berander and Jönsson [20] report cases in which GQM approach has been used for the measurement definition. Proper planning the measurement framework and prioritization of the measurement goals may help in keeping the size of the measurement framework manageable [20].

The most studied facet is tool support for data collection, analysis and visualization. According to Johnson et al. [21], developers may not be willing to adopt metrics tools if they have to switch from the development environment to another set of tools for recording metrics. Automated tools help the individual developers' metrics collection process, but on the other hand, set new requirements on development tools. Furthermore, developers may feel that their privacy is threatened when data is recorded automatically [21].

Even though commitment of the senior management is recognized as a crucial success factor, aspects of people's attitudes towards measurement programmes have been little investigated. Furthermore, upper management commitment is usually listed among the most widely agreed success factors, but commitment is needed in every level in order to ensure accurate and adequate measurement data.

Measurement-related knowledge is another aspect that hardly any case study addresses, although training is obviously needed when introducing new methods and tools within the organization. Furthermore, understanding and knowledge of the measurement process develops over time in both individual and organizational level. It would be important to understand how the underlying measurement programme works and how the data can be explored in order to improve the measurement [22].

Table 3 summarizes the number of research articles focusing on different viewpoints to software measurement.

Table 3. The number of articles focusing on different measurement viewpoints

Planning	6
Tools	11
Commitment	2
Training & knowledge	1
Data utilization	8

During the analysis, it became evident that most of the research focuses on specific metrics, defining new metrics and implementations of tools for metrics collection and representation. There are, however, a number of articles describing measurement programme implementations. Especially interesting are reports that analyze possible reasons for a failure of measurement programme introduction. It is probably easier to detect absence of a particular success factor than its presence.

4.2 Measurement Evolution

Establishing a software measurement programme is not a straightforward task. Measurement process needs to be introduced carefully and maintained actively in order to keep it running and effective. Adoption of software measurement tools and procedures is unlikely to happen instantaneously. Implementation of the measurement programme usually requires defining, designing and implementing the appropriate tools for collecting and analysing data. This series of tasks and organizational processes typically follows the stages in innovation diffusion models describing the adoption of a new idea within an organization. Innovation diffusion is a widely studied theory in the information systems field describing how a new idea develops in an organization in stages. Rogers [23] has entitled these phases as knowledge, persuasion, adoption and routinization.

Another way looking at the evolution of a software measurement programme is the development of organizational memory. Measurement data describe the processes and products of a company, thus they contribute significantly to the organization's knowledge, which develops and cumulates in time. Anand et al. [24] define the concept of organizational memory as follows: "information and knowledge known by the organization and the processes by which such information is acquired, stored and retrieved by organization members". According to Stein [25], the organizational memory process consists of the following phases: 1) acquisition, 2) retention, 3) maintenance and 4) retrieval. When looked from the measurement viewpoint, acquisition phase is related to defining and operationalization of the needed metrics and collection of the data. Retention phase concerns the data structures and systems, as well as established practices of data collection. Maintenance phase relate to measurement data quality and reliability management. Retrieval phase concerns availability of the metrics for decision making, addressing as such also the analysis and presentation methods, tools and practices of metrics usage.

Figure 2 illustrates these four phases and points out the observations that have been emerged in the literature review. In the first phases, the upper management role is most crucial, as planning, resourcing and necessary organizational changes can be supplied only by management. When moving towards latter phases, operational roles are more important, because it depends heavily on developers and project managers, how accurate the data is - or if it is collected at all.

For example, Gopal et al. [5] stress the role of upper management in guiding the metrics framework implementation and providing resources for establishing the measurement successfully. The management role in adaptation is crucial, as it may be required to introduce structural changes within the organization [5]. If there are organizational and managerial problems, metrics programme will fail more likely than in well-managed organizations [15].

Even a well-planned metrics framework can result in a failure, if the data is not used to initiate improvements in organization-level software processes, as a case reported in [19] suggests. Another case in the same article shows that in addition to rigorous implementation, data integrity needs to be ensured during the operation of measurement.

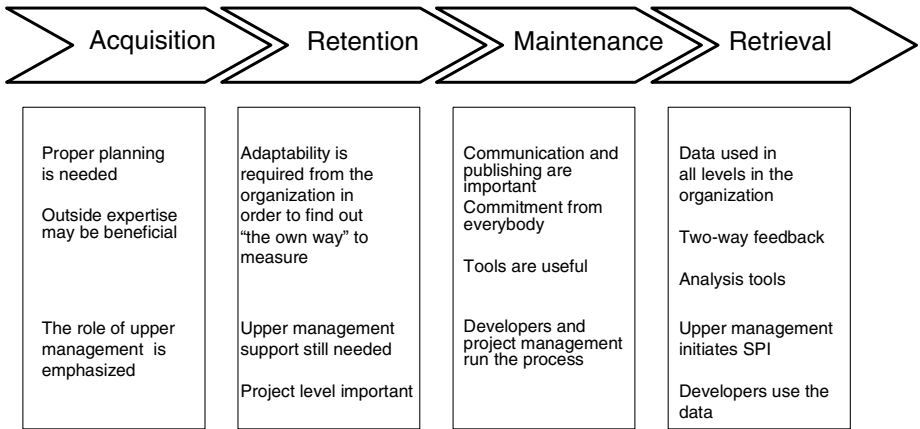


Fig. 2. The phases of measurement programme implementation

The case of Contel Corporation [6] demonstrates that streamlined data collection and analysis is necessary to achieve good results from measurement. Daskalantonakis [26] lists guidelines for data collection and interpretation and automated data gathering tools among the most important success factors in Motorola. It seems that without proper tools measurement may become too strenuous.

The main issue in keeping the measurement programme alive is its usefulness. Niessink and van Vliet [19] state that a measurement program should create value for the organization. This means that the data gathered in measurement must be used in a meaningful way. Positive influence in development costs, developer productivity or customer feedback is the best motivators to keep the measurement running.

The change in software engineering industry is quite rapid. As the markets change, development organizations' business goals will change. Metrics and measurement should be linked to the business goals of an organization to help in high-level decision making [27]. Thus, changes in the business and organizational goals should be reflected in the measurement. If metrics that are collected are not relevant, they are useless. Furthermore, criteria for success or failure can be different in different business situations.

Upholding a metrics framework is quite similar to keeping software process improvement continuous and running. Adequate resourcing, appropriate people roles and constant, visible feedback are necessities of success. Similar to SPI, the attitude of upper management in the beginning reflects to the later phases and the motivation of individual employees, who have a vital role in the operational phase.

5 Discussion

The main contribution of this study for practitioners is the model describing software measurement programme evolution. Paying attention to the typical life cycle of a metrics programme and targeting resources according to the involvement required by

top management, middle management or developers should help the organization to manage the measurement and its implementation.

Related research work usually stresses the importance of upper management support but does not describe concretely, in which phases the management support is most beneficial. Furthermore, we have arranged the success factors into two-level hierarchy that gives either overview of the measurement-related issues or more detailed list of the success factors. This helps to understand and direct the measurement efforts into right direction in the organization.

For academia, our review provides a new view on categorizing and prioritizing the well-known success factors. We would like to add organizational adaptability to the list of success factors. Certain research articles on software metrics report the organization's capability to change as a critical success factor, and in a number of studies this requirement can be observed implicitly. However, we would like to emphasize the importance of this organizational capability. We also believe that organizational aspects should be paid more attention to in software measurement research. The measurement programme can be reflected to innovation diffusion, for example [5]. This paper utilizes the concept of organizational learning for describing the institutionalization of a measurement programme. Life cycle model presented in this paper is quite rough and needs further studying. Empirical, industrial cases for refining the model and validating it are needed and we are planning to evaluate the metrics frameworks used in software development companies towards our categorization and life cycle model.

The analysis of measurement case studies also gives an overview, which aspects of the field are most studied. As can be noticed, tool support collecting and analyzing the metrics have been greatly emphasized in research, while training has been much less addressed. This could be an indication of a need for further research.

The number of articles that reported real-life experiences in such detail that they could be selected for the analysis was fairly low. More variety in the search criteria would have, of course, increased the amount of material. However, even with this magnitude, the evidence shows the directions that measurement research is going.

6 Conclusion

Establishing a software measurement programme is not a straightforward task. Measurement process needs to be introduced carefully and maintained actively in order to keep it running and effective. Adoption of software measurement tools and procedures is unlikely to happen overnight. Implementation of the measurement programme requires defining, designing and possibly implementing the appropriate tools for collecting and analysing data.

The key factors for keeping a metrics programme alive and continuously improving include the proper use of the data and meaningful tools to enable easy collecting and representation of the measurement results. Adjustment of the measurement according changing business surroundings is necessary in order to keep the data valid and meaningful.

Involvement of people working on different organizational levels varies in different phases of the measurement programme life cycle. In the beginning, the role of the

top management is very important, as initiating the required resources and motivation for use of metrics have to be ensured at the organizational level. Later on, when the measurement process matures, the role of middle management becomes more emphasized for streamlined and efficient operation of the process. Along the way, motivation and knowledge of individual developers have to be ensured in order to gain accurate and adequate data.

Further research is needed on the evolution aspect of a measurement programme in order to efficiently guide and manage measurement-related efforts. Training, knowledge and motivation of the people working with metrics have been rarely studied, yet those are extremely important issues to understand. One interesting approach would also be to contrast the measurement programme evolution with the maturity level of the organization.

References

1. Kan, S.H.: *Metrics and Models in Software Quality Engineering*. Addison-Wesley, Reading (1995)
2. Rainer, A., Hall, T.: A quantitative and qualitative analysis of factors affecting software process improvement. *Journal of Systems and Software* 66(1), 7–21 (2003)
3. Fenton, N.E., Neil, M.: *Software Metrics: Roadmap*. In: *Proceedings of The Future of Software Engineering*, pp. 357–370 (2000)
4. Dyba, T.: An empirical investigation of the key factors for success in software process improvement. *Transactions on Software Engineering* 31(5), 410–424 (2005)
5. Gopal, A., Mukhopadhyay, T., Krishnan, M.S.: The Impact of Institutional Forces on Software Metrics Programs. *IEEE Transactions on Software Engineering* 31(8) 679–694 (2005)
6. Pfleeger, S.: Lessons learned in Building a Corporate Metrics Program. *IEEE Software* 10, 67–74 (1993)
7. Hall, T., Fenton, N.: Implementing Effective Software Metrics Programs. *IEEE Software* 14(2), 55–65 (1997)
8. Iversen, J., Mathiassen, L.: Cultivation and Engineering of a Software Metrics Program. *Info Systems Journal* 13(1), 3–19 (2003)
9. Briand, L., Differding, C., Rombach, D.: Practical Guidelines for Measurement-based Process Improvement. *Software Process Improvement and Practice* 2(4), 253–280 (1996)
10. Dekkers, C.A.: The Secrets of Highly Successful Measurement Programs. *Cutter IT Journal* 12(4), 29–35 (1999)
11. Herbsleb, J.D., Grinter, R.E.: Conceptual Simplicity Meets Organizational Complexity: Case Study of a Corporate Metrics Program. In: *Proceedings of the International Conference on Software Engineering*, pp. 271–280 (1998)
12. Kitchenham, B., Kutay, C., Jeffery, R., Connaughton, C.: Lessons learnt from the analysis of large-scale corporate databases. In: *Proceedings of the International Conference on Software Engineering (ICSE 2006)*, pp. 439–444 (2006)
13. Niessink, F., van Vliet, H.: Measurements Should Generate Value, Rather than Data. In: *Proceedings of the IEEE Metrics Symposium*, pp. 31–38 (1999)
14. Frederiksen, H.D., Mathiassen, L.: Information-Centric Assessment of Software Metrics Practices. *IEEE Transactions on Engineering Management* 52(3), 350–362 (2005)
15. Berry, M., Jeffery, R.: An Instrument for Assessing Software Measurement Programs. *Empirical Software Engineering An International Journal* 5(3), 183–200 (2000)

16. Latum, F., Solingen, R., Oivo, M., Hoisl, B., Rombach, H.D., Ruhe, G.: Adopting GQM-Based Measurement in an Industrial Environment. *IEEE Software*, 78–86 (1998)
17. Basili, V., Weiss, D.: A Methodology for Collecting Valid Software Engineering Data. *IEEE Transactions on Software Engineering* SE10(6), 728–738 (1984)
18. Basili, V., Caldiera, G., Rombach, D.: Goal Question Metric Paradigm. In: Marciniak, J.J. (ed.) *Encyclopedia of Software Engineering*, vol. 1, pp. 528–532. John Wiley & Sons, Chichester (1994)
19. Niessink, F., van Vliet, H.: Measurement program success factors revisited. *Information & Software Technology* 43(10), 617–628 (2001)
20. Berander, P., Jönsson, P.: A goal question metric based approach for efficient measurement framework definition. In: *Proceedings of the Fifts ACM-IEEE International Symposium on Empirical Software Engineering (ISESE)*, pp. 316–325 (2006)
21. Johnson, P.M., Kou, H., Paulding, M.G., Zhang, Q., Kagawa, A., Yamashita, T.: Improving Software Development Management through Software Project Telemetry. *IEEE Software* 22(4), 76–85 (2005)
22. Mendonça, M.G., Basili, V.R.: Validation of an Approach for Improving Existing Measurement Frameworks. *IEEE Transactions on Software Engineering* 26(6), 484–499 (2000)
23. Rogers, E.M.: *Diffusion of innovations*, 4th edn. Free Press, New York (1995)
24. Anand, V., Manz, C.C., Glick, W.H.: An organizational memory approach to information management. *Academy of Management Review* 23(4), 796–809 (1998)
25. Stein, E.W.: Organizational memory: review of concepts and recommendations for management. *International Journal of Information Management* 15(2), 17–32 (1995)
26. Daskalantonakis, M.K.: A Practical View of Software Measurement and Implementation Experiences Within Motorola. *IEEE Transactions on Software Engineering* 18(11), 998–1010 (1992)
27. Basili, V., Heidrich, J., Lindvall, M., Munch, J., Regardie, M., Trendowicz, A.: GQM + Strategies - Aligning Business Strategies with Software Measurement. In: *Proceedings of the 1st International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*, Madrid, Spain (2007)

A Fault Prediction Model with Limited Fault Data to Improve Test Process

Cagatay Catal¹ and Banu Diri²

¹The Scientific and Technological Research Council of TURKEY,
Marmara Research Center, Information Technologies Institute
Kocaeli, TURKEY

cagatay.catal@bte.mam.gov.tr

²Yildiz Technical University, Department of Computer Engineering
Istanbul, TURKEY

banu@ce.yildiz.edu.tr

Abstract. Software fault prediction models are used to identify the fault-prone software modules and produce reliable software. Performance of a software fault prediction model is correlated with available software metrics and fault data. In some occasions, there may be few software modules having fault data and therefore, prediction models using only labeled data can not provide accurate results. Semi-supervised learning approaches which benefit from unlabeled and labeled data may be applied in this case. In this paper, we propose an artificial immune system based semi-supervised learning approach. Proposed approach uses a recent semi-supervised algorithm called YATSI (Yet Another Two Stage Idea) and in the first stage of YATSI, AIRS (Artificial Immune Recognition Systems) is applied. In addition, AIRS, RF (Random Forests) classifier, AIRS based YATSI, and RF based YATSI are benchmarked. Experimental results showed that while YATSI algorithm improved the performance of AIRS, it diminished the performance of RF for unbalanced datasets. Furthermore, performance of AIRS based YATSI is comparable with RF which is the best machine learning classifier according to some researches.

Keywords: Semi-supervised learning, software fault prediction, YATSI, artificial immune systems, AIRS.

1 Introduction

Software testing is one of the most crucial quality assurance activities for Software Quality Engineering. Beyond testing, there are various quality assurance alternatives that can be applied for high-assurance systems such as flight control software and large-scale telecommunication software. Some of these quality assurance activities are formal verification, fault prevention, fault tolerance, and fault prediction [1].

Software systems are becoming more and more complex. If we examine the evolution of Microsoft's Operating Systems (OS) with respect to lines of code, we can easily observe the complexity. According to Andrew Tanenbaum [2], Windows NT 3.1 had 6 Million lines of code (MLOC) in 1993 and Windows Vista Beta 2 had 50

MLOC in 2005. Mac OS X 10.4 has 86 MLOC [3], Debian 3.1 has 215 MLOC [2], and Eclipse Europa release [4] has 17 MLOC. As we see from these large-scale projects, lines of code for today's software can be expressed MLOC scale. According to the Assistant Secretary of the U. S. Army, the systems of the future are likely to have billions of lines of code. The Software Engineering Institute published a study report in June 2006 after 12-month investigation to solve the ultra-large-scale system problem of U.S. Department of Defense [5]. According to the current and future complexity of software systems, we can express that effective quality assurance activities are required to improve the quality of software systems.

Software fault prediction activity mostly uses previous software metrics and fault data to predict the fault-prone modules for the next release of software. This activity creates a model applied before system testing in the next release of software. Some tools [6] or experts may divide modules into three groups: High, Medium and Low fault-prone. Some researchers [7] also predicted the number of faults for each module of software. Halstead and McCabe metrics are method-level metrics and used as independent variables. Chidamber-Kemerer metrics suite [8] is an example for class-level metrics suite, but in this study we only applied method-level metrics. Various benefits of software fault prediction models are explained below:

- Identification of refactoring candidates,
- Improvement for software testing process and quality,
- Selection of the best design from alternatives by using class-level metrics,
- Reaching a highly assurance system.

Mostly researchers used public datasets during their studies from NASA Software Metrics Data Program. PROMISE repository stores public datasets from NASA [9]. We built prediction models using Artificial Immune Systems paradigm during our Fault Prediction Research Program [10]. However, our models used previous software metrics and fault data for modeling. When a company starts a new project type or does not have any previous fault data, these models can not be built. Therefore, we need new approaches under these circumstances. Zhong et al. [11] proposed unsupervised learning approaches using Neural-Gas and K-means clustering algorithms in 2004. Another challenging problem occurs when the company has few fault data. Supervised learning approaches can not build powerful models with few fault data. One solution can be using unlabeled software modules together with labeled data during learning. This type of learning is known as semi-supervised learning. Some situations where we need semi-supervised fault prediction models are given below [12]:

- During decentralized software development, some companies may not collect fault data for their software components.
- Execution cost of data collection tools may be expensive.
- Company may not collect fault data for a version due to the lack of budget.

Seliya et al. [12] used Expectation-Maximization (EM) algorithm for this problem in 2004 and had comparable results with classification algorithms. Seliya et al.'s study [12] is the first study in literature for semi-supervised software fault prediction. In this study, we investigate RF, AIRS, and YATSI algorithms for semi-supervised software fault prediction. YATSI algorithm has been proposed in 2006 by Driessens et al. [13], but

its performance has not been observed for unbalanced datasets specifically. It is a meta semi-supervised algorithm and it has been evaluated using several datasets. The evaluation parameter was accuracy in their study, but our study focuses on software fault prediction problem which has unbalanced limited fault data. The reason why we use RF is its high performance for software fault prediction as reported by Ma et al. [14] and Guo et al. [15]. According to their study, RF is the best classifier among other machine learning classifiers that locate in WEKA [16] and See5 [17]. Furthermore, AIRS has been chosen because of its high performance reported by Catal et al. [10]. In first stage of YATSI, RF and AIRS have been applied.

We attempt to answer five questions given below in this study:

- 1) Does YATSI improve the performance of AIRS for semi-supervised fault prediction?
- 2) Does YATSI improve the performance of RF for semi-supervised fault prediction?
- 3) How accurately do AIRS based YATSI, RF based YATSI, RF, and AIRS algorithms predict faults for semi-supervised fault prediction?
- 4) How does performance of algorithms change when percentages of labeled modules increase?
- 5) Does YATSI always increase the performance of classifiers?

To the best of our knowledge this is the first study to investigate Artificial Immune Systems paradigm for semi-supervised learning. This study explores performance of YATSI algorithm with unbalanced datasets for the first time. Furthermore, this study is the second attempt which applies semi-supervised learning algorithms for software fault prediction with limited fault data. This paper is organized as follows: the following section presents the related work. Section 3 explains semi-supervised learning approach. Section 4 introduces our modeling approach. Section 5 shows experimental results. Section 6 presents the conclusions and future works.

2 Related Work

Most studies on software fault prediction focused on using a supervised learning approach. Genetic Programming [18], Decision Trees [19], Neural Networks [20], Naïve Bayes [21], Dempster-Shafer Networks [22], Case-based Reasoning [23], Fuzzy Logic [24], and various different methods have been used as supervised learning approaches. We applied Artificial Immune Systems paradigm for software fault prediction during our Fault Prediction Research Program [10]. Except Seliya et al.'s study [12], we did not encounter any semi-supervised learning approach for software fault prediction problem. Seliya et al. [12] used EM algorithm for this problem and labeled the unlabeled data points iteratively. Missing values for EM algorithm were the class labels of the unlabeled data. According to their study, EM algorithm provided comparable results with classification algorithms. Driessens et al. [13] proposed a simple semi-supervised learning algorithm called YATSI in 2006. In this study, we applied AIRS algorithm in first step of YATSI algorithm. We investigated AIRS in first step because its accuracy was high for software fault prediction during our researches [10], [25]. Furthermore, we applied RF for the first step of YATSI because Ma et al. [14]

reported that RF always achieves better performance than other machine learning algorithms. Guo et al. [15] observed that performance of the RF is better than Discriminant analysis, logistic regression, and other machine learning algorithms that locate in WEKA [16] and See5 [17].

3 Semi-supervised Learning

Supervised and unsupervised learning methods in machine learning are well-studied subjects, not only in the context of the advanced algorithms, but also in terms of their clear benefits. Supervised learning methods use a training dataset consisting of inputs and their relevant labels to learn the input-output relationship. Unsupervised learning methods do not use class labels and they are categorized into clustering and component analysis groups [26]. However, collecting class labels is a time consuming, and expensive process because this process requires human efforts to be able to label each data point [27]. In speech recognition, recording speech is so cheap but labeling huge amount of speech requires a human to listen all of the recorded data [28]. If we could use unlabeled data together with labeled one for classification and clustering problems, we could prevent spending unnecessary time on labeling phase. Semi-supervised learning area in machine learning is interested in building such algorithms learning from labeled and unlabeled data. Genetic research, medical diagnosis, spam email detection, bioinformatics, and computer vision are the major areas which there are needs to explore semi-supervised algorithms.

Semi-supervised learning area can be categorized into two groups: semi-supervised classification and semi-supervised clustering. Traditional machine learning classifiers can not benefit from unlabeled data because they have not been designed for this purpose. Therefore, we need some algorithms to enhance performance with unlabeled data. Actually, semi-supervised learning concept is not a new idea. During 1960s, self-training approaches have been proposed [29], [30], [31] and they are accepted as the first methods which benefits from unlabeled data.

Some researchers are very optimistic to use semi-supervised classification algorithms when there is not enough labeled data [32], but building an accurate semi-supervised model requires a huge effort. Furthermore, using unlabeled data together with labeled one for classification does not guarantee to improve the classifier performance. Many publications [33], [34], [35], [36], [37] reported the performance improvement of the classifiers when unlabeled data are used during IJCAI2001, NIPS1998, NIPS1999, and NIPS2000 workshops [32]. Even though many researchers showed that unlabeled data improves the performance of classifiers, there exist papers reporting the performance degradation of classifiers [38]. Nigam et al. [34] showed that degradation in classifier performance can result when the data do not conform to the assumptions of the model. Baluja [33] and Shahshahani et al. [39] reported degradation in imagine understanding. Bruce [40] observed degradation in Bayesian network classifiers. Elworthy [41] explained that Hidden Markov Model with unlabeled data can lead to degradation in classifier accuracy for some occasions. Some researchers suggest that unlabeled samples should only be used if classifier performance is not satisfactory [32]. Transductive inference, proposed by Vapnik [42], is similar to semi-supervised learning. A general decision rule is not generated and only labels of

the unlabeled points are predicted [28] for transductive inference. We can divide the existing semi-supervised algorithms into five groups:

1-) *Self-training*: A classifier uses a small portion of the labeled dataset for training and the generated model is used to label the unlabeled data. The most confident data points from the new labeled one are added to the training set and this process is repeated until convergence [27]. Yarowsky [43] used self-training for word sense disambiguation problem.

2-) *Co-training*: Co-training proposed by Blum et al. [44]. Features are split into two sets and each classification algorithm is trained with one of these sets. Each classifier predicts the labels of unlabeled data and teaches the most confident data points to the other classifier. After this step, classifiers are again retrained and this process is repeated. Nigam et al. [45] made experiments to benchmark co-training with EM and generative mixture models.

3-) *Transductive Support Vector Machines (TSVM)*: Bennett et al. first implemented TSVMs using an integer programming method [37]. Joachims [46] implemented a combinatorial transductive approach called SVM-light TSVM and this is the first widely used software in this area [27]. The aim is to label unlabeled data so that maximum margin is reached on the available labeled data and new labeled data [27].

4-) *Graph based Methods*: Unlabeled and labeled data points are nodes and similarity of examples are edges for graph based methods [27]. Blum et al. [47] and Zhu et al. [48] proposed different graph based methods. Cluster kernels and Markov random walks are some examples of graph based methods. The performance of these methods depends on the graph structure and edge weights.

5-) *Generative models*: “It assumes a model $p(x, y) = p(y) p(x|y)$ where $p(x|y)$ is an identifiable mixture distribution, for example Gaussian mixture models” [27].

4 Modeling Approach

5%, 10%, and %20 of datasets have been used as training sets and rest data have been used as test sets. Arranging datasets with these percentages let us simulate the labeled-unlabeled data problem for software fault prediction using labeled datasets.

4.1 Immune Based YATSI Algorithm

Performance of YATSI is correlated with performance of the classification algorithm used in first stage. For this reason, in first stage of YATSI we experimented with two high performance algorithms, RF and AIRS, which have proved their performance for software fault prediction. Ma et al. [14] and Guo et al. [15] showed that RF is the best classification algorithm in WEKA and See5. Catal et al. [10] proved that AIRS provides remarkable results for software fault prediction. Immune based YATSI is a special form of YATSI algorithm for software fault prediction and it deals only with two classes, fault-prone and not fault-prone. Furthermore, adjustment factor for YATSI, F , is chosen 1 in this algorithm. RF based YATSI is same with this algorithm except the usage of AIRS in first stage. As in YATSI algorithm, the number of nearest neighbor is fixed to 10 and KD-trees are used for nearest neighbor search. Weights of the data points locating in labeled dataset are chosen 1 and weights of pre-labeled

points are smaller than 1. Weights of the pre-labeled points are calculated by dividing the number of labeled data points to the number of unlabeled data points. The reason of this difference is that we trust the labeled data points much more than pre-labeled points. In YATSI algorithm, weights are summed for each class and the largest weight identifies the class label of the data point. If it is in the nearest neighbors set, the only factor is its weight [13]. The pseudo code for immune based YATSI is shown below and it is similar to YATSI algorithm.

Algorithm	Pseudo code for Immune-based YATSI algorithm
Input:	Labeled data D_l , unlabeled data D_u , nearest neighbor number K , $N= D_l $, $M= D_u $, unlabeled data point d_u
Step 1:	Use AIRS classifier to produce the model M_l using D_l Use M_l to pre-label data points of D_u Assign 1.0 weight to data points of D_l and N/M weight to points of D_u Combine D_l and D_u to produce D
Step 2:	For each data point, d_u , inside D_u Search for the K -nearest neighbors to the d_u Sum the weights of the fault-prone K -nearest neighbors (W_{fp}) Sum the weights of the not fault-prone K -nearest neighbors (W_{nfp}) Predict the label of the d_u with the largest weight (W_{fp} or W_{nfp})

4.2 Artificial Immune Recognition Systems

Artificial Immune Systems is a biologically inspired computing paradigm such as Neural Networks, and Swarm Intelligence. This paradigm has been used for several application areas such as robotics, data mining, and computer security. Watkins [49] proposed Artificial Immune Recognition Systems (AIRS) algorithm for classification problems. Resource-limited approach of Timmis et al. [50] and clonal selection approach of De Castro et al. [51] are applied in this algorithm. Brownlee [52] implemented this algorithm in Java. Performance of AIRS algorithm has been examined for various machine learning datasets [49]. Each data point in dataset is called antigen. This algorithm has the following features [52]:

- Self-regulatory: No need to identify a topology before training,
- Performance: High performance for several datasets,
- Parameter stability: No need to optimize parameters of the algorithm,
- Generalization: No need to use all the data points for generalization.

The details of the algorithm are explained below [10]. Step 2, 3, 4 are used for each data point in dataset and Step 1 and 5 are applied one time.

1-) *Initialization*: Dataset is normalized into [0, 1] interval. Affinity threshold variable is computed.

2-) *Antigen Training*: Each data point in training set is provided to the memory pool to stimulate the recognition cells in memory pool. Stimulation values are assigned to the recognition cells and the cell which has maximum stimulation value is marked as the best memory cell. This cell is used for affinity maturation and cloned, then mutated. These clone cells are put into the Artificial Recognition Ball (ARB) pool.

Formula 1 depicts the stim formula. Formula 2 shows how to compute the number of clones.

$$\text{stim} = 1 - \text{affinity} . \quad (1)$$

$$\text{numClones} = \text{stim} * \text{clonalRate} * \text{hypermutationRate} . \quad (2)$$

3-) *Competition for limited resource*: After mutated clones are added to the ARB pool, competition starts. Antigen is used to stimulate the ARB pool and limited resource is computed with respect to stimulation values. ARBs with very limited resource or no limited resource are deleted from ARB pool. This step continues until stopping criteria is met. Otherwise, mutated clones of ARBs are produced.

4-) *Memory cell selection*: Candidate memory cell which has a maximum stimulation score from ARB pool is chosen. ARB is copied to the memory cell pool if ARB's stimulation value is better than the original best matching memory.

5-) *Classification*: Memory cell pool is used for cross-validation and K-nearest neighbor approach is applied for classification.

4.3 Random Forests

RF is a classification algorithm that includes tens or hundreds of trees. Results of these trees are used for majority voting and RF chooses the class who has the highest votes. Breiman's algorithm has been used in this study. "Each classification tree is built using a bootstrap sample of the data, and at each split the candidate set of variables is a random subset of the variables." [53]. "Each tree is constructed using the following algorithm [54]:

1. Let the number of training cases be N , and the number of variables in the classifier be M .
2. We are told the number m of input variables to be used to determine the decision at a node of the tree; m should be much less than M .
3. Choose a training set for this tree by choosing N times with replacement from all N available training cases (i.e. take a bootstrap sample). Use the rest of the cases to estimate the error of the tree, by predicting their classes.
4. For each node of the tree, randomly choose m variables on which to base the decision at that node. Calculate the best split based on these m variables in the training set.
5. Each tree is fully grown and not pruned (as may be done in constructing a normal tree classifier)".

5 Experimental Study

5.1 System Description

The datasets which belong to NASA projects have been accessed from PROMISE repository [9]. Each dataset has 21 metrics shown in Table 1. Table 2 depicts the datasets and their properties.

Table 1. Metrics inside datasets

Attributes	Information
loc	McCabe's line count of code
v(g)	McCabe "cyclomatic complexity"
ev(g)	McCabe "essential complexity"
iv(g)	McCabe "design complexity"
n	Halstead total operators + operands
v	Halstead "volume"
l	Halstead "program length"
d	Halstead "difficulty"
i	Halstead "intelligence"
e	Halstead "effort"
b	Halstead delivered bugs
t	Halstead's time estimator
IOCode	Halstead's line count
IOComment	Halstead's count of lines of comments
IOBlank	Halstead's count of blank lines
IOCodeAndComment	Lines of comment and code
uniq_Op	Unique operators
uniq_Opnd	Unique operands
total_Op	Total operators
total_Opnd	Total operands
branchCount	Branch count of the flow graph

Table 2. Datasets and their details

Dataset	Language	LOC	Project	Fault %	# of methods
KC2	C++	43K	Data processing	21 %	523
CM1	C	20K	Instrument	21%	498
PC1	C	40K	Flight software	7%	1109
JM1	C	315K	Real time	19%	10885

5.2 Experimental Setting

Eclipse has been used for our software development environment. WEKA source code from <http://www.cs.waikato.ac.nz/~ml/weka/> website, MARSDEN project from <http://www.cs.waikato.ac.nz/~fracpete/marsden/> website, and Artificial Immune System based algorithms from <http://weka.classalgos.sourceforge.net> website have been downloaded and exported into a Java project in Eclipse. MARSDEN includes several semi-supervised algorithms and YATSI is one of these algorithms. Since AIRS algorithm is a 3rd party classifier, it did not include Capabilities framework in its implementation. Therefore, we changed the source code of AIRS algorithm by overriding *getCapabilities()* method inside AIRS1 class. After building the project, a jar file has been created. During our experiments, we used Experimenter tool that locates in WEKA. We added four datasets from Datasets panel and four algorithms from Algorithms panel. We repeated experiment 20 times. Experiment type has been chosen as “Train/Test Percentage Split (data randomized)” and “Train percentage” has been used 5, 10, and 20 for our experiments.

5.3 Evaluation Criteria

Fault prediction datasets are mostly unbalanced and therefore, accuracy should not be used as evaluation criteria for classifier performance. Bradley [55] used AUC to compare several machine learning algorithms and showed that AUC has better properties than accuracy. Ling et al. [56] suggested using AUC for comparing classification systems and showed that AUC is more statistically consistent than accuracy for balanced or unbalanced datasets. Therefore, we compared performance of classifiers using AUC values.

5.4 Results and Analysis

We have conducted several tests and used J48, AIRS, and YATSI algorithms. Ma et al. [14] used G-mean1, G-mean2, and F-measure for their benchmarking study. They had marked the top three algorithms with respect to G-mean1, G-mean2, and F-measure values. They showed that RF always has top three values for G-mean1, G-mean2, and F-measure. However, in our experiments we could not use this approach because no algorithm had this feature in these datasets. Therefore, we used ROC value to compare the performance of algorithms as suggested in many studies. The test results are given in Table 3 and 4. In tables, we show the accuracy and AUC values for algorithms.

We've observed degradation in RF performance when YATSI applied and noticed that adding unlabeled data diminished the classification performance of RF for software fault prediction. For example, AUC value of RF is 0.74 and AUC value of YATSI (RF) is 0.71 for KC2 dataset with 5% labeled modules. Degradation for AUC value points out the degradation of the classifier performance. Driessens et al. [13] showed that YATSI often improves the performance of the base classifier, but RF loses some of its accuracy with YATSI algorithm. However, their study did not focus on unbalanced datasets. This study shows new evidence which points out the degradation in RF performance when used in first stage of YATSI for unbalanced datasets. According to our experiments and Cozman et al. [32], very optimistic ideas to use unlabeled data should be changed in machine learning community.

Our empirical results demonstrated that YATSI improves the performance of AIRS based software fault prediction model. If you compare AUC of AIRS with AUC of YATSI (AIRS), you can easily see that AUC becomes larger with YATSI. This shows that the performance of AIRS improves. For example, AUC of AIRS algorithm is 0.65 and AUC of YATSI (AIRS) is 0.76 for KC2 dataset with 5% labeled modules. Increase of AUC points out the improvement of the performance. Performance of RF is better than AIRS and AIRS based YATSI algorithm. We need to improve the performance of AIRS based YATSI algorithm for semi-supervised software fault prediction. Supervised learning approach (Random Forests) provided better results than semi-supervised learning approach (YATSI) for this study. Furthermore, more labels usually lead to better results as in Driessens et al.'s study [13]. Therefore, 20% training set and 80% test set with RF is the best combination for this semi-supervised software fault prediction problem.

Table 3. Performance results of algorithms on project JM1

Classifiers	Dataset	% labeled	Accuracy	AUC
AIRS	JM1	5	72.41	0.55
YATSI(AIRS)	JM1	5	73.89	0.59
RF	JM1	5	78.74	0.64
YATSI(RF)	JM1	5	77.95	0.61
AIRS	JM1	10	72.53	0.56
YATSI(AIRS)	JM1	10	75.39	0.62
RF	JM1	10	79.28	0.66
YATSI(RF)	JM1	10	78.66	0.63
AIRS	JM1	20	72.44	0.56
YATSI(AIRS)	JM1	20	76.62	0.63
RF	JM1	20	79.75	0.68
YATSI(RF)	JM1	20	80.16	0.65

Table 4. Performance results of algorithms on KC2, CM1, and PC1 datasets

Classifiers	Dataset	% labeled	Accuracy	AUC
AIRS	KC2	5	78.69	0.65
YATSI(AIRS)	KC2	5	79.13	0.76
RF	KC2	5	80.26	0.74
YATSI(RF)	KC2	5	79.88	0.71
AIRS	KC2	10	77.53	0.67
YATSI(AIRS)	KC2	10	79.19	0.73
RF	KC2	10	81.34	0.78
YATSI(RF)	KC2	10	81.09	0.75
AIRS	KC2	20	77.50	0.69
YATSI(AIRS)	KC2	20	80.24	0.77
RF	KC2	20	82.22	0.79
YATSI(RF)	KC2	20	82.16	0.77
AIRS	CM1	5	87.93	0.53
YATSI(AIRS)	CM1	5	87.40	0.55
RF	CM1	5	88.35	0.59
YATSI(RF)	CM1	5	87.63	0.56
AIRS	CM1	10	83.34	0.54
YATSI(AIRS)	CM1	10	85.29	0.59
RF	CM1	10	88.68	0.63
YATSI(RF)	CM1	10	87.72	0.58
AIRS	CM1	20	82.10	0.55
YATSI(AIRS)	CM1	20	85.43	0.61
RF	CM1	20	88.59	0.65
YATSI(RF)	CM1	20	88.97	0.60
AIRS	PC1	5	88.25	0.54
YATSI(AIRS)	PC1	5	89.77	0.59

Table 4. (continued)

RF	PC1	5	92.32	0.65
YATSI(RF)	PC1	5	91.73	0.58
AIRS	PC1	10	88.39	0.54
YATSI(AIRS)	PC1	10	90.69	0.63
RF	PC1	10	92.46	0.68
YATSI(RF)	PC1	10	92.18	0.63
AIRS	PC1	20	89.17	0.57
YATSI(AIRS)	PC1	20	91.53	0.69
RF	PC1	20	92.94	0.72
YATSI(RF)	PC1	20	92.91	0.67

6 Conclusions and Future Work

Supervised learning approaches can not build powerful models with very limited fault data. Therefore, unlabeled software modules should be used together with labeled data during learning in this study. AIRS based YATSI approach was presented for building semi-supervised software fault prediction problem. The datasets which have been used in this study are JM1, KC2, PC1, and CM. Our experiments answered to our five research questions clearly. The answers to these questions are given below:

- 1) YATSI improves the performance of AIRS for semi-supervised fault prediction.
- 2) YATSI does not improve the performance of RF.
- 3) RF provides the best performance for semi-supervised software fault prediction.
- 4) Performance improves when percentages of labeled modules increase.
- 5) YATSI does not always improve the performance of classifiers.

This paper makes a number of contributions: First, as a new model, we propose AIRS based YATSI classification system for semi-supervised software fault prediction. According to our literature survey, this is the first semi-supervised algorithm which uses Artificial Immune Systems paradigm. Second, we show evidence which points out the degradation in RF performance when used in first stage of YATSI for unbalanced datasets. Third, we show that YATSI improves the performance of AIRS algorithm for unbalanced datasets. Forth, we present that more labels lead to better results for semi-supervised software fault prediction. Last, semi-supervised learning approaches are applied for software fault prediction with limited data as the second attempt after Seliya et al.'s study [12]. For the future, we plan using feature reduction techniques prior to learning step to improve the performance of AIRS based YATSI algorithm. We plan using correlation-based feature selection algorithm because it improved the performance of AIRS during our Fault Prediction Research Program. We plan using different distance functions instead simple Euclidean distance. Furthermore, co-training and self-training approaches can be used to improve the proposed algorithm.

Acknowledgement

This project is supported by The Scientific and Technological Research Council of TURKEY (TUBITAK) under Grant 107E213. The findings and opinions in this study belong solely to the authors, and are not necessarily those of the sponsor.

References

1. Tian, J.: *Software Quality Engineering: Testing, Quality Assurance, and Quantifiable Improvement*. John Wiley and Sons Inc., Hoboken (2005)
2. http://en.wikipedia.org/wiki/Source_lines_of_code#_note-1 (Retrieved on 06-10-2007)
3. <http://www.macintouch.com/specialreports/wwdc2006/> (Retrieved on 2007-10-06)
4. <http://www.linuxdevices.com/news/NS9334092346.html> (Retrieved on 2007-10-06)
5. Northrop, 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 Software Challenge of the Future*. Carnegie Mellon University, Pittsburgh (2006)
6. <http://www.ismwv.com> (Retrieved on 2007-10-06)
7. Khoshgoftaar, T.M., Seliya, N.: Tree-based Software Quality Models for Fault Prediction. In: *Proc. 8th Intl. Software Metrics Sym.*, Canada, pp. 203–214 (2002)
8. Chidamber, S.R., Kemerer, C.F.: A Metrics Suite for Object-Oriented Design. *IEEE Trans. on Software Eng.* 20(6), 476–493 (1994)
9. Sayyad, S.J., Menzies, T.J.: *The PROMISE Repository of Software Engineering Databases*. University of Ottawa, Canada (2005), <http://promise.site.uottawa.ca/SERepository>
10. Catal, C., Diri, B.: Software Fault Prediction with Object-Oriented Metrics Based Artificial Immune Recognition System. In: *8th Intl. Conf. on Product Focused Software Process Improvement*, pp. 300–314. Springer, Latvia (2007)
11. Zhong, S., Khoshgoftaar, T.M., Seliya, N.: Unsupervised Learning for Expert-Based Software Quality Estimation. In: *Proc. 8th Intl. Symp. on High Assurance Systems Engineering*, Tampa, FL, USA, pp. 149–155 (2004)
12. Seliya, N., Khoshgoftaar, T.M., Zhong, S.: Semi-Supervised Learning for Software Quality Estimation. In: *Proc. 16th IEEE Intl. Conf. on Tools with Artificial Intelligence*, Boca Raton, FL, pp. 183–190 (2004)
13. Driessens, K., Reutemann, P., Pfahringer, B., Leschi, C.: Using Weighted Nearest Neighbor to Benefit from Unlabeled Data. In: *Proc. 10th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 60–69 (2006)
14. Ma, Y., Guo, L., Cukic, B.: A Statistical Framework for the Prediction of Fault-Proneness. In: *Advances in Machine Learning Application in Software Eng.* Idea Group Inc. (2006)
15. Guo, L., Ma, Y., Cukic, B., Singh, H.: Robust Prediction of Fault-Proneness by Random Forests. In: *Proc. 15th Intl. Symp. on Software Reliability Eng.*, Brittany, France, pp. 417–428 (2004)
16. Witten, I.H., Frank, E.: *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann (2005)
17. <http://www.rulequest.com/see5-info.html> (Retrieved on 2007-10-06)

18. Evett, M., Khoshgoftaar, T., Chien, P., Allen, E.: GP-based Software Quality Prediction. In: Proc. 3rd Annual Genetic Programming Conference, San Francisco, pp. 60–65 (1998)
19. Khoshgoftaar, T.M., Seliya, N.: Software Quality Classification Modeling Using The SPRINT Decision Tree Algorithm. In: Proc. 4th IEEE International Conference on Tools with Artificial Intelligence, Washington, pp. 365–374 (2002)
20. Thwin, M.M., Quah, T.: Application of Neural Networks for Software Quality Prediction Using Object-Oriented Metrics. In: Proc. 19th International Conference on Software Maintenance, Amsterdam, The Netherlands, pp. 113–122 (2003)
21. Menzies, T., Greenwald, J., Frank, A.: Data Mining Static Code Attributes to Learn Defect Predictors. *IEEE Transactions on Software Engineering* 33(1), 2–13 (2007)
22. Guo, L., Cukic, B., Singh, H.: Predicting Fault Prone Modules by the Dempster-Shafer Belief Networks. In: Proc. 18th IEEE International Conference on Automated Software Engineering, pp. 249–252. IEEE Computer Society, Montreal (2003)
23. El Emam, K., Benlarbi, S., Goel, N., Rai, S.: Comparing Case-based Reasoning Classifiers for Predicting High Risk Software Components. *Journal of Systems and Software* 55(3), 301–320 (2001)
24. Yuan, X., Khoshgoftaar, T.M., Allen, E.B., Ganesan, K.: An Application of Fuzzy Clustering to Software Quality Prediction. In: Proc. 3rd IEEE Symp. on Application-Specific Systems and Software Eng. Technology, vol. 85. IEEE Computer Society, Washington (2000)
25. Catal, C., Diri, B.: Software Defect Prediction using Artificial Immune Recognition System. In: IASTED Intl. Conf. on Software Engineering, Innsbruck, Austria, pp. 285–290 (2007)
26. Huang, T.M., Kecman, V.: Performance Comparisons of Semi-Supervised Learning Algorithms. In: Proc. Workshop on Learning with Partially Classified Training Data, Intl. Conf. on Machine Learning, Germany, pp. 45–49 (2005)
27. Zhu, X.: Semi-supervised learning literature survey (Technical Report 1530). University of Wisconsin-Madison (2005), http://www.cs.wisc.edu/~jerryzhu/pub/ssl_survey.pdf
28. Chapelle, O., Schölkopf, B., Zien, A.: *SemiSupervised Learning*. MIT Press (2006)
29. Scudder, H.J.: Probability of Error of Some Adaptive Pattern-Recognition Machines. *IEEE Trans. on Information Theory* 11, 363–371 (1965)
30. Fralick, S.C.: Learning to Recognize Patterns without a Teacher. *IEEE Trans. on Information Theory* 13, 57–64 (1967)
31. Agrawala, A.K.: Learning with a Probabilistic Teacher. *IEEE Trans. on Information Theory* 16, 373–379 (1970)
32. Cozman, F.G., Cohen, I., Cirelo, M.C.: Semi-supervised Learning of Mixture Models. In: Intl. Conference on Machine Learning, Washington, USA, pp. 99–106 (2003)
33. Baluja, S.: Probabilistic Modeling for Face Orientation Discrimination: Learning from Labeled and Unlabeled Data. In: Neural Infor. Proc. Syst., Colorado, USA, pp. 854–860 (1998)
34. Nigam, K., McCallum, A.K., Thrun, S., Mitchell, T.: Text Classification from Labeled and Unlabeled Documents using EM. *Machine Learning* 39, 103–144 (2000)
35. Miller, D.J., Uyar, H.S.: A Mixture of Experts Classifier with Learning based on Both Labeled and Unlabelled Data. In: Neural Infor. Proc. Systems, Colorado, USA, pp. 571–577 (1996)
36. Goldman, S., Zhou, Y.: Enhancing Supervised Learning with Unlabeled Data. In: 17th Int. Joint Conf. on Machine Learning, Stanford, pp. 327–334 (2000)

37. Bennett, K.P., Demiriz, A.: Semi-supervised Support Vector Machines. In: Proc. Advances in Neural Information Processing Systems, pp. 368–374. MIT Press, Cambridge (1999)
38. Cozman, F.G., Cohen, I.: Unlabeled Data can Degrade Classification Performance of Generative Classifiers. In: Florida Art. Intel. Research Society, Florida, pp. 327–331 (2002)
39. Shahshahani, B.M., Landgrebe, D.A.: The Effect of Unlabeled Samples in Reducing the Small Sample Size Problem and Mitigating the Hughes Phenomenon. *IEEE Trans. on Geoscience and Remote Sensing* 32, 1087–1095 (1994)
40. Bruce, R.: Semi-supervised Learning using Prior Probabilities and EM. In: IJCAI Workshop on Text Learning, pp. 17–22 (2001)
41. Elworthy, D.: Does Baum-Welch Re-estimation Help Taggers? In: 4th Conf. on Applied Natural Language Processing, Stuttgart, Germany, pp. 53–58 (1994)
42. Vapnik, V., Chervonenkis, A.: *Theory of Pattern Recognition*, Nauka, Moscow (1974)
43. Yarowsky, D.: Unsupervised Word Sense Disambiguation Rivaling Supervised Methods. In: Proc. 33rd Ann. Meeting of the Assoc. for Comput. Linguistics, pp. 189–196. Cambridge (1995)
44. Blum, A., Mitchell, T.: Combining Labeled and Unlabeled Data with Co-Training. In: Proc. 11th Annual Conf. on Computational Learning Theory, Wisconsin, pp. 92–100 (1998)
45. Nigam, K., Ghani, R.: Analyzing the Effectiveness and Applicability of Co-training. In: Ninth Intl. Conf. on Information and Knowledge Management, Washington, pp. 86–93 (2000)
46. Joachims, T.: Transductive Inference for Text Classification using Support Vector Machines. In: Proc. Intl. Conference on Machine Learning, Slovenia, pp. 200–209 (1999)
47. Blum, A., Chawla, S.: Learning from Labeled and Unlabeled Data using Graph Mincuts. In: Proc. 18th Intl. Conference on Machine Learning, Massachusetts, USA, pp. 19–26 (2001)
48. Zhu, X., Ghahramani, Z., Lafferty, J.: Semi-supervised Learning using Gaussian Fields and Harmonic Functions. In: 20th Intl. Conf. on Mach. Learning, Washington, pp. 912–919 (2003)
49. Watkins, A.: AIRS: A Resource Limited Artificial Immune Classifier, Master Thesis, Mississippi State University (2001)
50. Timmis, J., Neal, M.: Investigating the Evolution and Stability of a Resource Limited Artificial Immune Systems. In: Genetic and Evo. Compt. Conf., Nevada, pp. 40–41 (2000)
51. De Castro, L.N., Von Zuben, F.J.: The Clonal Selection Algorithm with Engineering Applications. In: Genetic and Evolutionary Computation Conference, pp. 36–37 (2000)
52. Brownlee, J.: Artificial Immune Recognition System: A Review and Analysis, Technical Report. No 1-02, Swinburne University of Technology (2005)
53. Jin, X., Bie, R.: Random Forest and PCA for Self-Organizing Maps based Automatic Music Genre Discrimination. In: Intl. Conference on Data Mining, Las Vegas, pp. 414–417 (2006)
54. http://en.wikipedia.org/wiki/Random_forest (Retrieved on 2007-10-06)
55. Bradley, A.P.: The use of the Area under the ROC Curve in the Evaluation of Machine Learning Algorithms. *Pattern Recognition* 30, 1145–1159 (1997)
56. Ling, C.X., Huang, J., Zhang, H.: AUC: A Better Measure than Accuracy in Comparing Learning Algorithms. In: Canadian Conference on Artificial Intelligence, pp. 329–341 (2003)

Big Improvements with Small Changes: Improving the Processes of a Small Software Company

Anu Valtanen and Jarmo J. Ahonen

University of Kuopio, Department of Computer Science, P.O.B 1627,
FI-70211 Kuopio, Finland

anu.valtanen@uku.fi, jarmo.ahonen@uku.fi

<http://www.cs.uku.fi>

Abstract. Majority of software companies are small. They have understood that it is crucial for their business to improve their software processes but they often do not have the knowledge and resources to do it. In this paper one way of introducing a process culture and improving the processes of a small company is presented. The problems that a small company has with its efforts towards better processes are also discussed and simple but working solutions to them are introduced.

1 Introduction

Majority of software companies are small [1]. For example in Finland, all companies operating in both, data processing and software engineering fields, employ less than 50 people [2]. Small companies (SC's) have understood that it is crucial for their business to improve their processes and working methods but they usually do not have the knowledge or resources to do it. Software process improvement (SPI) has been researched quite a lot since the late 1980's when it was proven that the quality of a software system can be improved by improving the quality of the process used to develop it [2]. However, the research mostly concerns SPI for the larger companies and there is not that much information on the topic of SPI in smaller companies.

One of the main problems with smaller software companies is that they do not have a process culture. In a process culture people's customs and behaviours are influenced by process-oriented thinking and process management principles. The process is followed naturally. Process culture and process infrastructure are needed to institutionalize processes [3]. When process culture is missing, employees of the company do not have common and documented ways of work. This leads to a situation where everyone does things their own way, and soon ends up in a chaotic situation where no one is responsible. The most problematic challenges in software industry are how to keep up with the contracted schedules and how to produce quality software. It is shown that well defined and followed processes help solving these problems [4].

¹ <http://www.stat.fi> (2006).

The first step in setting up a process culture and improving software processes is modeling the current process [3]. This is the first brain teaser for a small company. Where and how to start? Good way to start is to pick an SPI model to support the efforts [3]. The problem is that most of the popular and widely known SPI models and techniques, for example CMMI [5] and SPICE [6], are designed for bigger companies who often have thousands of employees and therefore they are hard to apply in an SC. Also ISO's widely used quality standard ISO 9001 [7] is popular in SPI but suffers from same limitations as CMMI and SPICE.

There are some methods and models developed for the needs of smaller companies, for example [8] [9] [10]. However, most of these models suffer from some limitations. They are usually developed using some standard or known SPI model and applying them requires some knowledge of the model they are based on. Small companies often do not have that kind of knowledge and acquiring it demands a lot of resources. In smaller enterprises one of the main problems with SPI efforts is usually the lack of resources. To help small companies in their process improvement efforts there are different kinds of approaches established, for example ASPE-MSM [11] that combines the methods and models mentioned above.

The improvement work of one small company's processes is described in this article. The starting situation was that the company had the urge to improve their processes and ways of work but they did not know where to start. To get going with the improvements their current processes were modeled and then optimized. Modelings were carried out using a lightweight software process modeling method which is founded on modeling the processes through a wall-chart technique. The modeling technique does not require knowledge on any other SPI model. The PISKO technique is described in [12], applied in [13], [14], [15] and adapted in [11], [16].

Important issues, when choosing process improvement model for a small company, are researched to be that the model relates to company's business goals, focuses on most important software processes and of course, gives maximum value to the money invested. It is also perceived that the improvement technique should be flexible and easy-to-use [17]. All these requirements can be considered fulfilled in our technique of choice.

In this article the story of one small software company's efforts of creating process culture, modeling and improving their processes is told. The first part of the article presents the research and the problems with improving the processes of an SC. In the second part the improvement technique used is introduced. After that the target company and the results and notions made during the improvement project are presented. In the next section the resource needs of the improvement project presented here are discussed. The obtained results were interesting. The sole descriptions of the process and discussions about the ways of work had a significant impact on the efficiency of the process. Through the modelings and discussions the weak parts of the process were identified and the processes were simplified, properly documented and made easier to follow.

2 Research Problem

The goal of this research was to see if it is possible to create a process culture, meaning that the company has individual defined processes which are followed automatically inside the company, and improve the processes of a small company with an approach that requires minimal amount of resources from the target company and leaves widely known SPI models and methods for future reference. The starting situation being quite chaotic with the target company's working methods, the first step was to make their processes in such a condition that the actual improvement work would be possible. Besides of researching if efficient improvement could be done with a lightweight technique, using the expertise of the target company's employees, one of the main goals was to find a working way to take advantage of the company's small size while evolving the processes. In other words, the research questions were:

- How to create a process culture in an SC?
- How to make efficient improvement happen with SC's limited resources?

The current tendency to improve software processes is to use the widely known standards and models like ISO 9001 [7], CMMI [5] and SPICE [6]. Using these methods is not the best way to improve the processes of a small company because of their often too heavy structure. For example, most problematic issues while using ISO 9001 and CMMI's predecessor CMM in small companies are researched to be their lack of guidance and action instructions [18]. This presents a big problem when typical situation in an SC is that they do not have special SPI experts at their service. It either is not often possible for them to hire much outside help in form of researchers because of limited resources. Therefore the improvement projects are realistic only with lighter approaches.

Another problem, related to the lack of guidance and action instructions provided by the earlier mentioned methods, is that CMMI and SPICE do not commit on how to improve the processes that are at initial level. It is not possible to start the actual improvement work and strive for higher maturity levels of the process if there are problems with the ground stones of the process. Especially for a small company, it is hard to find a way to improve their processes from the initial level in practice [19]. It is also said that the existing process models do not help to benefit from the smallness of the companies [18]. Smallness usually means that the companies are more flexible, have faster reaction time and better communication inside the organization than their bigger competitors.

Because of the problems adapting the widely known standards and methods for the use of smaller companies, there are quite a many alternative approaches developed that are based on methods like SPICE and CMMI. Such approaches for SPI in smaller companies are for example Mares [10], PRISMS [8] and PEM [9]. All of these models require some knowledge of the models they are based on. Because the goal of this research was not to find out process's maturity levels, or other CMMI and SPICE related issues, but to model the process with as simple way as possible and then streamline it, these adapted methods were not suitable here.

In software process improvement the improvement method works as a plain framework for the improvements. The method always needs to be adapted to the needs of the target company [3]. In this sense, especially in small companies, it seems rational to pick a method that does not need lots of adapting and particular training but can be adopted to use right away, the PISKO technique was chosen as the template of the improvement project because of this. The lightweight and informal nature of the technique has turned out to be efficient in revealing the true nature and the problems of the software engineering process at hand, see eg. [13], [14], [15].

In addition to the usage of PISKO technique, the answers to the questions stated above were searched using action research as a research method. Action research is "an iterative process involving researchers and practitioners acting together on a particular cycle of activities, including problem diagnosis, action intervention, and reflective learning" [20]. In this case the researchers and the target company worked in very tight cooperation.

3 The Improvement Technique

The PISKO technique [12] is based on the idea that any attempt to model the actual software process should emphasize the opinion and experience of the experts of the target organization. The PISKO technique is designed to be easy to use for people who have no prior knowledge of it and it requires minimal amount of resources. Further analysis and evaluation of the technique is presented in [12]. The technique has six phases.

1. Model the process with wall-chart technique.
2. Analyze the gathered information and define the problems and points of improvement.
3. Create an electronic version of the process descriptions.
4. Inspect and approve the electronic versions.
5. Analyze and enhance the approved process descriptions.
6. Inspect the results.

3.1 The Wall-Chart Sessions

The wall-chart technique is the back bone of the method. During the wall-chart sessions the current process is modeled and the problems with it are identified. The sessions consist of meetings with approximately five experts from the target organization and two researchers. The experts should be the actual people who realize the process at hand every day, they also should have quite a long experience with the process. The experts are the ones who model the actual process and the researchers work as a chairman and the secretary of the sessions.

In the beginning of the sessions the technique is explained to the participants. First task is to resolve the main phases of the process. All this is done by discussing together and then attaching paper notes with different phases on the wall-chart. While deciding the phases it is important that all the participators

bring out their opinion and that in the end every one has a consensus on the phases and their names. After the different phases are identified they are connected in order to make the process's progress visible. The result of this first modeling phase is a wall-chart presenting the current state. The next task is to create electronic version of the descriptions made.

PISKO techniques resource needs are minimal. As a whole it takes about two whole working days to create the process descriptions with the target company's experts, half a working day to create the electronic versions and a couple of days to analyze the results. Applying the technique does not require any investments into expensive tools, all one needs is regular office supplies, paper and drawing pens. The electronic versions are also generated using usual spreadsheet programs. In addition to time consumed the most valuable resources needed assembling the process descriptions using PISKO are the opinions and the expertise of the people using the process.

3.2 Process Descriptions and Their Analysis

After the modelings the wall-chart is turned into electronic descriptions of the process. The descriptions are completed with textual descriptions of the process's phases. The goal is to make an understandable and extensive documentation of the process's current state.

After creating the electronic version the descriptions are inspected and approved by the representatives of the target company. The approved descriptions are analyzed by the researchers and then enhanced with the target company. When all this is done and the process descriptions are ready it is time to inspect the results and find out what the possible problems and points of improvement are. The end product of the process modelings is extensive report that includes the problems found and suggestions how to improve the software process.

4 The Target Company and the Current State of the Process

The company whose processes were modeled and improved is a small company that has less than twenty employees. The company is a traditional software house. The company's working environment is quite free and they have a low hierarchy. The general director of the company takes part in almost everything and because of this has a good knowledge of what is going on in the company. In this sense the freeness of the environment is a good thing. The employees feel free to talk about their work related problems and other issues. But when thinking in terms of the process culture the environment seems to be a bit too loose. When starting the process improvement efforts the work force did not seem to have any idea how their processes worked or even if they had any. General working guidelines did not exist in the reality. So the first step was to model their current process with PISKO modeling technique.

The company had never before thought of their work as a process. There was no existing process descriptions or quality manuals. So it was necessary to start

from the basics. To set up a process culture the company had to start thinking in terms of a process. It seemed that the employees of the company thought that the concept of process was something fancy and difficult. Even though the truth is that every company follows a process of their own kind and by optimizing it the work becomes easier.

The modeling sessions and conversations were a good starting point. Bringing stakeholders in the same room and laying out pencils and paper notes and asking the employees to tell how they start developing a new version of their software received a good reception. Through modeling sessions and free conversations the current situation was modeled.

Using the wall-chart technique twenty-one stages of the process excluding the ones with marketing and training were identified. Already at this point many points of improvement were easy to identify. The established process model can be seen in Figure 1.

The result of the modelings showed that the company's lack of process descriptions and definitions had led them to follow an overly complicated way of working. Because there was no common guidelines, employees of the company made the same tasks differently every time. This was the reason why the process had so many stages and iterations. Some of the complexity also resulted from differences between the work groups. Even in a small company, people in certain roles form groups, and develop their own ways of work if they do not have common and documented process to follow. When every group develops a different way to do their tasks it is sure to add complexity to the process.

After the wall-chart sessions the process model was enhanced with textual descriptions of the process and approved by the target company, according to the PISKO technique. By working together with the actual experts and realizers of the process the process descriptions were completed. At this point the current situation was clear to everyone and there was a consensus between the target company's employees and the researchers that the process should really be streamlined in order to make it usable.

5 Problems Found

The process starts from development meetings where the possible new features of software are deliberated and ends in dispatching the new version to customers. The phases between follow the typical progress of a software process, from design to implementation through testing and to delivery of the product. Even though the phases seemed clear there was a lot of iteration and jumping back and forth between different work stages.

The complete process turned out to be overly complicated and it was obvious it needed to be simplified to make it more efficient. The modelings showed that the company's lack of defined processes and complicated ways of work had led them to a circle of problems. There were problems with roles and responsibilities, meeting practices, decision making, documenting and testing.

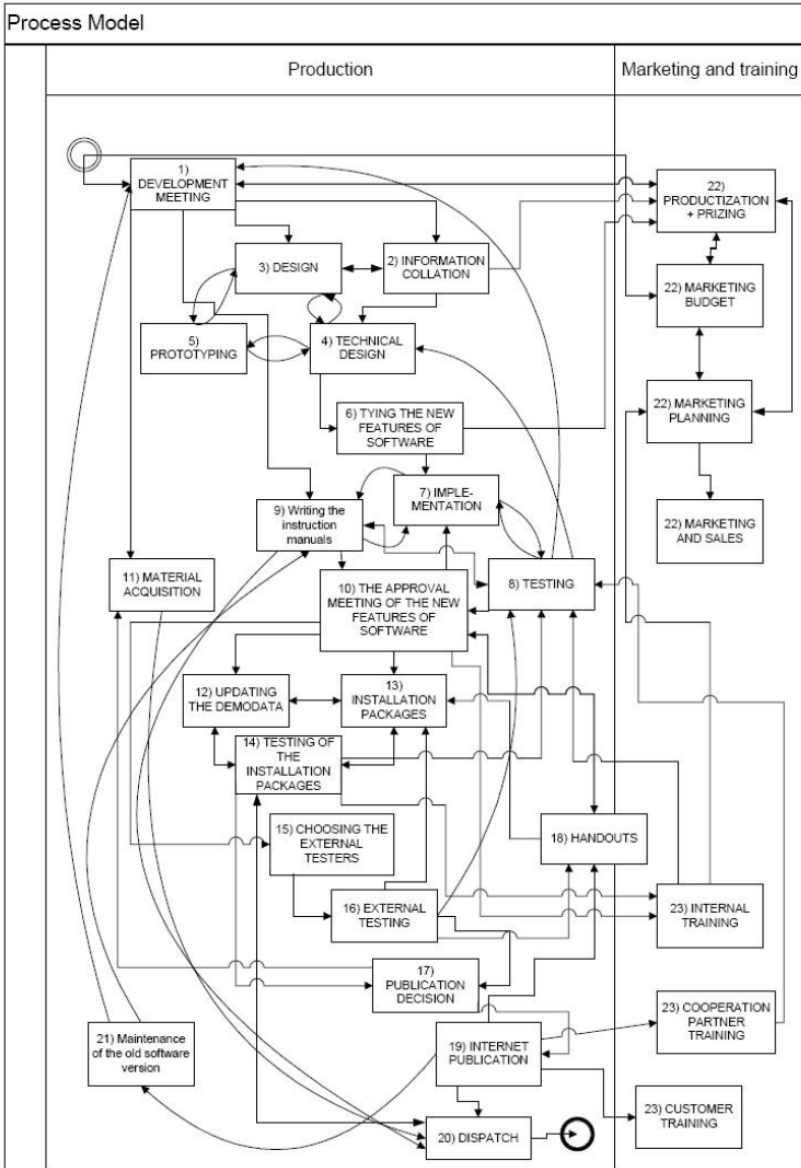


Fig. 1. The modeled process

Most serious problems found were the ones with documentation. In a small organization the information is often transferred in discussions and the documentation is neglected. This was the case here too. Not only the process documentation itself was neglected but also the basic documentation. When they

had a meeting they did not necessarily make a memo of it. So the organization also lacked good meeting practices. Their meetings were not carefully planned beforehand and because of this the important issues that should have been dealt were forgotten. The company also did not have the habit of choosing a clear chairman and secretary for their meetings.

The problems with documentation and meeting culture led to problems with decision making. The employees felt that they did not always stick to what they had decided and when there were no documents of the decisions it was not always clear what the decision really was. One of the most serious problems found was also that the company could not always hold to their schedules because it was not always clear what had been scheduled.

One more documentation related problem was found. The company had problems with testing its products. They had deficiencies in their knowledge of testing itself but it was also unclear what to test and when because there was no proper test plans. All the problems found while modeling and assessing the current state of the process were problems in the very ground stones of their working practices.

6 Improving the Process

After creating the process definitions and identifying the problems it was time to make improvement happen. The improvement efforts were continued by taking a closer look at the current process. What did the modeled phases really include? The text descriptions of the process were examined and the phases of the process compared. The purpose was to see if some of the phases were unnecessary and if some of them could be combined to others to avoid needless work. In addition to clarifying the process and making it more efficient the earlier found problems had to be excluded.

The improvement work was continued with the stakeholders of the company through the process descriptions made. It was important for them to take part in the improvement work so the improved process would be based on their opinions and expertise. This was a good way to approach the problems. The discussions held helped to trim the process's phases from twenty-one to ten. The main changes were that the actual planning phase was simplified and clarified. It was agreed that instead of six different work phases the planning of the new software version would be implemented in three phases. The work done inside these phases was specified and made clear to all of the stakeholders. Also the implementation phases were clarified in the same manner.

The goal, while creating the new process definitions, was to make descriptions that were extensive enough to cover the whole process but not too exhaustive so the process documentation would really be used as the basis of everyday work. So the process's primary documentation should cover only few pages. The idea was to create a poster-like definitions that could be spread via company's web pages or hung on the wall in order to be easily available for everyone. More specific descriptions were also created to support the actual process model to define what happens inside the phases. The revised process description can be seen in Figure 2.

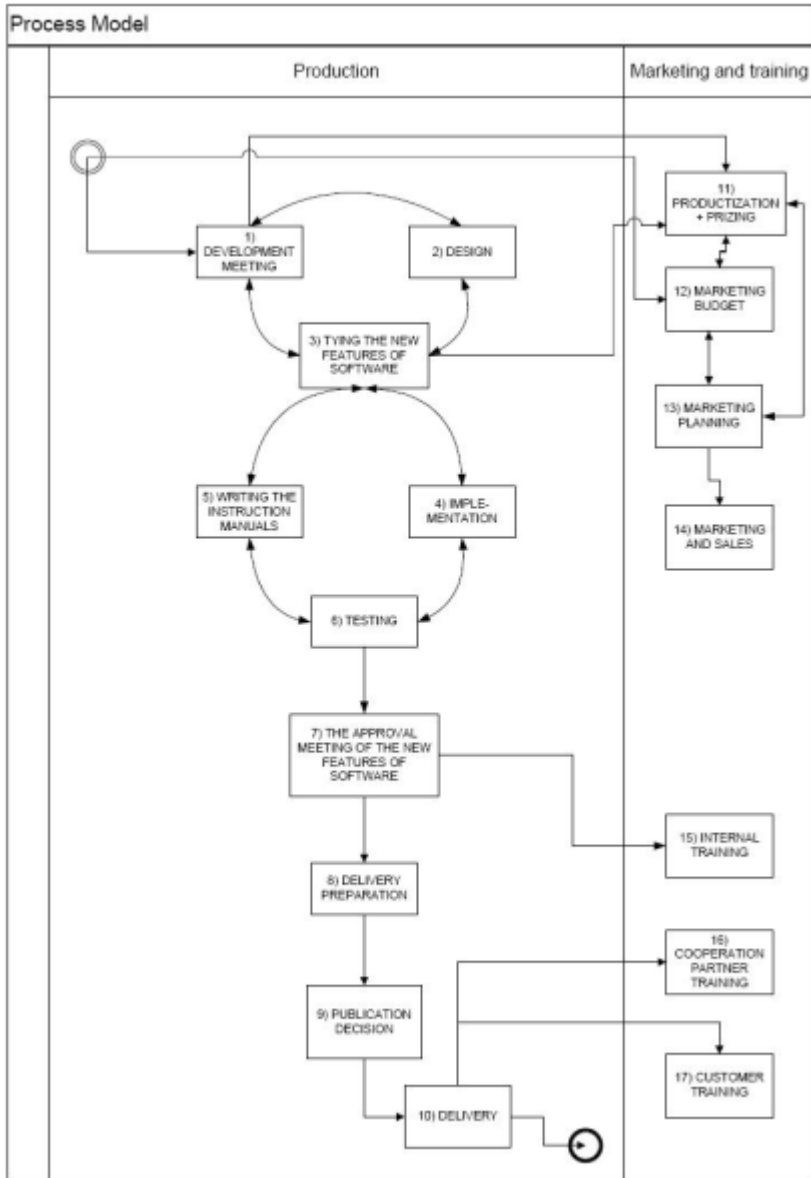


Fig. 2. The process after streamlining

To solve the problems found in the earlier phases of improvement efforts it was time to go back to the basics with the company's software engineering process. The roles and responsibilities of different parts of process had to be revisited,

meeting practices and decision making improved. Also the problematic issues with documenting and testing had to be dealt.

It was not enough just to simplify the process model. There had to be clear pre- and post-conditions in every phase to make it possible to follow the streamlined process. So as the first step of the improvement activities the conditions and their confirmation were decided. Many of these were different kinds of documents, signed decisions or other kinds of written products.

In addition to simplifying the process and making it easier to follow the earlier mentioned problems had to be solved. These problems seemed to be the reason why company's process was originally so complicated. When they did not have proper roles and responsibilities it directly affected the decision making and documenting. The main problem with testing process, that they were not always sure what should be tested and how, was also a documentation related problem.

Solving the problems was started from the ones with meeting practices. A proper meeting culture where every meeting had a clear agenda and convener was established, also memo templates to be used to document the meetings were created. Other documentation problems were also solved by creating documentation templates for the use of design, testing and so on. In addition to that different kinds of check-lists were created to control the phases and to make sure that the steps inside the phases really were properly implemented.

All of the improvement activities would be useless if no one took the responsibility to control their realization in practice. This being the situation, the company was advised to nominate a person in charge for every phase, whose duty is to make sure that the newly introduced process is followed; meetings are contrived, the documentation templates used and the check-lists filled the way it was agreed.

7 Resources Needed

One of the main goals of the improvement work presented here was the small amount of resources needed. The phases and the resource needs of the modeling work using PISKO technique are presented in Table 1.

As it can be seen from the Table 1, the resource needs were quite minimal. Creating the descriptions of the target company's current situation, defining the problems and points of improvement related to it and then streamlining the process took only 78 man-hours of the researchers and target company's mutual time and 117 man-hours in total, when counting the time that the researchers used creating the electronic descriptions and analyzing the situation. The time used in the actual modelings and meetings was 24 hours in total.

The practical improvement work took about 231 man-hours during four months, the time spent in the actual meetings being 45 hours in total. With the lightweight improvement actions, presented in Table 2, it was possible to get the new process going. The introduction of the new process was quite easy. The only training need was to introduce inspection protocol to the company.

The positive attitude among the target company's employees was a big help while taking the new process in use. It also helped that the new process was based

Table 1. The Resources used in the PISKO sessions

<i>PHASE</i>	p/TC^1	p/R^2	h/TC^3	h/R^4	$TOTALh^5$
1 Process modeling	5	2	20	8	28
2 Creating electronic descriptions and identifying the problems	0	2	0	6	6
3 Process model check-up	2	1	2	1	3
4 Electronic descriptions	0	1	0	1	1
5 Process model definition	5	1	10	2	12
6 Electronic descriptions	0	1	0	1	1
7 Enhancing the descriptions	3	1	6	2	8
8 Electronic descriptions	0	1	0	1	1
9 Approval of the descriptions and identifying the problems and points of improvement	5	2	20	8	28
10 Inspecting the results and planning improvement actions	5	2	20	8	28
TOTAL			78	39	117

Table 2. Resources used implementing the first improvement actions

<i>PHASE</i>	p/TC^1	p/R^2	h/TC^3	h/R^4	$TOTALh^5$
11 Creating check-lists	2	1	12	6	18
12 Inspection training	6	1	48	8	56
13 Inspecting the check-lists	5	1	20	4	24
14 Enhancing meeting practices	6	1	60	10	70
15 Creating document templates	2	1	30	15	45
16 Inspecting document templates	6	1	12	6	18
TOTAL			182	49	231

in the old one, the ways of work did not change dramatically and the changes made had a good reception because the employees themselves had agreed on taking the new practices in use.

The process improvement project, of which starting point the modelings described here were, was executed in 18 months during which the researchers were in tight co-operation with the target company. The monetary value of these improvements is hard to estimate, but the advantages brought by the improvement work were easily detectable while the improvement project was analyzed afterwards. The employees of the target company feel that the problems found

¹ The number of participants from the Target Company.

² The number of Researchers participating.

³ The man-hours the Target Company's personnel used.

⁴ The man-hours the Researchers used.

⁵ The man-hours used in total.

during the modelings of their process do not exist anymore. The employees also feel that planning their work has become easier when they have a concrete process to follow.

The improvement efforts with this company are still going on in form of a search of more mature and efficient processes. While working with new challenges with the company, it is self-evident that the improvement work described here was successful in creating process culture. The new process's terms are internalized and in every day use, the process descriptions are kept up-to-date and the newly introduced documentation and meeting practices are followed.

8 Discussion

In this paper, a simple way of introducing the process culture and improving the processes of a small company was presented. The problems that an SC has while attempting to improve its processes were also discussed and solutions to them proposed.

The improvement efforts described here proved that it is possible to obtain good results with small amount of resources while improving the processes of a small company. Similar results are reported in [13] and [11]. The use of a lightweight modeling technique like PISKO is a good starting point for the improvement efforts. The technique itself is flexible and does not demand a lot of resources so it fits in well with the needs of a smaller company's SPI.

The modelings proved that only presenting the concept of a process to the workforce has a big impact. Discussing and modeling the process at hand, using a wall-chart technique, works as an initiative of creating a process culture and changing the way of thinking. Discussions between the experts of the company help them outline different aspects of their work and during the wall-chart sessions not only the current state of the process is modeled but many good methods are transferred between the participants.

The key ingredients while improving the processes of a small company are the enthusiasm of the employees of the target company, planning and executing the improvements together with the experts of the company and taking care that someone is really responsible of making improvement happen, not only while the improvement project first takes place but also in the future.

By using a simple improvement method like PISKO one does not have to use limited improvement resources to train the personnel of the target organization. Quite the contrary, it is possible to start modeling the process's current state right away. After the modelings it is easy to identify the problems in the process and go on excluding them because the employees of the target company have taken part in the improvement work from the very beginning and are aware of the current situation. Even though it is clear that it is not possible to improve any process if the current situation is not known, it came almost as a surprise how big of a change it is possible to create just by making the process visible to its

realizers and then taking the next step to optimizing the phases in collaboration with the target company.

The modelings proved that the target company's processes were at initial state. The problems found were in the basics of the software process. The problems with roles and responsibilities, meeting practices, decision making, documenting and testing were due to inadequate formality of work. Once detected all of the problems were quite easy to solve with small improvements.

While solving the problems it became obvious that the documentation plays an important role also in a smaller software company even though it is often neglected. Even with only a few people, it is not enough just to transfer the information orally, the written documents are also needed to make the process successful. In our case most of the problems in the process were due to inadequate information transfer. Almost all these problems were solvable through documentation.

One of the main goals of these improvement efforts was to set up a process culture. To create a process culture, the company needs a process to follow, and to have a process means that some formality is required. At this point there was a conflict with our other goal, keeping the advantages of the smallness in mind. As mentioned earlier the freeness of the target company's working environment is both an advantage and disadvantage and too formal ways of work might kill the advantages brought by it. During all the improvements made, especially while optimizing the process and deciding on the pre- and post-conditions of the process's phases, it was necessary to keep in mind the advantages that the small size of the company brings and create a process that supports it. The flexibility of the PISKO technique supports this goal well. It was also a big help that the target company's employees had taken part in the improvement efforts from the very beginning. This way it was possible to create a process that really meets their needs and is easily adjustable for the future's changing requirements.

Modeling technique pretty similar to PISKO, is applied in [16] and the researchers end up recommending involving the developers in creating process guides. It is easy to agree with this in the light of our research. The introduction of the new process was quite easy in the target company. There was no special training needed and the positive attitude among the target company's employees was a big help while taking the new process in use. The implementation of the new process was easy probably because the users of the process had taken part in creating it. It also helped that the new process was based in the old one, the ways of work did not change dramatically so there was no significant problems with resistance to change.

The small amount of resources needed in the improvement work and the clearly visible advantages of having defined processes convinced the target company to keep improving their processes further. Now that there is a process culture created, the basic problems of the process fixed and the process optimized in the target company it is possible to start aiming towards even more efficient ways of work. In addition of making sure that the process is followed and kept up to date, the future work with the company consists of introducing

new working methods, for example in the form of enhancing their testing skills, that make it possible to keep improving their process and making their work even more profitable.

References

1. Richardson, I., Gresse von Wangenheim, C.: Guest Editors' Introduction: Why are Small Software Organizations Different? In: *Software*, vol. 24, pp. 18–22. IEEE, Los Alamitos (2007)
2. Humphrey, W.S.: *Managing the Software Process*. Addison-Wesley Professional, Reading (1989)
3. Zahran, S.: *Software Process Improvement: Practical Guidelines for Business Success*. Addison Wesley Professional, Reading (1998)
4. Fantina, R.: *Practical Software Process Improvement*. Artech House, Inc., Norwood (2005)
5. Chrissis, M., Konrad, M., Shrum, S.: *CMMI Guidelines for Process Integration and Product Development*. Addison-Wesley, Reading (2007)
6. El Emam, K., Drouin, J.-N., Melo, W.: *SPICE The Theory and Practice of Software Process Improvement and Capability Determination*. IEEE Computer Society Press, Los Alamitos (1998)
7. Bamford, R.: *ISO 9001:2000 for Software and Systems Providers: An Engineering Approach*. Auerbach Publishers, Incorporated (2003)
8. Allen, P., Ramachandran, M., Abushama, H.: PRISMS: an approach to software process improvement for small to medium enterprises. In: *Proceedings of Third International Conference on Quality Software*, pp. 211–214 (2003)
9. Trudel, S., Lavoie, J.-M., Par, M.-C., Suryan, W.: PEM: The small company-dedicated software process quality evaluation method combining CMMI and ISO/IEC 14598. *Software Quality Journal*, Springer Link 14, 7–23 (2006)
10. Gresse von Wangenheim, C., Anacleto, A., Salviano, C.F.: Helping Small Companies Assess Software Processes. *IEEE Software* 23, 91–98 (2006)
11. Wangenheim, C.G., Weber, S., Hauck, J.C.R., Trentin, G.: Experiences on establishing software processes in small companies. In: *Information and Software Technology*, vol. 48, pp. 890–900. Elsevier, Amsterdam (2006)
12. Ahonen, J.J., Forsell, M., Taskinen, S.K.: A modest but practical software process modeling technique for software process improvement. *Software Process Improvement and Practice* 7, 33–44 (2002)
13. Savolainen, P., Sihvonen, H.-M., Ahonen, J.J.: SPI with Lightweight Software Process Modeling in a Small Software Company. In: Abrahamsson, P., Baddoo, N., Margaria, T., Messnarz, R. (eds.) *EuroSPI 2007*. LNCS, vol. 4764, pp. 71–81. Springer, Heidelberg (2007)
14. Ahonen, J.J., Junttila, T.: A case study on quality-affecting problems in software engineering projects. In: *Proceedings of IEEE International Conference on Software: Science, Technology and Engineering*. SwSTE 2003, pp. 145–153 (2003)
15. Ahonen, J.J., Junttila, T., Sakkinen, M.: Impacts of the Organizational Model on Testing: Three Industrial Cases. In: *Empirical Software Engineering*, vol. 9, pp. 275–296. Springer, Heidelberg (2004)
16. Dingsøy, T., Moe, N.B., Dybå, T., Conradi, R.: A Workshop-Oriented Approach for Defining Electronic Process Guides—A Case Study. In: *The 11th Norwegian Conference on Information Systems* (2004)

17. Richardson, I.: SPI Models: What Characteristics are Required for Small Software Development Companies? *Software Quality Journal* 10, 101–114 (2002)
18. Demirors, O., Demirors, E.: Software Process Improvement in a Small Organization: Difficulties and Suggestions. In: Gruhn, V. (ed.) *EWSPT 1998*. LNCS, vol. 1487, pp. 1–12. Springer, Heidelberg (1998)
19. Harjumaa, L., Tervonen, I., Vuorio, P.: Using Software Inspection as a Catalyst for SPI in a Small Company. LNCS, pp. 62–75. Springer, Heidelberg (2004)
20. Avison, D., Lau, F., Myers, M., Nielsen, P.A.: *Action Research*. Communications of the ACM 42, 94–97 (1999)

Software Process Improvement Methodologies for Small and Medium Enterprises

Deepti Mishra and Alok Mishra

Department of Computer Engineering, Atılım University,
Incek, 06836, Ankara, Turkey
deepti@atilim.edu.tr, alok@atilim.edu.tr

Abstract. Today, the software industry is one of the most rapidly growing sectors and small software development companies play an important role in economy. Many such organizations have been interested in Software Process Improvement (SPI). It has been observed that the successful implementation of SPI methodologies is generally not possible within the context of small and medium-sized software enterprises (SMEs) because they are not capable of bearing the cost of implementing these software process improvement programs. Further the proper implementation of software engineering techniques is difficult task for SMEs as they often operate on limited resources and with strict time constraints. There are number of methodologies to address these issues. In this paper, various SPI methodologies for SMEs are discussed and compared. This will lead towards maturity of software process improvement in SMEs and also facilitates in development of automation tools for SPIs in future.

Keywords: process, software process improvement, software quality, small and medium enterprises, SME.

1 Introduction

The way with which we develop software impacts the quality of the software and hence software process is one of the most crucial factors in determining the quality of the software. A software process is a set of activities, together with ordering constraints among them, such that if the activities are performed properly and in accordance with the ordering constraints, the desired result is produced. The desired result is high quality software at low cost. As each software development project is an instance of the process it follows, it is essentially the process that determines the expected outcomes of a project [23]. Software processes play an important role in coordinating different teams in large organizations so that their practices don't grow out of touch with one another [14]. Ideally, these processes should combine the need for flexibility and creativity, but that balance is hard to achieve [17]. A vast majority of software producers, which have not yet implemented a methodology for software process improvement, are paying high costs of production and systems maintenance, and therefore being displaced from the global market, not being on the same competitiveness level than companies that possesses a process improvement method [21]. There are several models for software process improvement, such as the Capability

Maturity Model Integration (CMMI), the Software Process Improvement and Capability dEtermination (SPICE) and the ISO 9000 norms from the International Standardization Organization. These models provide quality patterns that a company should implement to improve its software development process [21]. Unfortunately, it has been observed that the successful implementation of such models is generally not possible within the context of small and medium-sized software organizations because they are not capable of bearing the cost of implementing these software process improvement programs [26, 53] and the proper implementation of software engineering techniques is difficult task for SMEs as they often operate on limited resources and with strict time constraints [53]. Dyba [14] indicated that SPI can be used as a competitive advancement strategy for both small and large organizations [14]. Today, the software industry is one of the most rapidly growing sectors and this situation stimulates especially the constant creation of small companies which play an important role in economy [53] and in the last few years, a great number of organizations have been interested in Software Process Improvement (SPI) [10]. A considerable amount of software is produced world-wide by SMEs ranging from 1 to about 50 employees [19]. In this context, German and Brazil software market of these companies was around 77% and 69% during 2001 [37]. Richardson [43] observed that there is need for small software companies in Irish sector to improve their software process. The term *small setting* has been defined as an organization or company of fewer than approximately 100 people, and a project of fewer than approximately 20 people [49]. As mentioned in the Software Engineering Institute Web site for small settings, a major aspect to be considered in these environments is that the amount of resources used to support a process improvement effort would be a large percentage of an organization's operating budget, [49]. Brodman and Johnson define a small organization as fewer than 50 software developers and a small project as fewer than 20 developers [24].

2 Related Works and Rationale of SPI in SMEs

Existing software engineering and organization development literature acknowledges that there are fundamental operational differences between small and large organizations [14]. Small organizations seem more concerned about practice, while large organizations seem more concerned about formal process [14]. Russ and McGregor [45] observed that software development process can be just as critical to a small project's success as it is to that of large one due to number of external dependencies per team member. They further argued that its goal is to produce the high quality and timely results for today's market without imposing a large overhead on a small project.

Larsen and Kautz [33] also viewed that these organizations are afraid of the initial expenses which they assume are large both with regard to direct costs for process assessment, training and tools, but also due to indirect costs for personal and time resources when implementing improvement actions. Kuvaja et al. [30] further supports that it is quite difficult for any SME to choose an improvement approach, and to apply it in their organization without help of external consultants or substantial investment in time of their software managers. Cultural issues like resistance to change from the employees or the management areas, who regard the extra work

required for quality assurance as a useless and complicated burden put on the developing team. According to Biro et al. [6] national culture also affects the process improvement methods. Kuvaja et al. [30] mentioned that main problem of the small companies is that they cannot afford to maintain substantial expertise of software process improvement within their companies, but they have to buy it from external sources. Further problem related to the lack of expertise is to find how to start the improvement and what experts to use. Due to budget constraints services of a consultant organization to improve the software quality is not possible, still the need for a good quality assurance program is becoming more evident, and managers are striving to achieve international quality standards that, in the long run, result in lower production cost [21]. According to Kautz [26] the software process improvement is rewarding and advantageous also for small organizations if it takes into account the peculiarities of such organizations. Dyba [14] also found empirically that small organizations implemented SPI as effectively as large organizations, and in turn, achieve high organizational performance. According to his study main lesson to be learned is that to implement SPI at least as effectively as their larger counterparts, small software organizations should capitalize on their relative strengths in employee participation and exploration of new knowledge. There are various approaches, languages and tools for process definition [1] however, are rarely applied in practice [11] specifically with small organization [39]. Further only few studies in the context of small software companies have been performed [29, 46, 47]. In order to get an edge in ever-growing highly competitive software development world, it is significant for an organization to regularly monitor the software process. It is important for an organization to continuously improve its software process on the basis of feedback from various stakeholders. It is also supported by Mintzberg [38] that for smaller organization where much of the work is coordinated through direct supervision and mutual adjustment, it is important to find a balance between these mechanisms and formal, defined and highly detailed documented procedures to facilitate organizational learning [40]. Despite the fact that even in the US most software producing units are comparably small and state a need for improvement [8], little is known about software process improvement in this kind of organization [26]. Kautz [26] further supported the view that even small organizations with little more than two developers can profit from some basic formal routines. According to his research project conclusion if procedures are defined, concisely described, tested and feedback from these tests can be used as feedback to improve the procedures and routines. According to Kuvaja [30] it is quite common understanding amongst the SMEs that full-scale assessment methods are only useful in large organizations and do not serve the SMEs appropriately. Dyba [14] found empirically that small organizations implemented SPI as effectively as large organizations, and in turn, achieve high organizational performance. Nevertheless, small software development teams can improve their software processes beneficially as well as large organizations [41, 13]. Therefore the objective of this paper is to present these software process improvement methodologies for SMEs from a comparative perspective. This will lead towards maturity of software process improvement and also facilitates in development of automation tools for SPIs which can be tailored according to the specific organization. It can also result in interesting empirical outcome and comparisons in SPI approaches among organizations.

The remainder of this paper is organized as follows: The following section discuss software process improvement methodologies for SMEs. Later, these methodologies are compared. Finally, the paper concludes with limitations and directions for future research in this area.

3 Software Process Improvement Models for SMEs

Any software process improvement plan requires a qualified statement about the current status of software development in the companies and a description of strengths and weaknesses identifying areas for improvement. On the basis of literature survey we have selected following five SPI methodologies which have been implemented in SMEs. Due to limited resources and the size of the organizations, an extensive, formal assessment of the software practices following defined comprehensive approaches like the Capability Maturity Model [42], the ISO9000-3 guidelines [22], the TickIT scheme [52], Bootstrap [31] and IDEAL [28] model was not considered to be necessary or appropriate in this context. It is also supported by Kautz [26]. Further MESOPYME objectives are similar to those of the IDEAL model [36] from the Software Engineering Institute (SEI).

Salient features of selected software process improvement methodologies for SMEs are discussed in this section.

3.1 A Methodology for Self-Diagnosis for Software Quality

This methodology for self diagnosis is based on concepts, goals and activities defined by Capability Maturity Model (CMM) which can be used by a small or micro organization as a part of internal audit plan before the official appraisal. It is difficult for SMEs to assess their current capabilities by using SCAMPI A (only method in CMMI product suite that can result in a rating) appraisal method because it takes longer and consume more resources. In order to gather this information related to the current processes of the organization, researchers have created 3 questionnaires [21]:

- The extended maturity questionnaire(EMQ)
- The Goals, Activities and Responsibilities Matrix(GAR)
- The Directed Questionnaire

The Extended Maturity Questionnaire (EMQ)

EMQ is based on the Maturity Questionnaire developed by SEI. The main difference between EMQ and maturity questionnaire developed by SEI is that every question has potential three answers (YES, NO, PARTIALLY ACHIEVED) instead of two (YES, NO). So, this questionnaire accurately represents organizations current states as some of the goals are only partially achieved and if the organization will use SEI questionnaire then it will result in NO.

The Goals, Activities and Responsibilities Matrix(GAR)

The success of a model based on CMM depends on the complete achievement of certain goals and commitments for every Key Process Areas (KPA). There is a close relationship among goals, activities and abilities, which are not that immediately

apparent from the 344 pages description of the CMM standard [9]. In order to facilitate the task of the software administrators a matrix is proposed. This matrix includes relationship between abilities (variables), activities (practices and sub-practices associated to each KPA), goals and commitments (objectives to achieve in each KPA) as well as the responsible individuals (The client, the requirement analyst, the software engineering group, the manager, the quality assurance group) for each KPA. GAR Matrix can be automated by means of an expert system.

The Directed Questionnaire

The last format of Self-Diagnosis Methodology is a direct questionnaire with which a lead auditor can construct a knowledge base. This questionnaire has the essence of the original Maturity Questionnaire from CMM but in this case each new question is generated based on the answer of the previous questions. So a new question may be directed to complement information obtained earlier, or to confirm such information. In any case, useless questions are discarded.

Evaluating the Result of the Self Diagnosis

The results obtained from the questionnaires answer the basic question: Are the Key Process areas required by CMM for a certain level achieved? For each KPA, there are four possible answers: The KPA is either fully achieved, partially achieved, not achieved, or it doesn't apply. The KPAs that are partially achieved or not achieved are the areas of opportunity for improvement and that should be part of an action plan.

3.2 Software Process Matrix (SPM) Model

This model helps the organization in finding the relative importance of software processes. For the high priority processes, the practices that need to be worked on are determined by Software Process Matrix (SPM). SPM is based on Quality Function Deployment (QFD). In QFD, the 'voice of the customer' is collected, and the relative importance of each customer requirement is measured. In the house of quality matrix, these requirements are used to identify design characteristics which have the greatest impact on customer requirements. Although QFD consists of many matrices, the main focus is often this matrix, as using it alone can have a significant effect on the product development process [16]. Using QFD, the software process model is treated as the customer where software processes are the customer requirements. These processes were identified from software process literature. The design characteristics are the practices which must be followed for processes to be successful. These practices were also identified from the software process literature.

A crucial part of the development of the software process matrix was to identify the relationships between processes and practices. Those which are explicitly mentioned in the literature were easily identified. Using expert opinions and various statistical techniques, other relationships between processes and practices were identified, resulting in the development and verification of the software process matrix which was then validated in the industry.

For a small company to use any software process model to their advantage, it is imperative that the effort expended is minimal. The SPM provides them with a generic section that has been completed previously and can be used in their company. A questionnaire is provided to assess the current performance, planned future performance

and importance to the company for every process. From the company's point of view, all they need to provide are the measurements for calculating the overall importance of the software process considering the following [43]:

- Current capability as assessed using a self-assessment questionnaire.
- Future capability as input from management.
- Importance of software process to the business.
- Competitive analysis
- Market leverage for company specific requirement e.g. ISO-certification.

Allowing management to choose whether or not to include figures for competitive analysis and market leverage allows flexibility within the model.

Practices with the highest values are the most important, and therefore it is suggested that these should be worked on first in the organization. From this, the priorities to be included in any software process improvement action plan are established and can help the organization to determine their improvement strategy. The complete SPM provides the organization with a ranked list of actions which can be input to their software process improvement strategy. This ranked list can be combined with cost figures and time-effective calculations thus taking these factors into account when determining the action plan for the organization.

3.3 An Approach for Software Process Establishment in Micro and Small Companies (ASPE-MS C)

An Approach for Software Process Establishment in Micro and Small Companies (ASPE-MS C) is defined by integrating and adapting existing approaches [2,4,5,12,32,34,48] to the characteristics of small software companies. The principal phases of the approach are:

Planning: In the beginning, the process establishment is planned on a high level. Later on, during strategic analysis, the plan is revised, completed and adapted in accordance to the decisions made.

Phase 1, Diagnosis: The objective of this phase is to contextualize the organization and to obtain a high-level snapshot of the actual software process in place. Such a baseline can be established through a software process assessment using, e.g. MARES [18], an ISO/IEC 15504 conformant process assessment method tailored to small companies.

Phase 2, Strategic analysis: The objective of this phase is to specify the scope and to prioritize candidate processes to be established based on the results of the diagnosis and in accordance with the organization's business and improvement goals. This can be done by using, e.g. an adaptation of the SWOT (Strengths/Weaknesses/Opportunities/Threats) analysis technique [25] relating the importance of processes and their assessed/estimated capability.

Phase 3, Definition: The objective of this phase is to define the selected software process(es) in form of a process guide in order to support process performers. Generally, the definition of the selected process(es) begins with the descriptive modeling of the actual process(es) in place. This activity is composed of a process familiarization phase and a

detailed elicitation phase [4]. During the process familiarization phase an overview of the software process and its general structure, interaction and sequence is obtained and documented, for example, in a process flow diagram. In a next step, roles, competencies and responsibilities related to each activity are identified.

Phase 4, Implementation: First, the evaluation of the defined process(es) is planned in parallel to their implementation. This includes the revision and/or definition of measures in order to monitor and determine the effectiveness and suitability of the process(es) and whether the expected benefits are achieved.

Monitoring & Control: The complete establishment of the process (es) is monitored and controlled. Therefore, data is collected and analyzed by the process engineer and assistant. If required, corrective actions are initiated and the plan is updated.

Post-mortem: Once a complete process establishment cycle is terminated, the process establishment approach is evaluated as a basis for continuous improvement. This is done by collecting and analyzing feedback from process performers, sponsor, and the process engineer and assistant in a feedback meeting or by questionnaires.

3.4 PRISMS: An Approach to Software Process Improvement for Small to Medium Enterprises [3]

PRISMS is an action research project, with a team of three researchers from Leeds Metropolitan University working alongside managers and developers in participating companies advising and assisting with the planning and implementation of software process improvement programmes, over a three year period.

The key features of the process are:

- The existing informal process is examined, and, if resources permit an explicit model is created.
- In the PRISMS programme the business goals are defined earlier by management. These goals drive much of the subsequent activity, especially the selection and prioritization of key process areas for improvement, and the selection of measurements.
- A consultation exercise is carried out, involving all members of development teams. A brainstorming session, and/or questionnaire-based survey help the developer's team to take ownership of the SPI programme, and to be involved in the programme from the earliest stage.
- A tailored version of the CMM assessment is carried out by members of the research team, primarily to help identify key process areas (KPAs) for improvement.
- Using these inputs the KPAs for improvement are identified and prioritized. The main criteria here should be the extent to which the KPAs are likely to contribute to the identified business goals. One company has found a weighted selection approach of the type described by Martin [35] to be useful. The process/practice matrix approach described by Richardson [44] could also be used.
- Measurements are defined as an integral part of the SPI planning process. Managers are generally keen to have more precise ways of tracking key

resource and quality indicators. The Goal Question Metric paradigm can be used to measure selected attributes based on the business goals defined for the SPI programme [7].

- The SPI plan is periodically reviewed, and there is provision to collect feedback from stakeholders.

Most important aspects of measurement for SPI programmes in smaller organization is that they should be simple to gather and interpret, and that they should be used in planning and decision making. Simple automation can help reduce the overhead associated with data collection and processing.

3.5 MESOPYME [10]

MESOPYME has been defined, taking into account a generic SPI model defined by ISPI [15] with four stages—whose objectives are similar to those of the IDEAL model [36] from the SEI. The key features of MESOPYME are as follows:

- **Stage 1: Commitment to improvement.** Its objective is to obtain the support of senior management to carry out the improvement project.
- **Stage 2: Software process assessment.** Its objective is to obtain strengths and weaknesses of the process assessed with respect to a software process model— CMM (Capability Maturity Model). From this assessment, processes (usually 1 to 3) to be improved are selected.
- **Stage 3: Improvement solution.** Its objective is to provide the needed infrastructure to carry out improvement (in selected processes), and to create the plan to follow in order to define and implement improvement in these selected processes. The improvement solution stage is performed through the application of a generic set of components that we have called an Action Package. An Action Package is a general solution to a particular software process area that should be customized to a company, taking into accounts its business goals and assessment results. An action package is implemented in some selected pilot projects.
- **Stage 4: Institutionalize.** Finally, improvement must be institutionalized.

4 Discussion

As these SPI methodologies are divergent in characteristics, therefore it is required to find out some significant but common attributes so that we can find a comparative view of all selected SPI approaches. Kautz et al. [28] concluded in their findings that first lesson for small organizations, which wish to perform improvement activities, is that it makes sense to use a structural model to organize the process. They further suggested that the second lesson is that model should be adjusted to the particular conditions of the organizations and the third lesson is that it makes sense to perform the improvement activities as a project with clearly assigned and documented roles, responsibilities and resources. Beyond the adjustment of general models (which is in fact a base for these approaches), Kautz [27] points out the significance of factors to be studied further like management support and commitment, project planning and

organization, education and training, assessment, monitoring and evaluation, staff involvement, support and knowledge transfer by external consultants, usability and validity of the introduced changes and cultural feasibility for process improvement in software SMEs. As SMEs have limited budgets and resources, following factors are important for them before selecting any SPI model.

1. If it is based on already established SPI methods like CMM then it may be better in the long run. Although this factor is not important right now as achieving some specific CMM level is not the objective at present and SME cannot afford to achieve this in the present position. But later organization may grow and may wish to achieve a specific established method like CMM. If the SPI model they are choosing at present is based on for example CMM then it will be easy to switch.
2. There are two key questions: where am I and what needs to be improved? and how to improve it? If a SPI model answers both these questions successfully, then it is easier for the organization to use and implement it.
3. Whether it takes into consideration specific needs of the organization then it is better for the organization.
4. If it provides some flexibility to the organization like choice of different methods for assessment etc. then it is better. It is also supported by Glass [17] that these processes should combine the need for flexibility and creativity. Further Richardson [43, 44] found flexibility as significant characteristic for software process and included in her proposed model.
5. Whether it is continuous or staged? An organization may choose one over another. Continuous representation allows an organization to select the order of improvement that best meets the organization's business objectives and mitigates the organization's areas of risk. On the other hand, staged representation provides a proven sequence of improvements, beginning with basic management practices and progressing through a predefined and proven path of successive levels, each serving as a foundation for the next [50].
6. Involvement of software development team members from the starting is very important. Their views should be considered while deciding what needs to be improved? It may help in securing their confidence and commitment in SPI initiative. Otherwise they may resist SPI initiative later on.
7. Whether it requires SME's people, who will take part in SPI initiative, to have prior experience in this field. If it does, it may not be suitable as SMEs have difficulty in recruiting and retaining experienced staff.
8. Whether it requires the need to take the help of external expert. If this is the case, it might be difficult for the organization as they have to bear the extra cost.
9. Whether roles and responsibilities are clearly assigned to all people taking part in SPI initiative. Also, if they need training, it should be provided. Both these factors are important for any successful SPI initiative as mentioned by Kautz [27].
10. If a tool can be used for self assessment, it will be easier to assess the current status and to determine the areas, which needs to be improved. Additionally, more people can be involved during this phase without much substantial effort.
11. Data collection and evaluation is integral in any SPI initiative. It can be difficult for software practitioners to do this if an organization does not have special team

to do this task. Use of tool for this purpose can make the job of software practitioners easy in this case.

12. Sometimes origin of an SPI method is also important. A particular SPI method originated in a particular country is tested in the software development organizations of that country. Although due to the emergence of global standards in software development, organizations all over the world are similar to each other in terms of platforms, technical tools and other things they are using. Still cultural factors play an important role, and there one SPI initiative which was tested successfully in one country may not get equal success in another country. This is also supported by Biro et al. [6] that national culture affects the process improvement methods. Additionally, people who developed a particular SPI model may be available for helping the organizations situated in their country.

These models for SMEs are based on some existing methods like CMM, GQM, QFD etc. These approaches are adapted and simplified either by incorporating some additional questionnaires (in Self-diagnosis model) or matrix (in SPM model) or process guides (in ASPE-MS) or action packages (in MESOPYME) so that they can be used by these organizations.

One key point is that all methods except self diagnosis model considers business objectives of the organization while making the SPI plan. Moreover, these methods (excluding self diagnosis) are flexible enough that although methods for identifying and prioritizing areas of improvement are suggested but organizations can choose any other method also. Furthermore, organizations have the flexibility to select processes more important to them for SPI plan. These methods not only detects what needs to be improved but also provides the roadmap that how to improve it.

Software practitioners are involved from the beginning in both SPM and PRISMS method. They take active participations during self assessment. All practitioners' views, regarding which processes need to be improved, were taken into consideration.

As far as practitioner's knowledge level is concerned, Self-diagnosis and MESOPYME do not require much experience while other models need much knowledge and experience to assess current capabilities of the process. SMEs generally do not have people dedicated for quality work alone. A person has many roles in these organizations for example people who are doing software development are also responsible for SPI initiative. These individuals may or may not have experience dealing with SPI initiative so it may not be easier for them to use any of these models without the help of some external consultant.

These SPI models are specifically developed for SMEs as these organizations do not have the resources and cannot bear the cost to implement CMMI, SPICE etc. In this context it is important to note some outcomes for instance SPIRE results indicated that "of the small software development units who applied to be involved in SPIRE, 27% dropped out. The most common reasons given were resource or funding problems" [51]. Wieggers says [54], "the most common point of failure in SPI is lack of follow-through into action planning and action plan implementation." Also performance of these activities is expensive- yearly cost of improvement \$245,000 [20], and time consuming – a full process improvement cycle could take between 18 and 24 months [55]. Moreover, this is more difficult to perform in SMEs because they do not

Table 1. Comparison of various software process models for small and medium enterprises

SPI Models → Criteria ↓	Self-diagnosis	SPM Model	ASPE-MSC	PRISMS	MESOPYME
Based on	CMM	QFD	Many existing approaches	CMM and GQM	CMM
Key Question	Where am I? What should I do?	What needs to be improved? How to improve?	Where am I? What needs to be improved? How to improve?	Where am I? What needs to be improved? How to improve?	Where am I? What needs to be improved? How to improve?
What is new?	EMQ questionnaire for assessment. Also, relationship among abilities, activities, goals, and commitments for every KPA in matrix format is provided.	SPM (software process matrix) that identifies practices needed for software processes to be improved.	Iterative-Incremental approach for assessment, identification and implementation of SPI plan.	Adapting CMM by incorporating business objectives with the help of GQM paradigm	Emphasis on SPI implementation step with the help of action packages developed by problem domain experts.
Implementation details	Current situation of the company is assessed with EMQ questionnaire. This identifies KPA's which are partially achieved or not achieved at all to attain a particular CMM level. Once KPAs for improvement are identified, goals, and commitments for every KPA can be found out with the help of GAR matrix.	First prioritized list of processes for software process improvement according to the business objectives and other factors are made. Then a ranked list of actions is made with the help of SPM to improve above mentioned processes.	First diagnosis of current capabilities is done then prioritized list of candidate processes is made according to the diagnosis, business objectives and improvement goals. Later these processes are defined in form of a process guide. Thereafter implemented with the help of their process guide and evaluated continuously.	First current process is examined and assessed. Then KPAs for improvement are identified based on current capabilities, business goals given by the management and after consultation with developers. Later process improvement plan is made and implemented.	Firstly current process is assessed and then action package for each process area consisting of action plan, infrastructure needed, techniques, tools, metrics etc., is developed according to the business goals and assessment results. Later this action package is implemented in some pilot projects. Finally improvement is institutionalized.
Flexibility	Not flexible. Defined methods and tools. Elimination is not preferred.	Flexible. Company is not required to include all factors of measurement of overall importance of software processes. Only processes important for company are considered for improvement.	Flexible. Methods for assessing current capabilities and to prioritize processes are suggested but organization can use other methods also. Only processes important for company need to be considered for improvement.	Flexible. Methods to identify and prioritize KPAs for improvement are suggested but organizations can use other methods also.	Flexible. Can be tailored to the specific need of an organization.
Continu./ Stage	Staged	Continuous	Continuous	Continuous	Continuous
Involvement of Software deve. Team members from the very beginning	Not mentioned. One auditor does the self assessment	Yes. They give information about which processes needs to be improved.	Only one person (process engineer or assistant PE) is involved during assessment. Others are involved during implementation.	Yes. They give information about which processes needs to be improved.	Not mentioned. It is not clear that who decides which process needs to be improved? They are involved during implementation.

Table 1. (continued)

SPI Models → Criteria ↓	Self-diagnosis	SPM Model	ASPE-MSC	PRISMS	MESOPYME
Practitioner's knowledge level	Doesn't need much experience. Easy to use.	Needs considerable experience to assess current capabilities and planned future performance of a software process and importance of process to company.	Needs considerable experience to assess current capabilities, process prioritization, and definition of implementation. If the organization lacks required expertise, an external expert (process engineer) is used with one employee acting as assistant to enable knowledge transfer.	Needs considerable experience to assess current process, identify KPAs for improvement. Also development and implementation of process improvement plan requires experienced person.	Doesn't need much experience as action packages are developed by domain experts according to the organizations' business goals and current capabilities.
Roles, responsibilities and training	Not mentioned	Only quality assurance engineer is responsible for SPI. Other roles are not mentioned.	Roles, competencies and responsibilities related to each activity are defined. Assistant process engineer is given training, if required.	Only role is process improvement champion who is responsible for implementation of improvement actions. Other roles are not mentioned.	Roles and responsibilities are established for every improvement initiative. Special training is given before implementation.
Tool for self assessment	Work in progress to develop expert system for self assessment that can act as virtual auditor.	Not mentioned	Not mentioned	Yes, based on modified and customizable version of questionnaire.	Not mentioned
Automated data collection for measurement and evaluation	Not applicable	Not mentioned	Not mentioned for data collection. Tool support for process selected for improvement.	Can be done if organization is ready to devote resources.	Not mentioned for data collection. Tool support for process selected for improvement.
Outcome	Actions, goals that needs to be achieved for software process improvement	Ranked list of actions for software process improvement.	Process guide for processes selected for improvement that contains entry/exit criteria, techniques and tools to use, examples, required infrastructure, checklists etc.	Process improvement plan and later revised process model after implementation of process improvement plan.	Action package containing action plan, infrastructure needed, techniques, tools, metrics etc. according to the business goals and assessment results for each process area.
Constraints	Can only be used for internal assessment before official assessment by CMM. If the CMM adoption is not intended then its application is limited.	Measuring the importance of software processes needs considerable experience.	Considerable experience is needed for process assessment, process prioritization and development of process guides.	Considerable experience is needed to identify current process model and process improvement plan.	Needs assistance of domain expert to develop action package according to the organizations' business goals and current capabilities.
Origin	Latin America	Ireland	Mexico	Britain	Spain

have resources to carry out improvement implementation [10]. By these reasons, this SPI approach is restricted to large organizations but Dyba [14] found that small organizations can and do implement SPI elements as effectively as large organizations, and in turn, achieve high organizational performance. Therefore, this indicates that SPI can be used as competitive advancement strategy for both small and large software organizations. But whether a small or medium scale organization can implement these methods without the help of some external quality consultant is yet to be proven.

5 Conclusion

In this paper we had studied software process improvement methodologies for SMEs and compared their significant characteristics. Each one has its benefits and limitations. Organization's should select the specific process improvement methodology keeping in view their business goals, models, characteristics and resource limitations. These methodologies can be adapted and tailored according to the organizational context. Russ and McGregor [45] proposed a software development process for small projects by integrating portions of an iterative, incremental process model with a quality assurance process and a measurement process used for process improvement. This process integrates many activities that might appear in separate processes in a larger project and its goal is to produce the high quality and timely results required for today's market without imposing a large overhead on a small project. Resources are scarce for small companies and most of them think they cannot afford the investment [33].

Furtherwork in this area is directed to perform case studies and empirical validation in real software development environment. It would be interesting to study the impact, efforts and comparison of these approaches on SPI in SMEs. Dyba [14] also suggested that future studies should focus on the specific needs of small software organizations in more depth; for example, through longitudinal, multiple case studies. Further research should be related to the study of new and improved measures of SPI success, comparison of measurement instruments, and validation of SPI success measures [14]. These further experiences will move towards tailoring software engineering methods and improvement strategies [14]. According to Russ and McGregor [45], if process monitoring and evaluation can be automated more, it will further free team members to focus on the project's goal of producing a quality software systems in SMEs.

References

1. Acuna, X., Ferre, M., Lopez, L.M.: *The Software Process: Modeling, Evaluation and Improvement*. World Scientific Publishing Company, Argentina (2000)
2. Ahonen, J.J., Forsell, M., Taskinen, S.-K.: A modest but practical software process modeling technique for software process improvement. *Software Process Improvement and Practice* 7 (2002)
3. Allen, P., Ramachandran, M., Abushama, H.: PRISMS: an Approach to Software Process Improvement for Small to Medium Enterprises. In: *Proceedings of the Third International Conference On Quality Software (QSIC 2003)*, November 6-7. IEEE, Dallas (2003)

4. Becker-Kornstaedt, U.: Towards Systematic Knowledge Elicitation for Descriptive Software Process Modeling. In: Bomarius, F., Komi-Sirviö, S. (eds.) PROFES 2001. LNCS, vol. 2188. Springer, Heidelberg (2001)
5. Becker-Kornstaedt, U., Hamann, D., Verlage, M.: Descriptive Modeling of Software Processes, IESE-Report 045.97/E, Fraunhofer Institute IESE, Germany (1997)
6. Biro, M., Messnarz, R., Davison, A.G.: The impact of national cultural factors on the effectiveness of process improvement methods: The third dimension. *Software Quality Professional* 4(4), 34–41 (2002)
7. Briand, L., Differding, C., Rombach, H.D.: Practical Guidelines for Measurement-Based Process Improvement. *Software Process: Improvement and Practice* 2, 253–280 (1996)
8. Broadman, J.D., Johnson, D.L.: What small business and small organizations say about the CMM. In: Proceedings of the 16th International Conference on Software Engineering, pp. 331–340. IEEE Computer Society, Los Alamitos (1994)
9. Bush, M.: CMM, The Capability Maturity Model. In: Guidelines for Improving the Software Process, Carnegie Mellon University, Software Engineering Institute. SEI Series in Software Engineering. Addison-Wesley, Reading (1995)
10. Calvo-Manzano, J.A., Agustin, G.C., Gilabert, T.S.F., Seco, A.D.A., Sanchez, L.Z., Cota, M.P.: Experiences in the Application of Software Process Improvement in SMES. *Software Quality Journal* 10, 261–273 (2002)
11. Christie, M., et al.: Software Process Automation: Interviews, Survey and Workshop results. Technical Report CMU/SEI-97-TR-008, Carnegie Mellon University/SEI (October 1997)
12. Curtis, B., Kellner, M.I., Over, J.: Process modeling. *Communications of the ACM* 35(9) (1992)
13. Damele, G., Bazzana, G., Maiocchi: Quantifying the benefits of software process improvement in Italtel Linea UT Exchange. In: Proc. ISS Conf., Berlin (April 1995)
14. Dyba, T.: Factors of Software Process Improvement Success in Small and Large Organizations: An Empirical Study in the Scandinavian Context. In: Proceedings of the 9th European software engineering conference (ESEC/FSE 2003), Helsinki, Finland, September 1–5, pp. 148–157 (2003)
15. ESSI, IBERIA, LAE. SPIE: Software Process Improvement and Experimentation, ESSI Project: No 10344 (February 1994)
16. Fortuna, R.M.: Beyond quality: Taking SPC upstream. *Quality Progress*, 23–28 (June 1988)
17. Glass, R.L.: *Software Creativity*. Prentice-Hall, Englewood Cliffs (1995)
18. von Wangenheim, C.G., Anacleto, A., Salviano, C.F.: Helping Small Companies Assess Software Processes. *IEEE Software* (January/February 2006)
19. Gresse, C., Punter, T., Anacleto, A.: Software measurement for small and medium enterprises – A Brazilian-German view on extending the GQM method (2003), <http://www.sj.univali.br/prof/Christiane%20Gresse%20Von%20Wangenheim/papers/ease2003.pdf>
20. Herbsleb, J., Carleton, A., Rozum, J., Siegel, J., Zubrow, D.: Benefits of CMM-based Software Process Improvement: Initial Results, Technical Report: CMU/SEI-94-TR-013, Pittsburgh (August 1994)
21. Herrera, E.M., Trejo Ramirez, R.A.: A Methodology for self-diagnosis for software quality assurance in small and medium-sized industries in Latin America. *The Electronic Journal on Information Systems in Developing Countries* 15(4), 1–13 (2003)
22. ISO9001, Quality systems- model for quality assurance in design, development, production, installation, and servicing, European Standard EN29001, Brussels, Belgium (1987)

23. Jalote, P.: *An Integrated Approach to Software Engineering*, 2nd edn. Narosa Publishing House (2000)
24. Johnson, D., Johnson, L., Brodman, J.G.: *Applying the CMM to Small Organizations and Small Projects*. In: *Proceedings of the 1998 Software Engineering Process Group Conference*, Chicago, IL (1998)
25. Johnson, G., Scholes, K., Sexty, R.W.: *Exploring Strategic Management*. Prentice Hall, Englewood Cliffs (1989)
26. Kautz, K.: *Software Process Improvement in Very Small Enterprises: Does it Pay Off?* *Software Process – Improvement and Practice* 4, 209–226 (1998)
27. Kautz, K.: *Making Sense of Measurements for Small Organizations*. *IEEE Software* 16(2), 14–20 (1999)
28. Kautz, K., Hansen, H.W., Thaysen, K.: *Applying and Adjusting a Software Process Improvement Model in Practice: The use of the IDEAL Model in a Small Software Enterprise*. In: *Proceedings of ICSE 2000, Limerick*. ACM Press, New York (2000)
29. Kurniawati, F., Jeffery, R.: *The Long-term effects of an EPG/ER in a small software organization*. In: *Proceedings of the Australian Software Engineering Conference*, Australia (2004)
30. Kuvaja, P., Palo, J., Bicego, A.: *TAPISTRY- A Software Process Improvement Approach Tailored for Small Enterprises*. *Software Quality Journal* 8, 149–156 (1999)
31. Kuvaja, P., Simila, L., Krzanik, L., Bicego, A., Koch, G., Sankonen, S.: *Software Process Assessment and Improvement: the BOOTSTRAP Approach*. Blackwell, Malden (1994)
32. Kellner, M.I., et al.: *Process Guides: Effective Guidance for Process Participants*. In: *Proceedings of the Fifth International Conference on the Software Process, USA* (1998)
33. Larsen, E.A., Kautz, K.: *Quality Assurance and software process improvement in Norway*. *Software Process – Improvement and Practice* 3, 71–86 (1997)
34. Madhavji, N.H., Holtje, D., Hong, W., Bruckhaus, T.: *Elicit: A Method for Eliciting Process Models*. In: *Proceedings of the Third International Conference on the Software Process, SA*, 1994 (2002)
35. Martin, S.: *Business Process Improvement*. McGraw-Hill, New York (2002)
36. McFeeley, B.: *IDEALSM: A users guide for software process improvement*, Handbook CMU/SEI-96-HB-001, Software Engineering Institute, Carnegie Mellon University (1996)
37. *Ministerio da Ciencia e Tecnologia, Quality and Productivity of the Brazilian Software Sector (in Portuguese)*, Ministerio da Ciencia e Tecnologia, Brazil (Government report – No Author) (2001)
38. Mintzberg, H.: *Structures in Fives: Designing Effective Organizations*. Prentice Hall International, Englewood Cliffs (1993)
39. Moe, N.B., Dingsoyr, T., Johansen, T.: *Process guides as Software Process Improvement in a small company*. In: *Proceedings of the EuroSPI Conference*, Germany (2002)
40. Nonaka, I.: *A dynamic theory of organizational knowledge creation*. *Organization Science* 5, 14–37 (1994)
41. Paulish, D.J.: *Case studies of software process improvement methods*, SEI Technical Reports, CMI SEI-93-TR-26 (1993)
42. Paulk, M.C., Curtis, B., Chrissis, M.B., Weber, C.V.: *Capability Maturity Model version 1.1*. *IEEE Software*, 18–27 (July 1993)
43. Richardson, I.: *SPI models: What characteristics are required for small software development companies?* *Software Quality Journal* 10, 101–114 (2002)
44. Richardson, I.: *Software Process Matrix: a Small Company SPI Model*. *Software Process: Improvement and Practice* 6, 157–165 (2001)

45. Russ, M.L., McGregor, J.D.: A Software Development Process for small projects. *IEEE Software*, 96–101 (September/October 2000)
46. Scott, L., Carvalho, Jeffery, R., Becker-Kornstaedt, U., Ambara, J.D.: Understanding the use of an electronic process guide. *Information and Software Technology* 44(10) (2002)
47. Scott, L., Jeffery, R., Becker-Kornstaedt, U.: Preliminary results of an industrial EPG evaluation. In: *Proceedings of Fourth ICSE Workshop on Software Engineering over the internet*, Canada (2001)
48. Scott, L., Zettel, J., Hamann, D.: Supporting Process Engineering in Practice: An Experience Based Scenario. In: *Proceedings of the Conference on Quality Engineering in Software Technology (CONQUEST)*, Germany (2000)
49. Software Engineering Institute, Improving processes in small settings: A research initiative of the SEI's IPRC, <http://www.sei.cmu.edu/iprc/iprc-overview.pdf>
50. Software Engineering Institute Capability Maturity Model@Integration (CMMISM) Version 1.1, <http://www.sei.cmu.edu/pub/documents/02.reports/pdf/02tr004.pdf>
51. SPIRE, Software Process Improvement in Regions of Europe, European Analysis Report v2.0, ESSI Project: No. 23873, Dissemination action (April 1999), <http://www.cse.dcu.ie/spire>
52. TickIT, A Guide to software quality management system construction and certification using EN29001, Issue 2.0, UK Department of Trade and Industry, London, UK (1992)
53. Wangenheim, C.G.V., Weber, S., Hauck, J.C.R., Trentin, G.: Experiences on establishing software processes in small companies. *Information and Software Technology* 48(2006), 890–900 (2006)
54. Wieggers, K.E., Sturzenberger, D.C.: A Modular Software Process Mini-Assessment Method. *IEEE Software* 171, 62–69 (2000)
55. Zahran, S.: *Software Process Improvement: Practical Guidelines for Business Success*. Addison-Wesley, Reading (1998)

An Empirical Study on Software Engineering Knowledge/Experience Packages

Pasquale Ardimento and Marta Cimitile

Dept. of Informatics, University of Bari, Via Orabona, 4,
I-70126 Bari, Italy
{ardimento, cimitile}@di.unibari.it

Abstract. This paper is concerned with characterization of software engineering knowledge and experience packages (EP) in the user perspective. It presents the first iteration of an evidence-based study. Results are presented from surveys conducted with many practitioners about the available experience bases, and on literature, to improve our understanding about the state of the practice and art for EP. Additionally, the paper presents attributes and their properties that, in the opinion of the participant practitioners, are relevant for characterizing an EP in the user perspective. Subsequently, with regard to this empirical system, the Acceptability indirect measurement model is provided for experience components. Moreover, the test of this measurement model is shown, which involved both developing qualitative evaluations with practitioners, and measuring ten Internet-available experience bases. Finally, the threats to validity are considered that, as usual for pilot studies, call for further investigation.

Keywords: Experience Package, Experience Base, Software Measurement, Empirical Software Engineering, Survey, and Experiment.

1 Introduction

Software Engineering (SE) knowledge and experience is a critical factor for software industry advancement [1,2,3]. The main SE processes - Development, Maintenance, Evolution, and nowadays Revolution [4] (i.e. software architecture radical reengineering) – still depend strictly on man-centered activities. Consequently, SE knowledge and experience, as gained in doing research and developing products, should be capitalized organization wide for reuse [5,6]. Assets to reuse should not be limited to mostly mature knowledge and experiences, more or less formalized, in case already taught; maturing and also pioneering experiences, lessons learned, and implicit or tacit knowledge gained during the development of application software or research projects, constitute a patrimony that should be [1,8,9] collected, analyzed, in case developed ad hoc in laboratory, synthesized, documented, wrapped in Experience Packages, EP(s), and distributed organization wide for reuse and tailoring to project specificities, rather than neglected or left in the individual ownership, hence subject to migration, and eventually forgotten and/or lost [10,11]. Finally, in our view, knowledge and experience should be allowed for unrestricted circulation [7] both

intra and inter the SE research world and the production world, those scientists and practitioners who share a value chain at least.

This paper is concerned with experience packaging: EP(s) and related structures, e.g. an Experience Base (EB), or a System of EB(s), that is a structure of EB(s) or Experience Base System (EBS). Because the paper shows that these objects share experience-related basic properties, in the remainder of this paper, let us denote such an object by **EO**, whatever it might be, an EP item, an EB or EBS structure.

The goal [12] is to characterize **EO**(s). The focus is on **EO** comprehension for selection, and application. It is assumed the perspective of the software project team, manager and technicians, in the context of an academic lab that works from many years in strict relationship with software development organizations.

In order to meet this goal, we conducted surveys on literature and with practitioner to identify the basic attributes and properties of the **EO** empirical relational system. Moreover, we utilized practitioners to measure these attributes both qualitatively, and qualitatively but in ordinal scale. Furthermore, we developed a software measurement model (SMM) on the given empirical relational system. Finally, we verified these SMM(s) vs. those qualitative/quantitative data by a pilot experiment.

The remainder of this paper is structured as follows: Section 2 recalls previous work. Section 3 shows the paper motivation and research questions. Section 4 presents surveys, lesson learned, and qualitative/quantitative evaluations. Based on these, Section 5 presents identification and characterization of the basic layers and components of the Acceptability measurement model. Section 6 synthesizes on such a model, which Section 7 verifies in lab by using ten experience bases. Section 8 completes the paper by providing some conclusive insights and final remarks, and showing prospective works.

2 Previous Work

Let us recall a few of the **EO** related works. An extended technical report [16] presents both the Empirical Relational System (ERS), and the Formal Relational System (FRS) of this study, their properties, operations, and relations; it also includes extended bibliography. An ERS can be, briefly defined as a model of the part of the “real world” we are interested in and FRS represents the mapping to numbers or symbols of the concepts of the real world, in particular, the formal objects relate to the empirical objects, formal relationships model the empirical relationships and formal operations are the mapping of empirical operations.

Software measurement models: Basic concepts are in classic books [13] and in some more recent surveys [14, 15]. Some aspects were investigated that concern formal definition and validation of attribute-based measurement models (see references in [15]).

Regarding the development of SMM(s), it is fundamental the work about the Goal-Question Metrics paradigm [12,17,18].

SE knowledge and experience representation for reuse and exchange: The Experience Factory (EF) [19] was introduced for collecting, analyzing, synthesizing, storing, and spreading organization wide knowledge and software experience of any kind, hence

making it available for project organizations. The EF concept was successively refined [36], surveyed [8,20], taught [21], specialized to different domains [5], implemented [22,23,24,25,26,27], and extended [28]. The EF is an architecture framework that is able to support both the Quality Improvement Paradigm [29], and the Goal-Question-Metrics method [18]. EF is a logical organization, which might have or not have a separate physical implementation [5, 8, 1]. From the organizational and technological points of view, an EF is founded around an experience base. From a process perspective, an EF consists of methods, techniques, and tools for working on reusable experience. To facilitate diffusion and reuse organization wide, knowledge is organized in EP(s), which populate the EB.

In order to reuse a package without having to consult the person or group, who initially gained and reported about an experience or formalized and stored knowledge, an EP is to intend as a chunk of knowledge/experience that is organized according to EF and EB rules and structures, including instructions for usage [30], and pros and cons of such a usage.

Experience Packages: Some pioneering organizations begun with storing into repositories, describing by digital catalogues, and publishing knowledge/experience packages. Nowadays, an amount of EP repositories are available through Internet. Moreover, a number of published papers describe and characterize the available EP(s) [21]. However, regarding EP(s) transfer and circulation, the literature recognize that we are still missing rigorous definition of the EP attributes necessary for having transferable and able for circulation EP(s) [24].

3 Paper Motivation and Research Questions

Current knowledge/experience packages are very often published through Internet; apparently, it should be easy to identify by various search engines, and eventually access and use them for free or at reasonable cost.

Nevertheless, we should not neglect that practitioners still seem to access and use those knowledge/experience packages rarely [31]. Whether this would hold true, we should deduce that there must still be some significantly wrong things in the way we use to package experiences. Regarding this point, a relevant question is hence: In what extent, is it actually accessed and utilized the set of PE(s) available through Internet? Answering this question in a useful way requires taking in major account the point of view of the EP end-users, i.e. business-software stakeholders.

This leads to place further relevant questions, like: In what extent is it well defined and implemented the set of EF and EO rules and structures that we use to accommodate knowledge/experiences in packages and bases? Is it easy for business stakeholders to look for recognizing useful EO(s), if any, among the available ones? Is it reasonable the effort required; can a given organization afford it? Moreover: Does it pay off using a recognized meritorious EO to start from the project on hand? What is the trade-off, and who should pay for basic costs, like searching for useful EO(s) and training personnel? In other words, some technical and economic barriers still hamper the development and subsequent usage of EO(s). Removing or, at least, moderating the impact of those barriers is a research task; hence, questions placed above are research questions.

4 Preliminary Empirical Work

This paper is based on the development of a number of technology innovation and transfer projects within an academic SE Research Laboratory (SERLab) [32]. These projects were mainly aimed to transferring research results from that lab to production environments [33].

At certain point in times, our doubts about the practical utility of some EP(s) we were considering, pushed us to ask practitioners for their opinion about the usages, if any, they were doing of **EO**(s) in their projects, and then, based on their depressing answers, to conduct survey both on literature, and **EO**(s) as they are available through Internet.

The goal of this preliminary stage of our research was hence to conduct a qualitative study aimed to get confirmation/disconfirmation that **EO**(s) are rarely utilized, in our region at least, and in case of disconfirmation, to start reasoning about the why of such a result, what the influential factors might be, and what is the perception about the levels of presence or absence of such factors in actual **EO**(s).

In order to meet this goal, we started with informally interviewing many practitioners. Eventually, based on the analysis of the language these practitioners and ourselves were using to describe pros and cons of using **EO**(s), we found some recurring terms used, like **EO**'s "acceptability", "comprehensibility", "identifiability", "applicability", "evaluability", "usability", "cost", and so on. Additionally, an initial kernel of lesson learned (LL) was established [34].

Then, in order to know whether SE had already identified these terms – and, in case, had already transposed them in direct/indirect SMM(s) – we launched a survey. This additional work gave us further insights, allowing us to extend the initial list of collected LL [34].

Researchers and practitioners, who participated to this survey, agreed in recognizing "acceptability" as the main term in the user view. Hence, we called *acceptability* (*Ea*) the correspondent attribute of the **EO** empirical system; it should be able to represent as a whole an experience component (i.e. package, base, or system of bases), thus it was defined as in the followings (let the terms EP, EB and EBS map the empirical attributes *EPa*, *EBa*, and *EBSa*, respectively):

- *acceptability* (*Ea*) concerns the extent in which an **EO** is adequate for usage in the software applications a practitioner is developing or expect to develop.

Participants also identified three main empirical factors as in the followings, which might influence the ability of software practitioners to cope with an **EO**; we should look for relationships of these factors with *Ea*, if any:

- *identifiability* (*Ei*), which concerns the user quickness in recognizing effectively the contents of an **EO**;
- *applicability* (*Ec*), which is the availability in an **EO** of the information that users need for assessing cost and benefit of using that **EO**.
- *evaluability* (*Ee*), i.e. making explicit technical constraints, requirements, and technical barriers, which relate to using an **EO**.

Additionally, the participants identified other potentially influential factors/sub-factors/parameters, which next sections 5 and 6 will take in consideration.

Our consequent decisions were: to collect an as large as possible set of the EB(s) as available through Internet; to select a number of them for pilot investigation; to

evaluate these EB(s) first qualitatively, and then quantitatively but using a scale as rough as an ordinal one, from the perspective of the EB user, in view of utilizing qualitative results to intersect [35], and results of all kinds to verify more precise and reliable quantitative results, as a controlled experiment should provide.

Subsequently, we searched Internet for the available EB(s) by using common search engines, so acquiring a number of EB(s). For this first iteration of our investigation, we selected 10 EB(s), and all of them were in the SE area. These constituted the EBS to investigate; rather than using pure random selection, we preferred to conduct, and use results from, literature survey on EP analysis [36, 23,24]. We were aware of the fact that the size of the chosen EBS (#10) could not be enough for deep inference analysis; however, we evaluated it adequate for a pilot experiment, and the effort we could be able to enact in this stage of the study.

Following this point, we worked with each of the selected EO(s) in the aim of evaluating them qualitatively and quantitatively from the consumer perspective. Concerning the former, the question was: As a whole, is it the given EO good enough for a consumer? Expected answer: Yes (Y), Not (N) or a documented Doubt (D). Concerning the most recurrent attributes, how do you score each of them? Expected result: Null or quite null (NN), Very Low (VL), Moderately Low (ML), Moderately High (MH), Very High (VH), or Top most (TT).

Table 1 shows generic local identifiers (leftmost column) for the EB(s) we utilized in the empirical study, and qualitative (rightmost column, QL) and quantitative results obtained. These results flow over any pessimistic conjecture about the state of the EP(s) practice in the user perspective.

In fact, all of the investigated EO(s) seem to be of no utility for practitioners. However, these results also seem to give some interesting insights: Qualitative results seem to show a similar movement than *Ea*. Additionally, a number of points could depend on a combination of *Ei*, *Ec*, and *Ee*.

Table 1. Qualitative evaluation of, (column QL), and quantitative evaluations in ordinal scale of some empirical attributes for, the experience bases utilized in this work

	<i>Ei</i>	<i>Ec</i>	<i>Ee</i>	<i>Ea</i>	QL
EB₁	MH	MH	MH	MH	D
EB₂	ML	NN	MH	VL	N
EB₃	TT	MH	VL	MH	D
EB₄	ML	MH	NN	ML	N
EB₅	MH	MH	NN	ML	N
EB₆	MH	NN	NN	ML	N
EB₇	MH	MH	NN	ML	N
EB₈	ML	NN	VL	VL	N
EB₉	ML	NN	NN	VL	N
EB₁₀	MH	MH	NN	MH	D

5 Attributes of the Acceptability Software Measurement Model

The goals of this section are to reason on knowledge/experience packages and bases, and show some basic properties of an SMM, which we call *Acceptability* for the

empirical attribute *acceptability*, *Ea*. An extended version of this paper [16] is made available on request to authors, which includes the detailed definition of the ERS, FRS, and scales for **EO**(s). Nevertheless, some basic empirical findings should be sketched, which concern the **EO** empirical system, including attributes aforementioned *Ee*, *Ei*, *Ec*, and *Ea*. In order to abstract on these attributes, let us use words that derive from “worth” (e.g. “worthiness”) to denote such an empirical attribute, whatever it might be. Given two or more **EO**(s), some operations are allowed, including the following ones. (i) Moving or copying information to an EP (Merge, Copy); in the destination EP, the quantity of empirical attributes is affected; this occurs in a way that a simple addition is not enough to model; what occurs is a more or less complex *averaging*. In other words, it makes to decrease the worthiness of a worth-for-use experience package moving to it unworthy or moderately worth information, and vice versa, for what concerns moving worth-for-use knowledge to a not meritorious package. (ii) Removing information from an EP package; similar to behaviors describes in point (i) but reversed. (iii) Putting EP(s) in the same set; it is an operation (Union) that does not change properties of those EP(s). Moreover: an EB is a structure on such a set; keys should be provided for facilitating the access to EB items; eventually, when the size of the EB is high enough, a catalog should be attached to the set, in order to help users in property-based searching for an EP.

Based on the ERS sketched above, an ERS-homomorphic FRS is constructed that, once augmented with a Ratio scale in a Real range, e.g. [0..1], leads to define SMM(s) for Acceptability and related sub-factors, as next sub-section shows.

5.1 Components of the Acceptability SMM

It gave us the chance of identifying the Acceptability’s main factors, sub-factors, and so on up to reach leaf parameters, the **EO** surveys that we had conducted.

Let us scale all these factors, sub-factors, and parameters in the Real sub-range [0..1]. Because each leaf parameter relates a chunk of information that serves a precise objective in the **EO** user view, let 0 be assigned when the information is missing or definitely bad, 1 when it completely meets the user needs, and an intermediate value in proportion to the quality of the registered information, as being between the worse and the best one. Of course, we expect that applying an indirect SMM to an **EO** will provide a measure that is reasonably dependent on the values that the **EO** components assume: e.g., an indirect SMM should return 0 (0.5 or 1, respectively) when its leaf parameters measure 0 (0.5 or 1, respectively).

Let us note that, when we were in this stage with our study, we also started to design the verification of the Acceptability SMM. Hence, in order to make manageable our first iteration of the SMM development, we made the further design decision to restrict the test cases for the leaf parameters to the central point, and the upper and lower bounds, of the Real scale we had chosen for them; in practice, we designed them to assume values 0.0, 0.5, and 1.0, in case just 0.0 and 1.0, so postponing other test settings to further iterations.

5.1.1 Basic Factors

As already mentioned, in our view, three basic factors impact on Acceptability:

- *Identifiability*, i.e. the user quickness in recognizing the contents of an EP;

- *Applicability*, i.e. making explicit technical constraints, requirements, and technical barriers, which relate to using an EP;
- *Evaluability*, i.e. the availability in an EP of the information that users need for assessing cost and benefit expected for using that EP.

5.1.2 Sub-factors

The survey that we conducted also provided us with knowledge for identifying the sets of information kinds that an EP is requested to include, in order to satisfy the preconditions that make applicable the Acceptability basic factors.

Because many stakeholders participated in deriving these kinds of the due information, some of those sets show a huge size. In other word, this lead the breakdown of each basic factor in multiple sub-factors, and these in sub sub-factors, eventually leaf parameters, i.e. factors that we are able to measure or want to leave to their stakeholders for subjective evaluation. The remainder of this section presents the decompositions that we made for the basic factors. See [16] for an extended presentation.

5.1.2.1 Description Parameters. We call Description Parameters (DP) the categories of the information that impact on Identifiability of an EO. In the followings, number 3 categories of description parameters are shown italicized, together with an associated information kind and some comments. *Domain (DP₁):* Application domain of the EO. *Problems afforded (DP₂):* Problems the EO could help to solve. The author/owner of EO should express these problems in the user view and terms, i.e. as the target stakeholders perceive and express them. *Keywords (DP₃):* They specify about the domain indicated for the given parameter. As far as a stakeholder proceeds in refining her or his selection, keywords can give help in understanding the set of problems that EO can solve, including problems that documentation does not explicitly mention elsewhere.

5.1.2.2 Experience Parameters. Let us call Experience Parameters (ExP) the categories of the information that impact on Applicability. In the followings, number 3 categories of identifiability parameters are presented. *History (ExP₁):* concerns (i) the When, Where, and Why the EO was originated and, in case, modified; (ii) successes and failures in applying EO in industrial settings, and (iii) problems of any kind encountered with adopting EO for previous applications. *Prerequisites (ExP₂):* in order to let an EO be applicable, they enumerate the conditions that business process has to meet, including activities to enact in advance, and semi finished materials, tools, techniques, methods, resources, and skills to make available. *Platform (ExP₃):* concerns the infrastructures allowed for applying EO. It is worth to include and highlight information about the flexibility of using EO with given infrastructures, and the level of scalability these can afford without affecting the effectiveness of the EO adoption.

5.1.2.3 Evaluability Parameters. Let us call Experience Values (EV) the categories of the information that impact on Evaluability. In the followings, number 11 categories of evaluability parameters are presented. *Economic Impact (EV₁):* it is the set of key performance indicators that might help users in assessing the EP significant business value. Those indicators are requested to summarize on the whole economic

impact of each of the following parameter fields. *Impact on Process (EV₂)*: The set of key performance indicators that should be used to assess the significant impact of the package for each business process. They must summarize the whole impact of the following fields. *Impact on Products (EV₃)*: Description of the consequences the new knowledge will eventually have. Again, in this case, a model should be attached for estimating the costs of application of the package (e.g. modification of any interaction of the working products with the innovative ones; recovery of data from the old products to be used in the new, any changes in platform hardware and software...) and the benefits. *Market Impact (EV₄)*: Description of what market changes the company setup will note after introduction of the knowledge package. A cost-benefit estimation model should be attached. *Impact on Value Chain (EV₅)*: Description of changes in the whole value chain, and especially the impact on the business processes related to the innovated process. One integrated or two separate cost-benefit models should be present. *Value for the Stakeholders (EV₆)*: Detailed analysis of all the expected values per type of stakeholder, who is concerned with applying the given EP. Values are expected to be expressed quantitatively as far as possible, therefore have an attached estimation model. *Adoption Risks (EV₇)*: Events that could occur during the EP adoption process and give a negative impact on time needed to acquire the knowledge that the EB includes, or on cost to incur and/or benefits to gain for using the EP. Actions that the EP puts in place to mitigate the risk should be explained in detail, including how and in what extent they work. All the risks listed should be associated with mitigation actions described in the EP Package Acquisition Plan (see next point). *Package Acquisition Plan (EV₈)*: Details about the actions to be taken in order to apply the EP. The plan is requested to point out how to govern the bearing of the innovation. The acquisition process is requested to report on trace from needs to resources. It is also requested to indicate acquisition time and costs; these should be predictable; hence the PE should be attached to appropriate estimation models. The EP is also requested to indicate what are activities that it ensures, benefits that it provides, and actions to take for maximizing benefits and mitigating the risks. *Skills Required (EV₉)*: The skills necessary to utilize the package should be described in detail. Models for package acquisition should also be described. It is also worth to describe in detail how the skills will evolve that the package has a bearing on. *Evidence (EV₁₀)*: The documentation files should be attached, where data and information are located that concern verification of the included empirical models. Owing to the cost of this validation, collections of experience applying the same package in previous projects, even in different contexts, can be amply used. *Package Owner Cost (EV₁₁)*: Return On Investment (ROI) calculation Model taking into account the expected costs and benefits derived from all the other variables regarding application of the knowledge package.

6 Synthesizing the Acceptability SMM

The goal of this section is to provide synthesized models, i.e. SMM(s), for the breakdown shown in Section 5, and put all together, layer after layer, up to reach the top layer, and so defining the Acceptability SMM. Let us recall that all the presented

SMM(s) take value in the Real range [0..1]. In the remaining SMM(s) are formally defined¹.

$\forall k \in [1 .. \text{EBS.size}()]$, let it be:

- *Description Parameter Presence*, DPP[i, k], i in [1 .. DP.size()]. DPP[i, k] takes value in {0, 1}, taking the latter if and only if (“iff”) the EB[k] includes one or more EP(s) having DP[i] = 1.
- *Experience Parameter Presence*, EPP[i, k], as for DPP; takes 1 iff the EB[k] includes one or more EP(s) having Exp(i) = 1.
- *Experience Value Presence*, EVP[i, k], s for DPP, but taking 1 iff the EB[k] includes one or more EP(s) having EV(i) = 1.
- *Description Parameter Rate*, DPR[k]: for EB[k], it takes value according to (1).
- *Experience Parameter Rate*, EPR[k]: for EB[k], it takes value according to (2).
- *Experience Value Rate*, EVR[k]: for EB[k], it takes value according to (3).
- Based on SMM(s) from (1) to (3): Let the acceptability EPA of a given EP be a function of the EP’s triplet (DPR, EPR, EVR). The same holds for the EBA of an EB, and the EBSA for an EBS. Hence, it is allowed using the simpler notation EA to denote any of them. The more this triplet approaches the value (1,1,1), the more acceptable is to consider the EO. In order to make simpler the usage of our SMM, let EA take value according to (4), again in the Real range [0..1].

Finally we can define three further SMM(s), one per basic-factors, which we call *Identifiability Influence* (II), *Applicability Influence* (AI), and *Evaluability Influence* (EI), respectively. They take Real value again in the Real range [0..1], according with formulas (5), (6), (7), respectively.

$$\text{DPR}_n = \sum_i \text{DPP}_{i,n} / \text{DP.size}(). \quad (1)$$

$$\text{EPR}_n = \sum_i \text{EPR}_{i,n} / \text{EP.size}(). \quad (2)$$

$$\text{EVR}_n = \sum_i \text{EVP}_{i,n} / \text{EV.size}(). \quad (3)$$

$$\text{EA}(\mathbf{E}) = (\text{DPR}(\mathbf{E}) + \text{EPR}(\mathbf{E}) + \text{EVR}(\mathbf{E})) / \text{BasicFactors.size}(). \quad (4)$$

¹ In the aim managing the complexity of the description in the limited room made available, the notation utilized is a classic object-oriented index-based notation. Hence, for a given EBS, EBS.size() return the number of the included EB(s), and similarly for other structures: e.g. BasicFactors.size() returns the number of basic factors (3 in the current version of the model), DP.size() returns the number of DP parameters (3 in the current version of the model), and the same holds for EP.size(). Again, EV.size() returns the number of EV parameters (11 in the current version of the model). An expression like $\forall k \in [1 .. \text{EBS.size}()]$ should read “For any EB in the current EBS, let the following points be applied”. EB[k] should read “The k-th EB of EBS”. Similarly for EB[k].Z.size(), and EB[k].Z[i], which should read “The size of the item Z in an EB”, and “the i-th element of the item Z in the k-th EB”, respectively.

$$II = \sum_t (DPR_{t=1..E.size()}) / E.size(). \tag{5}$$

$$AI = \sum_t (EPR_{t=1..E.size()}) / E.size(). \tag{6}$$

$$EI = \sum_t (EVR_{t=1..E.size()}) / E.size(). \tag{7}$$

7 Pilot Experiment

In order to verify the Acceptability SMM, we conducted a pilot empirical by using the EB(s) in Table 1, first column, as experiment objects. Two subjects – the same researchers who had developed the Acceptability SMM – utilized each of the EB(s) ES and assigned values to the ES leaf parameters. They eventually applied bottom-up the indirect SMM(s) in Section 6, so assigning value to leaf parameters and measuring for sub-factors, factors, and Acceptability.

The basic experiment hypotheses to confirm or disconfirm were that SMM outcomes significantly meet the correspondent quantitative and qualitative measures in Table 1, as agreed by a large collectivity of practitioners and researchers, based on experience. These practitioners, in some way, should be to consider experiment subjects too.

7.1 Data Presentation and Analysis

For each EB(s) in the given EBS, reports the values that factors DPR, EPR, EVR, and EBA assume. An extended version of this paper [16] reports figures for frequency distribution, box plots, and further graphics of data in

Table 2.

Let us observe that it usually shows very low values, the Acceptability (see rightmost column in Table 2) of the given EBS'EB(s) (first column).

The best EBA value (0.56) relates to EB₁ and is a few up the central point of the scale (0.50); the worst EBA value (0.11) relates to EB₉; this is an EB with which we were in great difficulty also when trying to visit it. Note that Real values in Table 2 completely map the qualitative/ordinal values in Table 1.

Based on the EBA data in Table 2, we can use expression (4) for an easy calculation of the Acceptability for the given set of EB(s), so obtaining again a quite low value (see expression (8)).

Based on the DPR, EPR, and EVR data in Table 2 (see the homonymous columns), we can easily obtain values for II, AI, and EI, respectively (9).

$$EBSA = 0.32. \tag{8}$$

$$II = 0.56; AI = 0.26; EI = 0.13. \tag{9}$$

Table 2. Acceptability of the experiment EBS

	DPR	EPR	EVR	EBA
EB ₁	0.66	0.66	0.36	0.56
EB ₂	0.33	0.00	0.27	0.20
EB ₃	1.00	0.33	0.18	0.50
EB ₄	0.33	0.33	0.09	0.25
EB ₅	0.66	0.33	0.09	0.36
EB ₆	0.66	0.00	0.09	0.25
EB ₇	0.66	0.33	0.00	0.33
EB ₈	0.33	0.00	0.18	0.17
EB ₉	0.33	0.00	0.00	0.11
EB ₁₀	0.66	0.66	0.00	0.44

Concerning the EB sample utilized in the experiment, let us consider the results of tests run to determining whether EBA, DPR, EPR, and EVR can be adequately modeled by normal distributions. As expected, the chi-square tests were not run because the number of observations was too small for each of those variables. However, since the smallest P-value among the test performed is not greater than 0.10 for EBA and EVR, we cannot reject the idea that EBA and EVR come from a normal distribution with 90% or higher confidence. Vice versa, since the smallest P-value among the test performed is greater than 0.10 for DPR and EPR, we cannot reject the idea that any of them comes from a normal distribution with 90% or higher confidence.

Concerning the results of fitting a multiple linear regression model to describe the relationship between EBA and the independent variables DPR, EPR, and EVR in Table 2, expression (10) shows the equation of the fitted model. Since the P-value in the ANOVA table is less than 0.05 (see [16]), there is a statistically significant relationship between EBA and the given three independent variables at the 95% confidence level. The R-Squared statistic indicate that the model as fitted explain 79.11% of the variability in EBA. In determining whether the model can be simplified, notice that the highest P-value on the independent variables is 0.74, belonging to EVR. Since the P-value is greater than 0.10, that variable is not statistically significant at the 90% or higher confidence level. Consequently, we should consider removing EVR from the model.

$$\text{EBA} = 0.12 + 0.18 * \text{DPR} + 0.37 * \text{EPR} + 0.07 * \text{EVR}. \quad (10)$$

7.2 Discussion

Computation (8) asserts that it is definitely low the level of Accessibility of the EBS, which we made by selecting through Internet from the EB(s) that SE literature mostly mentions. It seems to justify the definitely scarce usage of SE EB(s) in professional environments, the lack of supports and guidance that those EB(s) offer to professional users. At the present state of the art, because of the structure and organization of EB(s) as they are, it seems that the EB(s) designers do not account enough the point

of view of the practitioner software engineers, who, in their turn, demand for simplifying the access to, and suppressing barriers to consultation, comprehension, and acquisition of EB(s).

Expression (9) shows that values of Evaluability and Identifiability are quite small. However, it also shows that Applicability seems to influence more than Identifiability the EBSA (low) result, while Evaluability seems to be influential. Of course, due to the importance of the latter in the empirical domain, there must be something wrong in the used EBS and/or the proposed SMM.

The median of DPR is 0.67; hence, its distance from the top value (1.0) is 50% of its own value. Despite the dissatisfaction that this value generates, it also indicates that more than half of the analyzed EB(s) do wrap information which significantly facilitate the identification of the EP(s). Because the lowest DPR value does measure 0.33, then the whole EBS contains some EP-identification-oriented pieces of information. Hence, while we recognize that producers pay some attention to describing knowledge and experiences that their EP(s) encompasses, our study seems to tell that a major attention should be prescribed for information of this kind, and related formatting rules and standard representation schema should be provided.

The median of EPR is 0.33, which is really far away the top of the scale (1.0). This confirms that, concerning the EP(s)' technicalities, the analyzed EBS show a lack of information for applicability. The lowest EPS value is 0.0, which occurs 4 out of 10 times. This means that a consistent part of the EBS does not include technical descriptions of any kind, which could help users to evaluate the given packaged knowledge for use in their applications. In conclusion, EP(s) producers seem to place no care in providing information oriented to support EP(s) applicability.

The median EVR is 0.10, which is a value near to the very bottom of the scale (0). In fact, for the given experiment data set, the bottom-scale occurs three out of ten times. Data range from 0.00 up to 0.18; hence, the whole experiment EBS is very short of validated cost-benefits information. This means that EP users are not allowed to develop reliable benefit/cost analysis related to the acquisition/usage of such an EP.

In conclusion, it is reasonable that practitioners rarely utilize the knowledge, which the EBS we analyzed makes available. The Acceptability SMM we have been providing is quite as huge as the empirical domains it addresses; however, it makes explicit a practitioners' tacit believe that there are problems with quality of information that EP(s) encompass.

7.3 Validity Evaluations

The conducted empirical studies belong to the categories of Survey, and Pilot Study for SMM verification, respectively.

As already mentioned, we enacted surveys informally, i.e. without having a plan defined and documented in advance. However, because we had continual tight contact with the participant practitioners, we met them as frequently as we needed, and hence were able solve doubts, and eventually to keep threats in control, to the best of our understanding.

Concerning the validity of results: *External validity*; we are sufficiently confident that the selected experiment EP(s) are built as similar as professional and business ones. Hence, threats should not affect the external validity of this work. *Internal*

validity: due to the higher expertise of involved subjects, and the used treatments, threats should not affect the internal validity of this study. However, not using common forms for collecting EP knowledge from practitioners could be a threat. *Construct validity*: because members of the research group were also involved as experiment subjects, and they were interested to get successful results for their own PhD thesis, hypotheses guessing/fishing should be accounted as a threat. *Conclusion validity*: both the type and limited number of the experiment objects used should be accounted for threatening seriously the conclusion validity.

8 Conclusions and Future Work

In the present paper, we have been studying experience bases (EB) and their packages (EP) in the perspective of the professional user. We used ten EB(s), which are mostly mentioned in the scientific literature, as experiment sample. We analyzed these packages, and conducted surveys with experts and through literature, to the aim of understanding the extent in which they are able to support their actual/potential target users by giving these users the information they need to import those chunks of knowledge in their processes and products. To this aim: (i) with the help of many software practitioners, we detected basic attributes for, (ii) we developed, and (iii) verified, a basic Software Measurement Model (SMM), which we called Acceptability, and some other related sub-SMM(s).

Results show that Acceptability is a promising measurement model for EP(s); however, it should be further verified with more objects and subjects. However, those results confirm the believe of the involved practitioners that the acceptability of the experience bases as they are is mostly not yet sufficient for an extensive inclusion in their software processes, (ii) fortunately, there are many EB(s) that encompass information devoted to support the use on EP(S); however, this information usually shows very poor quality and organization.

The experience we gained is giving our prospective work a twofold goal: (i) enacting extended test of, and improving our Acceptability measurement model, according to the collected set of lessons learned, and (ii) identifying and experimenting with further ideas, which relates to the architecture of the information to organize in an EB in view of improving the EB's acceptability in the user perspective.

References

1. Davenport, T.H., Prusak, L.: Working Knowledge. Harvard Business School Press (1998)
2. Agresti, W.: Knowledge Management. Advances in Computers, 53 (2000)
3. Endres, A., Rombach, D.: A Handbook of Software and Systems Engineering. Addison-Wesley, Reading (2003)
4. Cantone, G., D'Angiò, A., Falessi, A., Lomartire, A., Pesce, G., Scarrone, S., Does a well structured code pay off? In: A Pilot Study, TR UoRM2 DISP submitted for acceptance to Profes 2008 International Conference (2008)
5. Basili, V.R., Caldiera, G., Cantone, G.: A Reference Architecture for the Component Factory. ACM Transactions on Software Engineering and Methodology (1994)

6. Decker, B., Ras, E., Rech, J., Klein, B., Reuschling, Ch., Höcht, Ch., Kilian, L., Traphoener, R., Haas, V., A Framework for Agile Reuse in Software Engineering using Wiki Technology. In: Knowledge Management for Distributed Agile Processes Workshop, Germany (2005)
7. Chesbrough, H.W.: Open Innovation: The New Imperative for Creating And Profiting from Technology. Harvard Business School Press (2005)
8. Basili, V.R., Caldiera, G., Rombach, H.D.: Experience Factory, Encyclopedia of Software Engineering, vol. 1. John Wiley & Sons, Chichester (1994)
9. Bomarius, F., Ruhe, G.: Learning Software Organization, Methodology and Applications. LNCS. Springer, Heidelberg (2000)
10. Thong, J.Y.L., Hong, W.Y., Tam, K.Y.: What leads to user acceptance of digital libraries? ACM, New York (2004)
11. Basili, V.R., Rombach, H.D.: The TAME Project: Towards improvement-oriented software environments. IEEE Transactions on Software Engineering (June 1988)
12. Basili, V.R., Weiss, D.M.: A methodology for collecting valid software engineering data. IEEE TSE SE-10(6) (1984)
13. Fenton, N.E.: Software Metrics: A Rigorous Approach. Intl. Thomson Computer Press (1991)
14. Cantone, G., Donzelli, P.: Software Measurements: from Concepts to Production, TR ISERN-96-15 (in Italian), For A slightly up-dated version of this paper see Cantone G., Donzelli, P. e Pesce, G.: Misure software: teoria, modelli e ciclo di vita. In: GUFPI – ISMA (Eds.) Metriche del Software. Esperienze e ricerche, Franco Angeli (2006), http://isern.iiese.de/network/ISERN/pub/technical_reports/iser-96-15.pdf
15. Morasca, S.: Software Measurements. In: Handbook of Software Engineering and Knowledge Engineering, ch. 26, vol. 1, World Scientific Publishing Co. Pte. Ltd, Singapore (2001)
16. Ardimento, P., Cimitile, M.: An Empirical Investigation on Software Engineering Knowledge/Experience Packages: Surveys on the State of the Art and Practice, Development of Measurement Models, and Pilot Experimental Verification, TR01, DI UoBari (January 2008)
17. van Solingen, R., Oivo, M., Hoisl, B., Rombach, D., Rue, G.: Adopting GQM-based measurement in an industrial environment. IEEE Software (January/February 1998)
18. Solingen, R., van Berghout, E.: The Goal/Question/Metrics Method: a practical guide for quality improvement of software quality. McGraw-Hill PC, New York (1999)
19. Basili, V.R.: Software Development: A Paradigm for the Future. In: Proceedings of COMPSAC (1989)
20. Koennecker, A., Jeffery, R., Low, G.: Lessons Learned from the Failure of an Experience Base Initiative Using Bottom-up Development Paradigm. In: Proceedings of the 24th Annual Software Engineering Workshop, USA (1999)
21. Ruhe, G.: Experience Factory-Based Professional Education and Training. IEEE Software (1999)
22. Basili, V.R., McGarry, F.E.: The Experience Factory: How to Build and Run One. In: Tutorial given at the 17th International Conference on Software Engineering. ACM Press, New York (1995) A slightly up-dated version of this paper was also presented at the 20th International Conference on Software Engineering (ICSE 2000), Kyoto, Japan, April (1998)

23. Basili, V.R., Lindvall, M., Costa, P.: Implementing the Experience Factory concepts as a set of Experience Bases. In: Proceedings of the 13th Conference on Software Engineering and Knowledge Engineering (2001)
24. Basili, V.R., Bomarius, F., Feldmann, R.L.: Get Your Experience Factory Ready for the Next Decade: Ten Years After Experience Factory: How to Build and Run One, ICSE Tutorial (2007)
25. Jedlitschka, A., Pfahl, D.: Experience-Based Model-Driven Improvement Management with Combined Data Sources from Industry and Academia. In: Proceedings of the 2003 International Symposium on Empirical Software Engineering (2003)
26. Broomé, M., Runeson, P.: Technical Requirements for the Implementation of an Experience Base. In: Proceedings of the Eleventh Conference on Software Engineering and Knowledge Engineering, Germany (1999)
27. Jedlitschka, A., Nick, M.: Software Engineering Knowledge Repositories. In: Conradi, R., Wang, A.I. (eds.) Empirical Methods and Studies in Software Engineering Experiences from ESERNET. Springer, Heidelberg (2003)
28. Feldmann, R.L., Rus, I.: When Knowledge and Experience Repositories grow new Challenges Arise. In: Proceedings of the 5th International Workshop on Learning Software Organizations (2003)
29. Basili, V.R.: The Maturing of the Quality Improvement Paradigm in the SEL and Experience Factory Fundamentals. In: 18th Software Engineering Workshop (SEL), NASA/Goddard Space Flight Center, Greenbelt, MD, 1-2 December (1993)
30. PERFECT Consortium, PIA Experience Factory- The PEF Model, PERFECT Consortium (1997)
31. Rus, I., Lindvall, M.: Knowledge Management in Software Engineering. IEEE Software (2002)
32. <http://www.serlab.di.uniba.it>
33. Boffoli, N., Caivano, D., La Nubile, F., Visaggio, G.: La Sperimentazione come Veicolo per l'Introduzione di Innovazione Tecnologica, AICA (2003)
34. Ardimento, P., Cimitile, M.: Lesson Learned while Studying Empirically Software Engineering Knowledge/Experience Base, TR01, DI UoBari (January 2008)
35. Seaman, C.: Qualitative Methods in Empirical Studies of Software Engineering. IEEE Transaction on Software Engineering 25 (1999)
36. Basili, V.R.: The Experience Factory and its relationship to other quality approaches. In: Advances in Computers, vol. 41, Academic Press, London (1995)
37. Wohlin, C., Runeson, P., Host, M., Ohlsson, M.C., Regnell, B., Wesslen, A.: Experimentation in Software Engineering: an Introduction. Kluwer Academic Publishers, Dordrecht (2000)
38. Jedlitschka, A., Pfahl, D.: Experience-Based Model Driven Improvement Management with Combined Data Sources from Industry and Academia. IEEE Software (2003)

Customized Predictive Models for Process Improvement Projects

Thomas Birkhölzer¹, Christoph Dickmann², Harald Klein³, Jürgen Vaupel²,
Stefan Ast³, and Ludger Meyer³

¹ University of Applied Sciences Konstanz, Braunegger Str. 55, 78462 Konstanz, Germany
thomas.birkhoelzer@htwg-konstanz.de

² Siemens Medical Solutions, Postfach 3260, 91050 Erlangen, Germany
{christoph.dickmann, juergen.vaupel}@siemens.com

³ Siemens CT SE 3, Otto-Hahn-Ring 6, 81730 München, Germany
{h.klein, stefan.ast, ludger.meyer}@siemens.com

Abstract. A methodology is presented to quantitatively model the expected relationships between investments in process improvements and improvements in business measures. Such a predictive model can be used as an auxiliary in process improvement planning in addition to established models like CMMI. Different from a generic model like CMMI, the proposed methodology allows for creating a fully customized model focusing on the context or product at hand. To manage the inherent parameter uncertainty of quantitative modelling of software processes a novel approach in this context is used by explicitly handling the parameter variations using interval arithmetic. The paper outlines the methodology and presents results from a study at Siemens.

Keywords: Process Improvement, Modelling, Quantification, Simulation.

1 Introduction

Each process improvement project or effort is guided by expectations of the relationships between investments in process improvements and the “resulting” improvements in business measures. For example, an improvement in the process area requirement engineering ought to yield better customer satisfaction and therefore better business results in the long term.

Ideally, these expected relationships determine the priority and order of improvement efforts (which process areas or practices should be considered first?), the chosen amount (how much is necessary / appropriate to achieve the desired return on investment?), and the measurement of success (did the improvement project meet the promises?).

However, process performances and therefore process improvements are just one factor influencing the respective business measures. For example, customer satisfaction depends not only on the quality of requirement engineering (or other internal processes) but on technology changes, competitor behaviour, market trends, and sometimes even unsubstantiated customer perception. Thus, the relationships between process improvements and business measures are highly perturbed by other superimposed effects and cannot easily be read off existing data. Therefore, only few of such

statistic relationships are reported in the literature based on long term observations or surveys with many participants [14]. Moreover, such generic findings may or may not apply to a concrete situation or organization.

This creates the well known dilemma for most process improvement efforts: The groundwork of attestable relationships between efforts and results is just not readily available. There are two common ways to cope with this:

Either, process improvement efforts are planned or proposed on implicit qualitative expectations of process experts, but without concrete or explicit quantification: A practice is proposed and argued as “better” in a concrete situation and in terms of certain issues, but usually without any quantification and consideration of side-effects. This approach is often successful, because it is based on the expert’s knowledge of the concrete environment; however, the implicit nature of such reasoning hampers priority planning, resource balancing, and control of success.

Or, process improvement efforts are planned based on existing generic process models like CMMI [13]. A CMMI assessment usually provides clear guidelines for improvement priorities and feedback on the success so far (improvement in capability or maturity level) based on a model that is acknowledged and widely adopted. However, the generic nature of the model does not consider any specific context, e.g. the priorities based on the capability or maturity levels may or may not be optimal for a concrete organization.

In this situation, a third alternative is described in this paper: Explicit quantitative modelling of the implicit knowledge of the experts of the organization. The basic goal is to cast the assumptions and experiences of the experts into a model that estimates the eventual business outcomes of improvement investments. Such a customized organization- or product-focused model can then be used as an auxiliary tool for priority planning, resource balancing, and control of success. Within a process improvement project, such a modelling can be accomplished with reasonable extra effort as will be outlined below. Such modeling also helps defining measures and improving the understanding of software development processes and their results.

Of course, such an estimation tool can not claim proven evidence. The results must be handled with the appropriate care and should augment, but not replace other reasoning. In order to support this, the inherent variation (due to uncertainty or different assessment by experts) of the parameters is preserved using an interval arithmetic calculation.

Modeling and simulation is an emerging methodology in the context of process improvement. An overview of the field can be found in [3] and [7]. The prevailing approach is to model software *project* performance [2], [4], [5], [6], [8], [9], [11], i.e. the simulation resembles the course of a single software project. To evaluate process alternatives, different simulation runs with different parameters or model structure are then compared.

In our work, the goal is a model on the level of the *organization as a whole*, i.e. simulation variables represent the process definition and performance in organizations that use process models to run development projects. Of course, from a bottom-up perspective, organization performance consists of the sum of project performances. Therefore, one might consider simulating organization performance based on single projects. However, such a bottom-up approach would create a very complex simulation which would be comprehensible only for specialized model experts. For this reason, a top-down modelling approach was chosen as described in section 2.1. A similar idea, however with a much more limited scope, can be found in [10].

In this paper, the use of interval arithmetic is introduced in this context to explicitly embody the existing, inevitable parameter uncertainty in the model and in the simulation outcomes.

In section 2 the generic structure of the model and the underlying calculus are outlined, section 3 discusses the necessary steps to actually instantiate a customized model, and section 4 presents some exemplary results of a pilot study. The paper is concluded by a brief discussion of possible uses of such a model in section 5.

2 Model

2.1 Structure

In order to compare the effects of different investment scenarios in the context of process improvement, the predictive model should relate investments in a (customizable) set of process areas to a (customizable) set of development or business metrics, see Fig. 1. The metrics as indicators of process performance can itself be used to decide about the process improvement investment in the next improvement iteration.

To achieve this, the model structure is to be designed as a compromise between expressiveness, i.e. ability to capture the important characteristics of the relations, and comprehensibility. The latter is of special importance to enable software process experts to use and customize the model, but requires the introduction of top-down (simplifying) abstractions.

Moreover, additional details of a model structure are justified only if data or knowledge is available to fill the associated parameters. Otherwise, parameter uncertainties superpose any effects gained by the added details.

Therefore, the model is based on the following abstractions:

- The performance of a process area can be described by a single number, e.g. the capability level. This concept is taken from established process models like CMMI or SPICE.
- A continuous investment is necessary to maintain a certain performance of a process area. This reflects the fact that process improvement is not a one time effort but needs continuous dedication.
- The performance of a process area reacts with a first-order time-dynamic with dead-time to a change in investment, see Fig. 2.

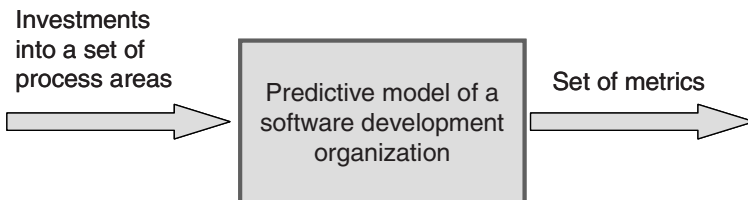


Fig. 1. External view of the intended predictive model: The model should relate investments to outcomes in order to enable analysis of different investment strategies

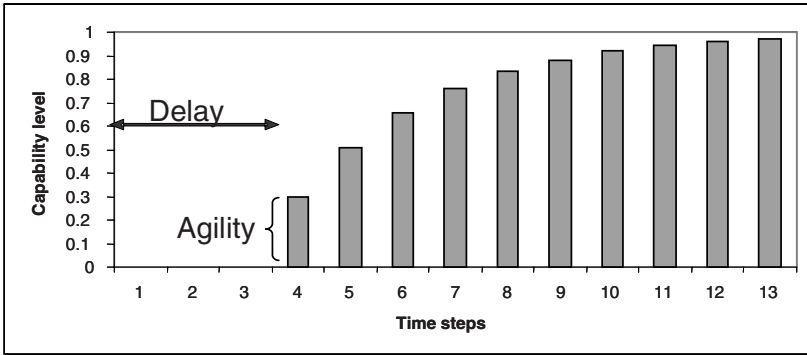


Fig. 2. Step response of a first-order time-dynamic with dead-time as used in equation (1) with the two parameters agility λ and delay τ

- Metrics depend on the performances of process areas only. Each metrics can potentially be influenced by any process area. Other influences are considered as constant (i.e. are omitted in the context of the model).
- All dependencies are either linear or gated by another model element. The latter is used to model the notion of prerequisites, i.e. the fact that some process areas require a certain performance of other process areas in order to be effective.

These abstractions were translated into mathematical formulas. The resulting mathematical model consists of sets of inputs $\hat{u}_t = (\hat{u}_{1,t}, \dots, \hat{u}_{n,t}) \in \mathbf{R}^n$ representing investments, internal state variables $\hat{x}_t = (\hat{x}_{1,t}, \dots, \hat{x}_{n,t}) \in \mathbf{R}^n$ representing normalized process area performances, and outputs $\hat{y}_t = (\hat{y}_{1,t}, \dots, \hat{y}_{m,t}) \in \mathbf{R}^m$ representing normalized business metrics. These model elements are related by time-discrete, nonlinear, first-order difference equations:

$$\hat{x}_{i,t+1} = (1 - \lambda_i) \cdot n(\hat{x}_{i,t}) + \lambda_i \cdot s_{i,t-\tau_i} \quad , \quad (1)$$

$$s_{i,t} = \alpha_i \cdot \hat{u}_{i,t} + e \cdot \left(\frac{\sum_j \beta_{ij}^{(x)} \cdot n(\hat{x}_{j,t}) + \sum_j \gamma_{ij}^{(x)} \cdot g(n(\hat{x}_{ij,t}), \nu_{ij}^{(x)}, \mu_{ij}^{(x)}) \cdot n(\hat{x}_{j,t})}{\left(\sum_j \beta_{ij}^{(x)} + \sum_j \gamma_{ij}^{(x)} \right)} \right) \quad , \quad (2)$$

$$\hat{y}_{i,t} = \frac{\sum_j \beta_{ij}^{(y)} \cdot n(\hat{x}_{j,t}) + \sum_j \gamma_{ij}^{(y)} \cdot g(n(\hat{x}_{ij,t}), \nu_{ij}^{(y)}, \mu_{ij}^{(y)}) \cdot n(\hat{x}_{j,t})}{\left(\sum_j \beta_{ij}^{(y)} + \sum_j \gamma_{ij}^{(y)} \right)} \quad , \quad (3)$$

$$n(arg) = \begin{cases} 1 & \text{for } arg > 1 \\ arg & \text{for } 0 \leq arg \leq 1 \\ 0 & \text{for } arg < 0 \end{cases} , \tag{4}$$

$$g(arg, \nu, \mu) = \frac{1}{1 + e^{-\nu(arg-\mu)}} . \tag{5}$$

Equation (1) embodies the time characteristic of the process area performances. It uses time difference equations (i.e. discrete time) instead to differential equations (continuous time) because the simulator should provide outputs only at discrete time points to reflect the fact that most real world metrics are also available a discrete time points only. Moreover, difference equations can be implemented without integration methodology (system dynamics). Equation (2) expresses the relationships within process area performances and equation (3) between process area performances and metrics containing linear and gated dependencies. Equations (4) and (5) are auxiliary functions for normalization and gating. The denominator terms in equation (2) and (3) normalize the values of all variables to the range zero to one. This complicates the formulas, but drastically eases parameter identification since no scaling issues need to be considered. The gating function as specified by equation (5) is used to model that a certain capability level of one process area is a prerequisite of the effects of another process area. Further details and motivation of this structure can be found in [12] and [15].

2.2 Interval Arithmetic

Equations (1) to (3) contain parameters $\lambda, \tau, \alpha, \beta, \gamma$, which are to be specified based on expert knowledge, see section 3. Since none of these parameters is amenable to a precise measurement or evaluation, this parameter identification can not provide or claim sharp values for each parameter. More realistically, each parameter is characterized by an interval, e.g. $\lambda_i \in [l_i, h_i]$, where l_i is the smallest and h_i the largest value, which the parameter might assume. This interval carries the notion of the parameter fuzziness due to varying expert appraisals, varying organizational environments, or inherent uncertainty.

Using interval arithmetic [1], this parameter fuzziness can be directly accounted for in the calculus of the model.

The basic idea of interval arithmetic is to provide worst case estimates for the results of mathematical operations involving variables bounded by intervals, e.g.

$$\begin{aligned} a_i \in [l_i, h_i] \wedge a_k \in [l_k, h_k] \\ \Rightarrow a_i + a_k \in [l_i + l_k, h_i + h_k] \\ \Rightarrow a_i \cdot a_k \in [\min(l_i \cdot l_k, l_i \cdot h_k, h_i \cdot l_k, h_i \cdot h_k), \max(l_i \cdot l_k, l_i \cdot h_k, h_i \cdot l_k, h_i \cdot h_k)] . \end{aligned} \tag{6}$$

The incorporation of this interval arithmetic into equations (1) to (5) is rather straightforward by replacing the standard mathematical operation by interval operation yielding to intervals as results for the process area performances \hat{x}_i and metrics \hat{y}_i . Special consideration, however, is required for the normalization in equation (2) and (3), because in these equations the same parameters appear at multiple places

(nominator and denominator). Simple interval arithmetic would ignore this dependency and create an overly pessimistic result, i.e. an interval larger than necessary. However, it can be shown, that the worst case values (minimum and maximum) are at the limits of the respective parameter intervals. Therefore, an embedded combinatorial search can provide the sought-after result interval.

3 Instantiating a Customized Model

In a study at Siemens, the methodology to instantiate a model based on the framework presented in section 2 was developed. It proceeds along the following steps:

1. Collection of the sets of process areas, which should be considered in the model. In a concrete environment this is usually rather straightforward process management knowledge or an organizational standard.

In the Siemens project, 19 process areas were used [15] that are part of a company-internal common process model which is based on CMMI. This static model comprises the process areas, the metrics, and qualitative relations between them. It was developed and agreed on within Siemens independent of the simulation model by a community process involving experts from all business units.

2. Collection of the set of metrics, which should be considered in the model. Depending on the maturity of the organization, the existing metric definitions might be refined or complemented during this step.

In the Siemens project, the 7 metrics of the company process model were chosen.

3. Identification of the model parameters using a questionnaire-based expert survey.

The questionnaire used in the Siemens project contained 126 questions, most of them with seven scaled answer alternatives to check. It took 40-90 minutes to complete. Specific care was taken to phrase the questions in the language of process experts, not using mathematical terminology or formulas. Instead, the questions focused on one model aspect at a time, e.g. the time variations (agility) of one process area or the strength of a particular relation. The 26 Siemens experts who completed the questionnaire had process knowledge from different Siemens business units with distributed software development and on average 6.6 years of process improvement expertise.

4. Compilation of the answers to determine the range for the respective parameter.

In the Siemens project, the first and third quartiles of the answers were used as boundaries for the respective intervals to capture the existing variation but to exclude outliers. The sizes of the intervals were between one and four on the scale of seven answer alternatives with an average of about two. For example the first and third quartile of the answers to the speed of change (parameter λ) for the process area "Project Management" yielded the interval $[3,6]$ and for the process area "Requirement Engineering" the interval $[3,4]$.

5. Determination of the mappings of the scaled answer alternatives to actual parameter values and of normalized model variables (process performances and metrics) to real world units.

The resulting model is filed as a spreadsheet (Microsoft Excel form) allowing easy access to any element of the specification without special knowledge or tools.

Cornerstone of the specification process described above is the identification of the model parameters by process experts (step 3). At a first glance, this might seem a rather difficult task and a considerable add-on effort for an improvement project. However, the questions collect systematically the following information:

- How much is an improvement in a process area about to cost?
- How much time does an improvement in a process area approximately need until first effects are visible (dead-time) and until it is fully effective?
- How much does a process area influence a certain metric?
- What are possible prerequisites for the effectiveness of a process area?

For almost all of these questions, there exist no readily available empirical evidence – therefore, the task seems indeed difficult. Nevertheless, each improvement project is driven by at least implicit assumptions about this information, otherwise no resource planning or prioritization would be possible. The questionnaire is therefore just a systematic approach to gather these implicit assumptions and document them explicitly. This might not be an additional burden but a value itself.

4 Results of the Study at Siemens

A special Java application was developed for interactive simulation, analysis and visualization of the models described above. Fig. 3 shows a screenshot of the program. The investment inputs are chosen by sliders on the left side. The diagrams on the right are the resulting metrics. Diagrams of the process area performances can be

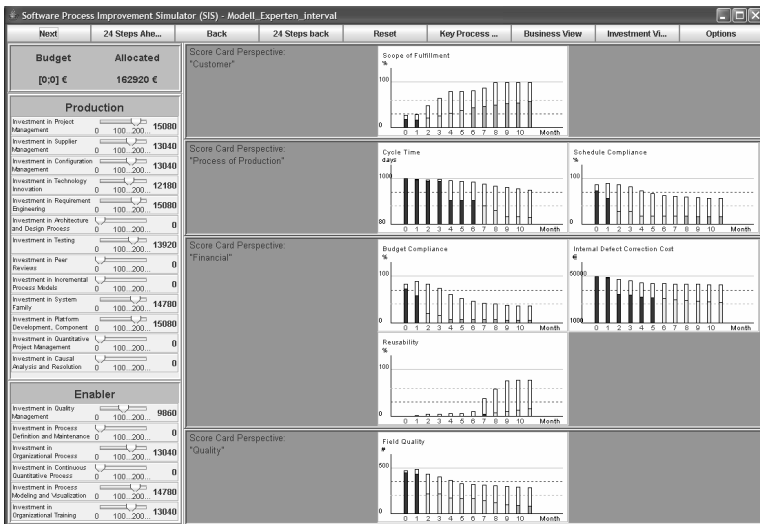


Fig. 3. Screenshot of the simulation application. The inputs are entered on the left side; the metrics are displayed in the center.

accessed in a separate window. The application permits models to be fully flexible within the framework of section 2 and implements the respective interval arithmetic calculations.

The model developed within the Siemens study was extensively analyzed with this application, for example checking the step response for all 19 investment inputs. Although more than 120 parameters were gathered just by expert estimates and not tuned beyond the procedure outlined in section 3, the model predicted expected or understandable results in almost all cases.

On the other hand, some results of the modeling revealed some omissions and impreciseness in the preexisting assumptions of the experts, e.g. missing aspects in the chosen set of process areas. This led to a refinement of the set of process areas.

It is beyond the scope of this paper to present or discuss company specific details of the model results. The following should rather give an impression of the type of results that can be obtained.

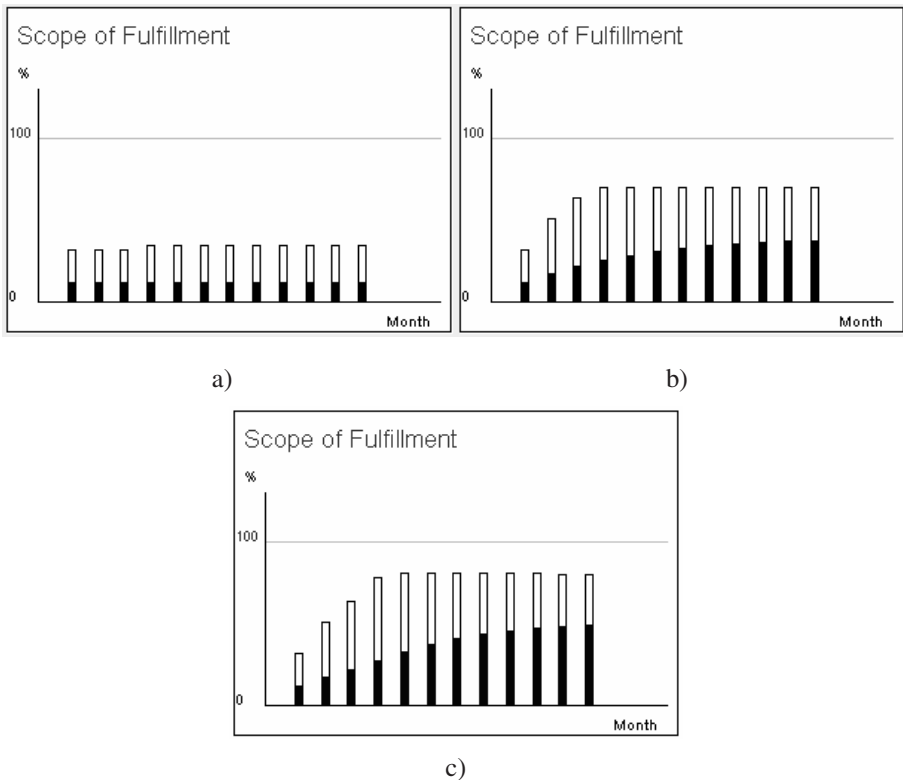


Fig. 4. Step responses of the metrics “Scope of Fulfillment”: a) investment in “Technology and Innovation” only; b) investment in “Requirement Engineering” only; c) investment in “Requirement Engineering” and “Technology and Innovation” at the same time. White parts of the bars indicate the range of variability.

As first example, the metrics “Scope of Fulfillment” is chosen. According to the experts, this metric is influenced (in the context of this project) by the process areas “Requirement Engineering”, “Technology and Innovation”, and “System Family”. The influence of “Technology and Innovation”, however, is gated by the process area “Requirement Engineering”. This models the experience that an advanced “Requirement Engineering” is necessary to actually translate innovation into customer value (“Scope of Fulfillment”).

Fig. 4 shows the step responses for an equal add-on investment of 10,000 Euro per month in “Technical Innovation” (Fig. 4a) alone, “Requirement Engineering” alone (Fig. 4b), and in both at the same time (Fig. 4c). Each bar represents a time step; the blank parts of the bars represent the intervals of uncertainty.

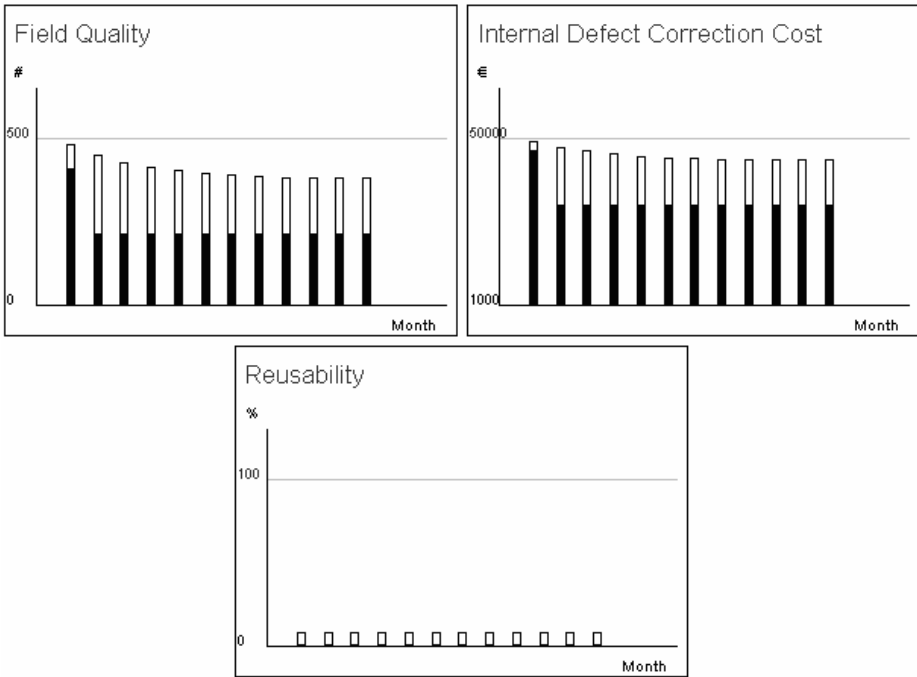


Fig. 5. Step responses of selected metrics for an investment in “Testing” only. White parts of the bars indicate the range of variability.

In this case, an investment in “Technology and Innovation” alone is predicted to be not effective, whereas an investment in “Requirement Engineering” alone shows considerable improvement effects on “Scope of Fulfillment”. A combined investment in “Requirement Engineering” and “Technology and Innovation” yields the best improvement even with a little less uncertainty. Thus, the model would suggest an improvement sequence starting with “Requirement Engineering” but not with “Technology and Innovation”. Note, that the intervals of uncertainty in Fig. 4b und 4c also reflect the variations in the assessment of time dynamics.

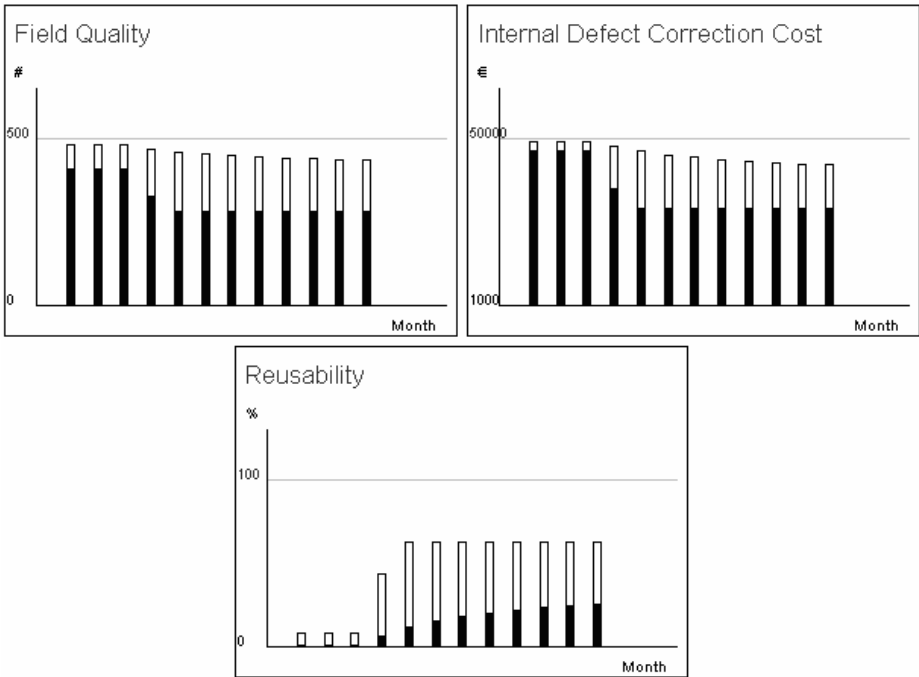


Fig. 6. Step responses of selected metrics for an investment in “Architecture and Design Process” only. White parts of the bars indicate the range of variability.

As second example, a trade-off between improvements in the process areas “Testing” and “Architecture and Design Process” should be explored. According to the experts, both process areas influence the metrics displayed in Fig. 5 and Fig. 6. The former one shows the simulation results of an add-on investment of 10,000 Euro per month in “Testing” only. In Fig. 6 the same amount is invested in “Architecture and Design Process”. Note, that the metrics “Field Quality” measured in number of defects and “Internal Defect Correction Cost” have an inverse direction, i.e. their values are reduced by an improvement action. In the simulator, this is achieved by mapping the internal normalized variable to an inverse scale.

In a first analysis step, only the two metrics “Field Quality” and “Internal Defect Correction Cost” (upper row of Fig. 5 and Fig. 6 respectively) should be considered. Both strategies yield somewhat different results for these metrics, but these differences are within the bandwidth of variations (white bars) anyway. This means considering field quality and defect costs only, there are different opinions about the trade-off between “Testing” (direct but end-of-pipe measures) and “Architecture and Design Process” (indirect but upfront measures) among the experts, while there is an agreement about the significance of both. It is unlikely, that this uncertainty can be resolved by a more precise model or parameter estimation.

In a second analysis step, however, the simulation directly and visibly indicates that the metrics “Reusability” is improved by an investment in “Architecture and

Design Process” only (and not by an investment in “Testing” only) providing a distinct advantage in a trade-off consideration agreed by all experts.

In a prepared and dissected example, such discriminative relationships might seem obvious. However, the incorporation of the multitude of such interrelations in a simulation model assures that none of them gets lost during the discussions and planning of a process improvement effort.

5 Discussion

Within the context of a process improvement effort or project, the development and use of a predictive model as described can provide several distinct advantages:

- The collection of model elements (process areas and metrics to be considered) and the investigation of their relationships is a systematic and thorough means to reveal and document the pre-existing assumptions and expectations of the involved process experts, i.e. valuable organizational knowledge. The quantitative nature of the model enforces at least tentative commitments with respect to these assumptions and expectations which might otherwise be sidestepped.
- In the best case, the estimations of the process experts widely agree and a model can easily be deduced. Dissimilar estimates, however, would yield too large intervals for the respective parameters, which hamper model expressiveness and use. Nevertheless, the uncovering and clarification of such differing expectations – in general or with respect to specific issues – might benefit the improvement project completely independent of model usage. Therefore, parameter variability can be used as an indicator to check and reduce unnecessary variants in process improvement perception and comprehension.
- Each single relationship just expresses the prevalent knowledge of experts. There is no “magic knowledge generation” in the methodology. However, the simulation model enables to evaluate the interrelations of all these single pieces supported by interaction, graphic, and animation. This might generate new insights into the complex cohesion and interplay of these elements.
- The simulation approach and tool allow different frameworks to be used. While CMMI was the underlying framework in the Siemens project, it can be used to model any relationships between efforts, performances of practices, and outcomes. Therefore, the methodology can be applied to other process models or organizational setups or as well.
- The quantitative model allows various modes of analysis, e.g. testing of alternatives to find an optimal improvement strategy or sequence given a specific goal. Automated search algorithms can find solutions, which might otherwise be missed. Of course, due to the abstract nature of the model, such “solutions” may not be considered as authoritative or dependable, but should just serve as proposals for further discussion and evaluation.
- The quantitative modelling and simulation indirectly supports the organization to achieve later, *quantitative levels* of CMMI, i.e. level 4 and 5. It prepares structuring and performing process improvement projects and measurements such that the model inputs and outputs are used to plan, control and present results.

6 Summary

In this paper, a new methodology was presented to develop a customized predictive model for process improvement efforts or projects. As an important part, interval arithmetic was introduced in this context to explicitly account for the inevitable parameter uncertainties. The mathematical framework as well as the necessary steps for model development were outlined and results from a project at Siemens were presented.

Unlike generic models like CMMI, all model elements, especially process areas and metrics (outcomes), can be freely chosen which enables an easy adaptation to organization-wide as well as to product- or department-specific contexts.

By explicitly documenting implicit assumptions and expectations and by providing a quantifiable base for decisions and comparisons of alternatives, such a model can serve as a valuable auxiliary for process improvement planning and implementation.

References

1. Moore, R.E.: Interval analysis. Prentice-Hall, Englewood Cliffs (1966)
2. Lin, C., Abdel-Hamid, T., Sherif, J.: Software-Engineering Process Simulation Model (SEPS). *Journal of Systems and Software* 38, 263–277 (1997)
3. Kellner, M.L., Madachy, R.J., Raffo, D.M.: Software Process Modeling and Simulation: Why? What? How? *Journal of Systems and Software* 46, 91–105 (1999)
4. Christie, A.M.: Simulation in Support of CMM-based Process Improvement. *Journal of Systems and Software* 46, 107–112 (1999)
5. Raffo, D.M., Vandeville, J.V., Martin, R.H.: Software Process Simulation to Achieve Higher CMM Levels. *Journal of Systems and Software* 46, 163–172 (1999)
6. Williford, J., Chang, A.: Modeling the FedEx IT Division: A System Dynamics Approach to Strategic IT Planning. *Journal of Systems and Software* 46, 203–211 (1999)
7. Raffo, D.M., Kellner, M.I.: Modeling Software Processes Quantitatively and Evaluating the Performance of Process Alternatives. In: Emam, K.E., Madhavji, N. (eds.) *Elements of Software Process Assessment and Improvement.*, pp. 297–341. IEEE Computer Society Press, Los Alamitos (1999)
8. Iazeolla, G., Donzelli, P.: A Hybrid Software Process Simulation Model. *The Journal of Software Process Improvement and Practice*, 97–109 (2001)
9. Martin, R., Raffo, D.M.: Application of a Hybrid Simulation Model to a Software Development Project. *Journal of Systems and Software* 59, 237–246 (2001)
10. Pfahl, D., Stupperich, M., Krivobokova, T.: PL-SIM: A Generic Simulation Model for Studying Strategic SPI in the Automotive Industry. In: *Proceedings of the 5th International Workshop on Software Process Simulation and Modeling (ProSim 2004)*, Edinburgh, pp. 149–158 (2004)
11. Raffo, D.M., Nayak, U., Setamanit, S., Sullivan, P., Wakeland, W.: Using Software Process Simulation to Assess the Impact of IV&V Activities. In: *Proceedings of the 5th International Workshop on Software Process Simulation and Modeling (ProSim 2004)*, Edinburgh, pp. 197–205 (2004)
12. Birkhölzer, T., Dickmann, C., Vaupel, J., Dantas, L.: An Interactive Software Management Simulator based on the CMMI Framework. *Software Process Improvement and Practice* 10(3), 327–340 (2005)

13. CMMI Product Team: CMMI for Development, Version 1.2. CMMI-DEV, V1.2, CMU/SEI-2006-TR-008, Pittsburgh (2006)
14. Galin, D., Avrahami, M.: Are CMM Program Investments Beneficial? Analyzing Past Studies. *IEEE Software* 23(6), 81–87 (2006)
15. Dickmann, C., Klein, H., Birkhölzer, T., Fietz, W., Vaupel, J., Meyer, L.: Deriving a Valid Process Simulation from Real World Experiences. In: Wang, Q., Pfahl, D., Raffo, D.M. (eds.) *ICSP 2007. LNCS*, vol. 4470, pp. 272–282. Springer, Heidelberg (2007)

Improving Customer Support Processes: A Case Study

Marko Jääntti¹ and Niko Pylkkänen²

¹ University of Kuopio, Department of Computer Science,
P.O. Box 1627, 70211, Kuopio, Finland
mjääntti@cs.uku.fi

² TietoEnator Forest&Energy Oy, Microkatu 1,
P.O. Box 1199, 70211, Kuopio, Finland
niko.pylkkanen@tietoerator.com

Abstract. IT organizations need systematic methods to manage the increasing number of service requests and software problems reported by customers. A large number of open problems can rapidly increase the costs of software maintenance and development. Therefore, an IT organization needs a well-defined customer support model. However, existing customer support models have one major shortcoming: a lack of process description that shows the interaction between different support processes (incident management, problem management and change management) and their activities. In this paper, we use a constructive research method to build an improved process model for customer support. Additionally, we present findings of a case study that focused on improving support processes in a medium-sized Finnish IT company. The research question in this paper is: What is the role of the problem management process in customer support?

1 Introduction

Customer support processes are often the most visible part of the IT organization to customers. By improving the quality of the support processes, IT organizations can easily increase the customer satisfaction on services and products. Additionally, the world-wide interest in IT service management processes, such as IT Infrastructure Library ITIL [1] and COBIT [2] provides evidence that the research area is important. In this paper, we focus on two service support processes: incident management and problem management. The primary focus is on problem management.

IEEE Standard Classification for Software Anomalies (IEEE 1994) [3] states that anomalies (problems and defects) may be found during the review, test, analysis, compilation, or use of software products or applicable documentation. A Framework for Counting Problems and Defects by the Software Engineering Institute (SEI) [4] emphasizes the same activities. It identifies five major activities to find problems and defects: software product synthesis, inspections, formal reviews, testing and customer service.

In the SEI model, the *software product synthesis* is defined as "the activity of planning creating and documenting the requirements, design, code, user publications, and other software artifacts that constitute a software product" [4]. *Formal reviews* include for example, defect causal analysis (DCA) meetings, service reviews, and problem reviews. A defect causal analysis method is based on the data received from a software problem report. The DCA approach [5] has three major principles: 1) Reduce defects to improve quality: software quality can be improved if the organization focuses on preventing and detecting defects in early phase of software development. 2) Apply local expertise: people who really know the cause of the failure and how to prevent problems in the future should participate in causal analysis meetings. 3) Focus on systematic errors: DCA people should select a sample of systematic problems from a problem database to be reviewed.

Service review meetings belong to the activities of the service level management process in ITIL [6]. The purpose of service review meetings is to review how service level requirements were met in the last service period, to identify weak areas in the service, for example the root cause of service breaks, and to define required improvement actions. *Problem reviews* [1] are quite similar than defect causal analysis meetings [7]. The purpose of problem reviews is to review the problems that have a high business impact. Problem reviews are focused to determine what was done right and wrong in problem resolution, what could be done better next time and how to prevent the problem from occurring again in the future.

Software testing is a process of executing a program on a set of test cases and comparing the actual results with expected results. The testing process requires the use of a test model that is a description about what should be tested and how testing should be executed [8]. Previous studies have emphasized the need of shifting testing to early phases of software development process such as requirements and specification phase and design [9].

Customer service including support processes is a broad concept. In fact, it can involve the other above mentioned problem finding activities. There are various theoretical frameworks and models regarding service support and managing problems and defects available for IT organizations: First, maturity models are designed for measuring the maturity level of software development processes or service management processes [10],[2]. Perhaps the most well-known maturity model in software engineering is the capability maturity model CMM [11]. There is also a specific CMM model for IT Service management CMM [12]. Second, quality standards (ISO 20000 service management standard [13], ISO/IEC 12207 [14]) include auditable requirements for processes. Third, there are IT service management process frameworks that define how to perform support processes: ITIL [1], COBIT [2], and Microsoft Operations Framework [15]. Fourth, software development lifecycle models (Rational Unified Process [16], for instance) also include information on quality assurance methods such as testing, risk management and defect prevention. Finally, academic literature provides a wide selection of other quality assurance models, such as Defect Management Process [17], the

framework for counting problems and defects [4], Personal Software Process [18] and software maintenance models [19], [20].

1.1 Our Contribution

All above mentioned service management models and maintenance models seem to have one major shortcoming. They do not include a process model that would show the interaction between different support processes as one diagram. The improved model should include incident management activities (identify and record, classify, investigate, and resolve incident), reactive problem management with problem control activities (identify and record, classify, investigate, and resolve problem, error control and proactive problem management), change management and application development. We call this model "a big picture of support processes".

The main contribution of this paper is 1) to provide an improved support process model, 2) to examine the role of problem management in customer support, and 3) to present experiences of a case study where we used that model to improve support processes in a medium-sized Finnish IT company. The results of this study are useful for support process managers and software quality managers and can be used to improve the quality of the service support processes.

The rest of the paper is organized as follows. In Section 2, the research methods of this study are described. In Section 3, an improved support process model and main findings from the case study are presented. Section 4 is the analysis of findings. The discussion and the conclusions are given in Section 5.

2 Research Methods

This case study is a part of the results of MaISSI (Managing IT Services and Service Implementation) and SOSE (Service Oriented Software Engineering) research projects at the University of Kuopio, Finland. The research question in this paper is: What is the role of the problem management process in customer support? Both constructive and case study research methods were used in this study. First, we build a theory-based model for customer support using a constructive research method. A theory consists of four key elements: a boundary that describes the domain of interest, key constructs within the domain, the values that constructs may take, and the relationships between the constructs [21]. Our model is based on the ITIL process framework (service support section) because ITIL is the most widely used service management framework in the world [1].

A case study can be defined as "an empirical inquiry that investigates a contemporary phenomenon within its real-life context, especially when the boundaries between phenomenon and context are not clearly evident" [22]. Case studies can be categorized into exploratory, explanatory and descriptive case studies. Our study is more exploratory than explanatory or descriptive because we did not have any predefined hypotheses in the study and did not focus on finding cause-effect relationships.

2.1 The Case Organization and Data Collection Methods

Our case organization Alfa supplies solutions for the navigation industry. Alfa is focused on mobile phones and portable devices. Data collection methods included informal discussions in research meetings with Alfa. Persons who participated in research meetings were a researcher and 2 research assistants (University of Kuopio), a quality manager (Alfa), and a support engineer (Alfa). The case organization was selected because they were very interested in improving customer support processes based on the IT Infrastructure Library framework. They were also planning to launch a knowledge base. The study was carried out according to the following schedule:

- September 14, 2006: Alfa and the research team defined goals for the pilot project (improving support processes: incident management and problem management).
- September 19, 2006: Alfa introduced the support process to the research team.
- October - December, 2006: The research team analyzed the support process of Alfa by using ITIL process framework as a benchmarking target.
- January 11, 2007: The results of the analysis (identified challenges) were presented in a review meeting (Alfa and the research team).
- January - March 2007: The research team (a research assistant) evaluates knowledge base / FAQ manager applications and their properties.
- March 1, 2007: The results of the tool evaluation were presented to Alfa. As a result of the evaluation, the best tool was selected and installed.
- March, 2007: The FAQ manager tool was configured, and user manuals and a new process description created.
- March 27, 2007: The FAQ manager tool, the process description were reviewed with a sample case (a customer sends a problem ticket, the first line support creates a draft FAQ item, the second line support approves and publishes the FAQ item (Alfa and the research team)).

2.2 Data Analysis Method

A within-case analysis method [23] was used in this case study. The data analysis focused on identifying strengths and challenges in Alfa's support processes. The most important results of the case study were tabulated according to the following questions:

- Which business goals have been set for the process improvement?
- Which tools are used for managing incidents and problems?
- How do they call the workflow asset (for example, helpdesk case)?
- What is the current state of the support process description?
- Which service support levels are visible in the process description?
- Which metrics are used within the support process?
- Which proactive methods are used within problem management?
- Which challenges are related to the customer support?

Additionally, we studied whether the customer support model of Alfa contains similar elements (activities and phases) than our improved customer support model.

3 The Role of Problem Management in Customer Support

In this section, the improved customer support process model is described. The support model was created by analyzing the service support processes of the IT service management framework (ITIL) and integrating the activities of the support processes together. Our model has the following advantages compared to the previous models:

- It emphasizes the role of customers (main users, normal users) in the support process. In fact, we consider a customer main user as a very first support level (we call it the support level '0').
- It integrates the activities performed by service desk teams, product support teams and product development teams together. The ITIL framework does not provide a detailed process diagram that would show the relations between different support process activities.
- It highlights the role of proactive problem management within customer support. IT organizations tend to document reactive problem management activities but usually forget to document proactive ones.
- It creates a bridge between problem management and error management (defect management). Previously presented defect management processes, such as [17], [24] focus solely on software defects although software users encounter also problems that never end to defects (for example, hardware problems, documentation errors, or problems with service availability and service performance)
- It uses knowledge management concepts [25] in proactive problem management. A knowledge base application helps organizations to create, store, share, and use the knowledge related to problems and their resolutions.
- It adopts the best features of ITIL-based support processes, such as users are never allowed to contact software developers directly. All service requests and problem reports go through a single point of contact, a service desk.

3.1 The Improved Customer Support Model

Figure 1 describes our improved support process model. The customer support process usually begins when a customer encounters a problem while using a service or a product and takes contact to the service desk (by phone, by email or by a web form). This contact, that is related to customer's or user's problem is called an incident. An incident can be defined as "any event which is not part of the standard operation of a service and which causes, or may cause, an interruption to, or a reduction in, the quality of that service" [1]. It is important to distinguish service requests from incidents. Service requests can be defined as

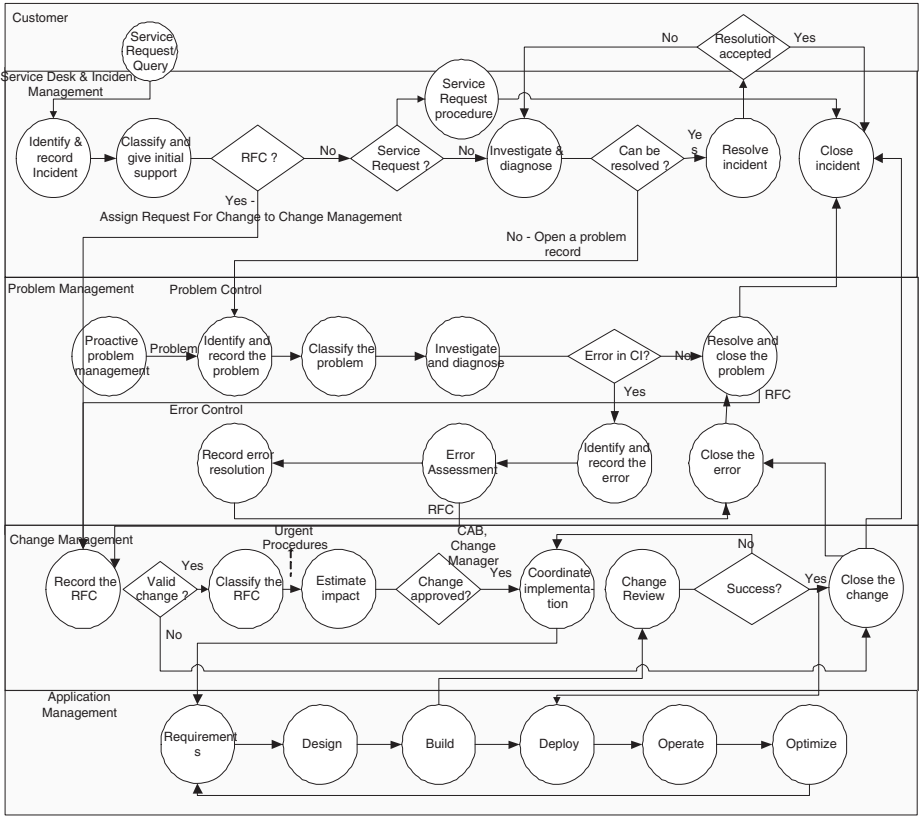


Fig. 1. Improved customer support model

"a request from user for support, delivery, information, advice or documentation, not being a failure in the IT infrastructure" [1]. A good example of a service request is a user's request for more disk space.

While the goal of incident management (the process performed by the service desk function) is to restore normal service operation as quickly as possible, problem management aims to minimize the impact of problems on the business, to identify the root cause of problems, and resolve errors related to configuration items (CIs). The key difference between problem management and incident management is that in problem management, the quality of the resolution plays a more important role than the resolution time.

3.2 The Relation between Problem Management and Customer Support

Problem management has both reactive and proactive aspects. Reactive problem management aims to resolve incidents and problems reported by customers.

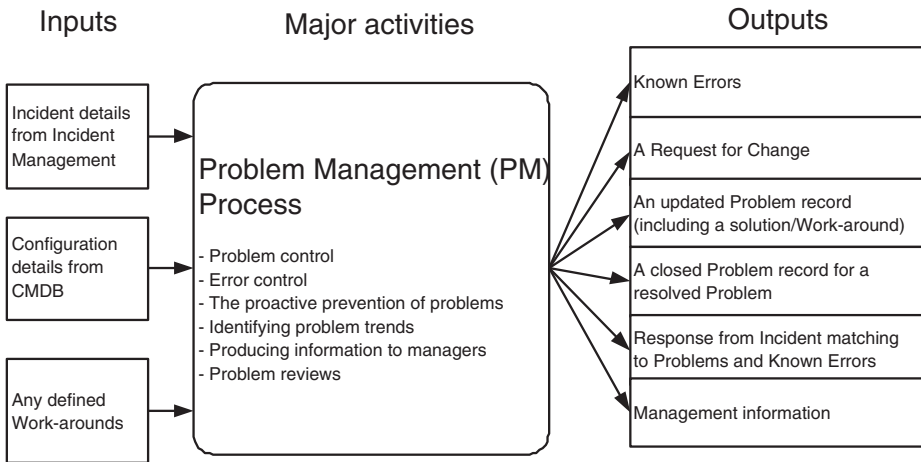


Fig. 2. The inputs, outputs and key activities of the problem management process

This is a traditional task of any service desk or help desk. The proactive problem management in turn tries to prevent incidents before they occur. Proactive activities work in a similar way than defect prevention activities [24]. Reactive problem management is divided into problem control and error control activities where the problem control is responsible for identifying the root cause of a problem [26] and defining a temporary solution (work-around) for the problem.

In order to improve a process, we need information on its inputs, outputs and phases (or stages). In Figure 2, the inputs, outputs and key activities of the problem management process are described.

A frequently asked question is when does the incident management process stop and when does problem management process start? Problem control (the first phase of the reactive problem management process) begins when incident analysis reveals repetitive incidents, or the incident does not match any of the existing problems or known errors. Additionally, when incidents are defined as very serious and significant, they are sent directly to problem control.

The content of problem control and error control phases are explained in ITIL process description and in our previous work. First, the problem management team identifies and records the problem (enters a basic description of problem). Second, the problem management team classifies the problem (defines category, impact, urgency and priority). Third, the problem is investigated and given a diagnosis including the root cause of the problem. In this phase problem management might create a Request for change to implement a problem resolution. *A Request for change* is "a formal part of the change management process, used to record details of a request for a change to any configuration item (CI) within an infrastructure, or to services, procedures and items associated with the infrastructure" [1].

The second frequently asked question is when does the problem control activity stop and when does the error control activity start? The error control

process focuses on correcting known errors by generating request for changes to change management. The basic activities of error control are error identification, error assessment, recording error resolution. Errors can be found both from users and customers (live environment) and testing and development (development environment). According to ITIL "known error status is assigned when the root cause of the problem is found and a workaround has been identified" [1]. We have interpreted this rule that the error control activity begins when the root cause of the problem is a defect (in code or in the IT infrastructure). Error assessment consists of identifying means how error could be resolved. Error resolution might require a contact to third-party service providers or technology providers. Finally, the error resolution (symptoms and resolution actions) should be recorded into known error database or a knowledge base. A knowledge base is a database for knowledge management. It can be used for collecting, organizing, and searching the knowledge. The knowledge base usually provides its users with solutions to known problems [27], [28]. A well-designed knowledge base helps both service desk and customers to find solutions for problems quickly.

3.3 Improving Customer Support Processes: Case Alfa

Table 1 shows the general findings related to the support process of Alfa. Data was collected in the research project meetings between Alfa and the research team.

Customer calls and emails are assigned to an outsourced service desk. The service desk is responsible for the first line support. If the service desk is not able to resolve the problem, the help desk issue (incident) is escalated to the second line support. Alfa also has third line support that is responsible for handling business-related service requests such as overcharging on the Alfa e-commerce site. The performance of the support process is measured by using various metrics such as the number of customer calls answered in less than 20 seconds and the number of missed calls. Service desk and incident management uses an open source based request tracking application that is configured to Alfa's needs.

The service desk records all customer contacts (incidents and service requests) into a request tracking application. Although the service desk has been outsourced to an external service provider, the second line support and a quality manager of Alfa are also able to monitor all the recorded customer requests and problems. Alfa uses a term "Problem Ticket" for customer requests and problems. The same Problem Ticket goes through the whole support process. Problem Tickets are categorized by types (for example, license problem, question, installation).

Problem tickets are not formally prioritized when they are recorded into request tracking application although it is possible to define the priority level for the problem record. However, the second line support performs some kind of prioritization for tickets. Alfa's quality manager stated that there has been no need to prioritize cases because Alfa is able to resolve most cases within a couple of days without priority levels. The request tracking application also includes a module for managing frequently asked questions. Alfa uses this FAQ module to

Table 1. The support process of an IT company Alfa

Factor	Alfa's way to do things
Goals for SPI	Adapt ITIL concepts to the support process Implement a knowledge base with a multi-language support Decrease the number of support requests
Tools	Open source issue management tool
Workflow asset	Issue
Service Support Levels	1st line: Service Desk 2nd line: Support engineer 3rd line: sales
Metrics	Number of support requests/month % of missed calls Number of support requests by type Number of open requests/week % of calls answered < 20 seconds, < 30 seconds Number of support requests resolved within 1 working day, next day, 3rd day
Proactive methods	Training of retailers Monitoring user forums A static FAQ column on the support site Testing

share information within the organization. The FAQ module enables publishing knowledge base articles to support the work of the first-line and second-line support. However, Alfa second line support stated that it is difficult to publish knowledge base articles for customers through the FAQ module because of the poor language support. Alfa would like to obtain a dynamic FAQ application that would decrease the number of the customer contacts to the service desk.

Problems that are assigned to the second-line support are investigated. If the root cause of the problem is an application bug, the problem will be recorded as a bug into bug management application. The traceability between the request tracking application and the bug management application is maintained by recording the request ID to the bug record. Thus, there is a connection between a customer request/problem and a bug. Alfa does not have a process of handling request for changes. Required changes are usually implemented in the next version of the application.

4 Analysis

The analysis of case study focused on identifying strengths and challenges in Alfa's support processes. We analyzed whether Alfa's support process includes the same activities, roles, and tasks than our improved support process description.

Table 2. The strengths and challenges of Alfa's support process

Strengths	Challenges
1. Customer contacts are sent to the Service Desk (SD), which takes care of the first line support.	1. The support process documentation does not include activities or roles of SD and incident management.
2. Measurable targets have been defined for the SD and the incident management processes. The progress of Incident handling is monitored.	2. Measurable goals have not been defined for the problem management process.
3. Employees in the SD are well educated (customer service, technical knowledge) and people are trained continuously (e.g. new features in products).	3. ITIL concepts are not visible in the support process.
4. All the reported cases are recorded to the incident database and those cases are updated when necessary.	4. The lack of a public knowledge base
5. Incidents are categorized according to predefined instructions.	5. Customers do not receive an incident confirmation receipt.
6. There is a bridge between the problem and error control: problem id is stored in the error record to maintain the traceability between those cases.	6. There is no unified tool to handle incidents, problems and defects.
7. Three support levels are recognized in Service Support process (outsourced SD, Problem Management and third line support).	7. Priority of the incident or the problem is not defined in SD or in problem management.
8. Problems concerning the products of third party providers are stored and handled.	8. The processes and activities of problem control, error control and proactive problem management are poorly defined.
9. Standardized reports are produced regularly about incident and problem rates.	
10. The organization has recognized the need for proactive problem management.	

Table 2 presents the results of the analysis (strengths and weaknesses in Alfa's support processes).

The research question in this paper was: what is the role of the problem management process in customer support? In our case organization Alfa, problem management process and its key activities (identify, classify, investigate, resolve problem) were poorly visible in the customer support process. We consider this as a major challenge. However, interviews revealed that these activities were performed by a support engineer. Additionally, Alfa's support process included no documentation on proactive problem management activities, such as trend analyses and preventive actions. In addition to the challenges, we identified a lot of positive things regarding Alfa's customer support, for example, they used three

support levels, they monitored the performance of the outsourced service desk daily, they recorded errors related to the services of 3rd party service providers, and used diverse metrics to monitor the recording and resolution of support requests.

5 Discussion and Conclusions

In this paper, the research question was: what is the role of the problem management process in customer support? Problem management plays a key role in customer support (the 2nd support level), as well as incident management. Unfortunately, the activities of problem management are often very poorly documented compared to incident management activities. As the main contribution of this study we first presented a theory-based model for customer support (with strong focus on problem management) by using a constructive research method. We defined the process inputs, main activities and the process outputs for the problem management process and presented an improved process model where activities of different support processes were described. Our process model provides a rapid overview how support requests move forward in the organization between different processes. A well-defined process description creates a basis for the process improvement. Some frequently asked questions related to the ITIL-based problem management were also discussed, such as when does the incident management process stop and when does problem management process start and what is the interface between problem control and error control.

Second, we presented results of a case study that focused on improving the support process in a medium-sized Finnish IT company "Alfa". Alfa's support process was examined and analyzed whether it includes the same activities, roles, and tasks than our improved support process description. As a key improvement area we identified that Alfa's support process documentation does not include a comprehensive definition of the activities of service desk, incident management and problem management. ITIL based concepts such as knowledge base, incidents, and known errors are not visible in the process description. However, Alfa's support process also had several strengths. For example, they had defined three support levels, they monitored continuously reported problems and had defined a wide selection of metrics for monitoring the performance of support process. The contributions of this paper can be used by problem managers and quality managers to improve problem management processes.

As with all case studies, there are following limitations and validity threats regarding our study. The case study methodology has received a frequent criticism concerning the generalization of research results. If a study is based on a single case, it cannot provide statistical generalizations. However, we can expand the theory. Regarding construct validity, researchers should ensure that the data is collected from several sources. In this study, the main sources of data collection were a quality manager and a support engineer of Alfa. Product developers did not participate in the meetings. We used the word "improved model" because the process model solved some problems that we identified in our previous studies.

However, more research work is needed to validate our model in practice and to define suitable metrics for support processes.

In future studies we intend to improve our research framework by examining proactive problem management from IT customer's viewpoint. We shall also continue the introduction of ITIL-based support processes with new case organizations. Additionally, we need to collect empirical data to show that our recommendations for problem management will lead to better results.

Acknowledgment

This paper is based on research in MaISSI and SOSE projects, funded by the National Technology Agency TEKES, European Regional Development Fund (ERDF), and industrial partners.

References

1. Office of Government Commerce: ITIL Service Support. The Stationary Office, UK (2002)
2. COBIT 4.0: Control Objectives for Information and related Technology: COBIT 4.0. IT Governance Institute (2005)
3. IEEE Standard 1044-1993: IEEE Standard Classification for Software Anomalies. IEEE (1994)
4. Florac, W.: Software quality measurement a framework for counting problems and defects. Technical Report CMU/SEI-92-TR-22 (1992)
5. Card, D.N.: Learning from our mistakes with defect causal analysis. IEEE Software 15(1), 56–63 (1998)
6. Office of Government Commerce: ITIL Service Delivery. The Stationary Office, UK (2002)
7. Leszak, M., Perry, D.E., Stoll, D.: A case study in root cause defect analysis. In: ICSE 2000: Proceedings of the 22nd international conference on Software engineering, pp. 428–437. ACM Press, New York (2000)
8. Kruchten, P.: The Rational Unified Process: An Introduction. Addison-Wesley, Reading (2001)
9. Binder, R.: Testing Object-Oriented Systems: Models, Patterns, and Tools. Addison-Wesley, Reading (2000)
10. Pink Elephant: Itil process maturity. Pink Elephant Whitepaper (2005), <http://www.pinkelephant.com/en-US/ResourceCenter/PinkPapers/PinkPapersList.htm>
11. Jalote, P.: CMM in Practice, Processes for Executing Software Projects at Infosys. Addison-Wesley, Reading (2000)
12. Niessinka, F., Clerca, V., Tjeldink, T., van Vlietb, H.: The it service capability maturity model version 1.0. CIBIT Consultants&Vrije Universiteit (2005)
13. ISO/IEC: ISO/IEC 20000 A Pocket Guide. Van Haren Publishing (2006)
14. ISO/IEC 12207: Information Technology Software Life-Cycle Processes. ISO/IEC Copyright Office (1995)
15. Microsoft: Microsoft operations framework (2007), <http://www.microsoft.com/technet/solutionaccelerators/cits/mo/mof/default.aspx>

16. Jacobson, I., Booch, G., Rumbaugh, J.: *The Unified Software Development Process*. Addison-Wesley, Reading (1999)
17. Quality Assurance Institute: A software defect management process. Research Report number 8 (1995)
18. Hirmanpour, I., Schofield, J.: Defect management through the personal software process. *Crosstalk, The Journal of Defense Software Engineering* (2003)
19. Kajko-Mattsson, M.: A conceptual model of software maintenance. In: *ICSE 1998: Proceedings of the 20th international conference on Software engineering*, pp. 422–425. IEEE Computer Society, Washington (1998)
20. April, A., Hayes, J.H., Abran, A., Dumke, R.: Software maintenance maturity model (smmm): the software maintenance process model: Research articles. *J. Softw. Maint. Evol.* 17(3), 197–223 (2005)
21. Eierman, M.A., Niederman, F., Adams, C.: Dss theory: a model of constructs and relationships. *Decis. Support Syst.* 14(1), 1–26 (1995)
22. Yin, R.: *Case Study Research: Design and Methods*. Sage Publishing, Beverly Hills (1994)
23. Eisenhardt, K.: Building theories from case study research. *Academy of Management Review* 14, 532–550 (1989)
24. Mays, R.G., Jones, C.L., Holloway, G.J., Studinski, D.P.: Experiences with defect prevention. *IBM Syst. J.* 29(1), 4–32 (1990)
25. CEN Workshop Agreement CWA 14924-1: *European Guide to Good Practice in Knowledge Management, Part 1*. European Committee for Standardization (2004)
26. Zhen, J.: It needs help finding root causes. *Computerworld* 39(33), 26 (2005)
27. Davis, K.: Charting a knowledge base solution: empowering student-employees and delivering expert answers. In: *SIGUCCS 2002: Proceedings of the 30th annual ACM SIGUCCS conference on User services*, pp. 236–239. ACM Press, New York (2002)
28. Jackson, A., Lyon, G., Eaton, J.: Documentation meets a knowledge base: blurring the distinction between writing and consulting (a case study). In: *SIGDOC 1998: Proceedings of the 16th annual international conference on Computer documentation*, pp. 5–13. ACM Press, New York (1998)

Influential Factors on Incident Management: Lessons Learned from a Large Sample of Products in Operation

João Caldeira and Fernando Brito e Abreu

Faculty of Sciences and Technology
Universidade Nova de Lisboa
Lisboa, Portugal

j.caldeira@fct.unl.pt, fba@di.fct.unl.pt

Abstract. Understanding causal relationships on incident management can help software development organizations in finding the adequate level of resourcing, as well as improving the quality of services they provide to their end-users and/or customers. This paper presents an empirical study conducted upon a sample of incident reports recorded during the operation of several hundred commercial software products, over a period of three years, on six countries in Europe and Latin America. The underlying research questions refer to the validation of which are the influencing factors affecting the incidents management lifecycle. Nonparametric analysis of variance procedures are used for testing hypotheses.

Keywords: Software development, Empirical Software Engineering, Software Quality, ITIL, Incident Management, Problem Management, Release and Deployment Management, Services Science.

1 Introduction

"If you want the present to be different from the past, study the past."
Baruch de Spinoza (1632-1677)

1.1 Motivation

Organizations with in-house software development strive for finding the right number of resources (with the right skills) and adequate budgets. A good way to optimize those figures is avoiding expenditures on overhead activities, such as excessive customer support. This can be achieved by identifying incident's root causes and use that knowledge to improve the software evolution process.

Software development and software quality improvement have been strong topics for discussion in the last decades [1, 2]. Software Engineering has always been concerned with theories and best practices to develop software for large-scale usage. However, most times those theories are not validated in real-life environments [3]. Several factors were identified that explain this lack of experimental validation [4].

In real-life operation environments end-users/customers face software faults, lack of functionalities and sometimes just lack of training. These incidents should be somehow reported. According to the ITIL¹ framework [5], in an organization with a Service Management approach [6-10], this problem is addressed by two specific processes: *Incident Management* [6], which deals with the restoration of the service to the end-user within the Service Level Agreements [7, 10] (if they exist), and *Problem Management* [6] which aims at finding the underlying cause of reported incidents.

When an organization implements these ITIL processes, then it will address all kind of incidents (software, hardware, documentation, services, etc) raised by the end-users/customers. In this paper we are only concerned about software-related incidents.

The incidents database is an important asset for software engineering teams. Learning from past experience in service management, allows shifting from a reactive approach to a more proactive one. The latter is referred in the Software Maintenance chapter of the SWEBOK² (see Table 1), although seldom brought to practice.

Table 1. Software maintenance categories (source: SWEBOK [11])

	Correction	Enhancement
Proactive	Preventive	Perfective
Reactive	Corrective	Adaptive

This paper presents a statistical-based analysis of software related incidents resulting from the operation of several hundred commercial software products, from 2005 to 2007. The incidents were reported by customers of a large independent software vendor. Although that vendor operates worldwide, we were only able to have access to data from six countries in Europe and Latin America. Further details regarding the products and their users cannot be provided here due to a non-disclosure agreement.

The main goal of this paper is shedding some light on the influential factors that affect incidents lifecycle from creation to its closure, namely the schedule of its phases. Understanding this lifecycle can help software development organizations in allocating adequate resources (people and budget), increasing the quality of services they provide and finally improving their image in the marketplace.

The work presented herein is on the crossroads of Empirical Software Engineering and of the emerging area of Services Science [12, 13]. It is organized as follows: section 2 presents a survey of related work; section 3 contains the empirical study; finally, section 4 presents the conclusions, the threats to validity and future work.

¹ The IT Infrastructure Library (ITIL) is the *de facto* standard for IT service management, an initiative launched in the late 80's by the UK Office of Government Commerce's (OGC). The ITIL framework is as a generic reference model proposing a set of concepts and good practices for managing information technology, infrastructure, development and operations.





² The Guide to the SoftWare Engineering BOdy of Knowledge (SWEBOK) is an IEEE CS initiative aiming to: (i) promote a consistent view of software engineering worldwide, (ii) clarify the place and set the boundary of software engineering with respect to other disciplines such as computer science, project management, computer engineering, and mathematics, (iii) characterize the contents of the software engineering discipline, (iv) provide a topical access to the Software Engineering Body of Knowledge and (v) provide a foundation for curriculum development and individual certification and licensing material.

2 Related Work

To support our research, we have tried to find related work in the area of empirical software engineering within the ITIL scope. Having searched several digital libraries such as the ones of ACM, IEEE, Springer or Elsevier, we were able to find only a few papers about incident management. Even scarcer were those referencing real-life empirical studies on software incidents and how they that can help improving the software engineering process. This section presents a categorized overview of the published works that we found to be closer-related to our work presented hereafter.

2.1 Categorization Process

ITIL is concerned about three basic aspects in IT Service Management (ITSM): technology, people and processes. The technology aspect refers to all the technical components (typically hardware and software) involved when dealing with IT services. The people aspect addresses the way persons are organized and the way they should behave when involved in a certain process. Finally, the process aspect relates to how activities are linked together in order to deliver value to a specific business area. We categorized the related work according to the extent it has approached those ITSM aspects. For classifying each of the aspects, we use the following ordinal scale:

- | | |
|--|---|
| <p>Absent /
Fuzzy view</p> <p>Partly /
Isolated view</p> <p>Largely /
Contextualized view</p> <p>Fully /
Holistic view</p> | <p> The topic is not addressed or addressed in a fuzzy way</p> <p> The topic is addressed insufficiently, not explicit or lacking context</p> <p> The topic is addressed explicitly and context is provided, although not exhaustively</p> <p> The topic is addressed exhaustively, sustained with evidence and with adequate rationale being provided</p> |
|--|---|

Besides that categorization, we provide, for each work, its main goal (as we perceived it), a commented abstract and, finally, we comment about the relation each work has with ours. Notice that we have kept the capitalized denomination of ITIL processes (e.g Incident, Problem or Configuration Management).

2.2 Review of Related Work

Barash et al. (2007) [14]

Technology	People	Processes
		

Goal – Managing service incidents and improving an IT support organization.

Comments – This work has a clear link with ITIL. The main topics addressed are Incident and Problem Management and the improvement an organization can achieve in their support activities by analyzing incident metrics. The authors suggest ways to improve staff allocation, shift rotation, working hours and the escalation of incidents.

We could not find, in this work, a clear link between Incident or Problem Management processes with the software development process and how they can help each other in improving the quality of the service to the end-users. We also could not

find a direct relationship to any other ITIL processes beyond the two referred ones. Nevertheless, we should not forget that if we improve the performance of the IT support organization, we are indirectly improving the performance of all other areas.

Relation with Our Work – This work is related with our own since it also addresses the management of incidents (herein we only address software incidents), and it tries to improve an IT Support Organization.

Sjoberg et al. (2005) [3]



Goal – A survey of controlled experiments in Software Engineering.

Comments – In this work there is a detailed classification about the areas where those software experiments were conducted. It is interesting to realize that among the group of areas with fewer experiments, we find Strategy, Alignment, IT impact. These are within the most important issues addressed by ITIL and Service Management. One of the things that first came to our eyes is the fact that there is no category named “Service”. We can assume that within all experiments done, none was made having the “Service” in mind. This is even more important since nowadays services are heavily dependent on software, and, on the other hand, the use of software can be seen as a service on its own. Overall, this work is a quantitative summary of controlled experiments. While the people and the processes aspects are briefly addressed, the technology aspect is only slightly covered. Indeed, few environment descriptions are provided on the technical conditions on which the experiments took place.

Although this survey was performed around three years ago, we have not found evidence, since then, contradicting the obvious need of more experiments relating software, services and their management processes.

Relation with Our Work – We expected that other studies like the one performed in our paper would be reported in this survey. While on the methodology side this is true, since many of the reported experiments use empirical data and statistical analysis, the same cannot be said regarding the context (incident management).

Niessink and Vliet (2000) [15]

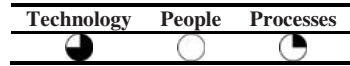


Goal – Software maintenance and software development from a service perspective.

Comments – The authors clearly identify differences between services and products and how these differences affect the way end-users or customers assess their quality. One of the more relevant aspects of this work is the focus put on the need for defining Service Level Agreements (SLA), Service Catalogs and the importance of good Incident and Problem Management processes within an organization. These three aspects and the positive impact they can have in organizations that implement them are highlighted and understood, but not exhaustively explained. This would be addressed by detailing and giving examples on the implementation of the above aspects. In brief, the important topics are there, but not enough detail is provided.

Relation with Our Work – The relation lies on the ITIL focus. This is not an empirical study, but it covers all the important aspects of Service Management.

Jansen and Brinkkemper (2006) [16]

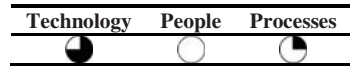


Goal – Study of the release, delivery and deployment of software.

Comments – This is a very interesting paper about the software update process and how it can help software vendors and end-users/customers in the software deployment process. The approach taken fits in the realm of the Asset and Configuration Management, Release and Deployment Management ITIL processes. Notice that the deployment phase, which is focused in this paper, is precisely the one when most incidents are usually reported. This is due to the fact that IT systems and platforms are becoming increasingly more heterogeneous and complex and also because quality management systems (in general) and SLA verification (in particular) imply the recording of incidents originated by the operation.

Relation with Our Work – This work focuses on the technology used to improve the software deployment process, but does not cover any empirical study or data analysis. It is related to our work because it touches another key process in ITIL.

Mohagheghi and Conradi (2007) [17]



Goal – Quality, productivity and economic benefits of software reuse.

Comments – This work is about software reuse and its benefits. Based on previous studies, the authors state that component reuse is related with software with fewer defects. The latter are identified by means of failures in operation and are the origin of reported incidents. The end-user perspective is not covered in this paper, and this is vital for a Service Management approach. Some references are made to software changes, software deployment and even infrastructure resources required for software execution. These are somehow implicit references to ITIL Change Management, Release and Deployment Management and Capacity Management processes.

Relation with Our Work – This work shares our objective of achieving a tangible and positive impact on the software development process by adopting ITIL-like best practices. This has strengthened our conviction that the impact of incident management on the software development lifecycle deserves further analysis.

2.3 Review Summary

It is widely accepted that we lack experimentation in Software Engineering in general. This phenomenon is even more acute on what concerns experimentation related with incidents and services. As Spinoza observed more than 300 years ago, we need to understand how services were provided in the past to improve their quality in the future. Even if the related work is scarce, we should look at it collectively to try drawing some picture of the current state-of-the-art. For that purpose, a summary of the categorized related work is presented in Table 2.

Out of the three aspects, the one that deserves the least attention is clearly “people”, while the “technology” and “process” aspects have somehow equivalent emphases. We believe that this difference is due to the fact that researchers working in this area have mostly an Engineering background. Understanding people and their motivations requires Social Sciences skills.

Table 2. Summary of related work

Proposal	Technology	People	Processes	Relation
Barash et al. (2007)				High
Sjoberg et al. (2005)				Medium
Niessink and Vliet (2000)				High
Jansen and Brinkkemper (2006)				Low
Mohagheghi and Conradi (2007)				Low

However, the most relevant conclusion we reached while performing this unambitious state-of-the-art survey, is that the empirical study of incident management has not yet been adequately addressed in the scientific literature. We believe this situation is due to the fact that real-life samples contain sensitive data to companies and so are usually unavailable to researchers.

3 The Empirical Study

3.1 Process and Instrumentation

Our empirical process consisted on the four steps represented in Fig. 1. We collected the data on the first days of January 2008, using an incident management system client interface. This tool allowed to export incidents data into a CSV (Comma Separated Values) file that could be loaded into a spreadsheet (MS Excel). Next, we filtered out a very small percentage of cases that had erroneous data (e.g. invalid dates). Then, we computed several variables from existing data, namely by calculating differences between pairs of dates.. The resulting dataset was then loaded into the SPSS statistical analysis tool, where the statistical analysis took place,



Fig. 1. Empirical study workflow

3.2 The Sample

The subjects of our empirical study are around 23 thousand incidents, reported by end-users/customers, occurred during the operation of around 700 software products³. The incidents were recorded with a proprietary incident management system during a time span of three years (2005 to 2007) in around 1500 companies in 6 countries.

We considered three geographical zones, with two countries in each one. The zones are Latin America (LA), Southwestern Europe (SE) and Central Europe (CE). Notice that there are 4 languages spoken in the considered countries: English (EN), French (FR), Portuguese (PT) and Spanish (ES). More details are provided in Table 3.

³ When a given product is available on different platforms, this number considers those instances as distinct products. Some distinction is also due to different licensing schemes.

Table 3. Countries with their zones and languages

Country	Zone	Language	# of Incidents	# of Customers	# of Software Products
England (UK)	CE	EN	7349	530	460
France (FR)	CE	FR	8237	554	444
Spain (ES)	SE	ES	4014	219	359
Argentina (AR)	LA	ES	535	66	88
Portugal (PT)	SE	PT	556	37	107
Brazil (BR)	LA	PT	2221	125	250
Total			22912	1531	

3.3 Descriptive Variables

The variables used in this empirical study are self-described in Table 4. The choice on the characterization of the incidents (*Category*, *Impact* and *Priority*) is performed by the person who registers the incident (the end-user/customer or a support staff member). Incidents have a defined lifecycle. In this paper we will only consider closed incidents, since those are the only ones for which we know the values of all timing variables. Fig. 2 describes how the three timing variables are calculated, regarding specific milestones in the incidents' lifecycle.

Table 4. Variables used in this empirical study, their scale types and description

Variable	Scale	Description
<i>Product</i>	Nominal	Name of the product causing the incident
<i>Company</i>	Nominal	Name of the company where the product is installed
<i>Country</i>	Nominal	Name of the country where the incident was originated
<i>Zone</i>	Nominal	Zone of the globe where the country lies
<i>Language</i>	Nominal	Language spoken in the country
<i>Category</i>	Nominal	Represents incident's root cause <i>Valid values are: 3rd Party Solution, Customer Support, Customization, Documentation, Function, Installation, Internationalization, Compatibility, Licensing, Localization, Performance, RFI, Security Threat, Stability, Education, Uncategorized</i>
<i>Impact</i>	Ordinal	Measures incident's business criticality <i>Valid values are: 1-Critical, 2-High, 3-Medium, 4-Low</i>
<i>Priority</i>	Ordinal	Measures incident's correction prioritization as seen by the support ⁴ Valid values are the same as for the impact
<i>Status</i>	Nominal	Current status of the incident in its life cycle
<i>WeekOfCreation</i>	Interval	Order of the week (in the year) when the incident occurred <i>Valid values belong to the interval [1, 53]</i>
<i>WeekdayOfCreation</i>	Interval	Order of the day (in the week) when the incident occurred. <i>Valid values belong to the interval [1, 7]</i>
<i>TimeToRespond</i>	Absolute	Elapsed time from incident creation until a support person has started to work on it
<i>TimeToResolve</i>	Absolute	Elapsed time from incident creation until a resolution is given to the end-user
<i>TimeToConfirm</i>	Absolute	Elapsed time since the resolution was given to the end-user until a confirmation is obtained that the incident is closed

⁴ According to ITIL, incidents priority should be calculated based upon urgency and impact. However, the incident management system used in this study does not yet support the concept of urgency. The priority is assigned directly by the incident recorder.

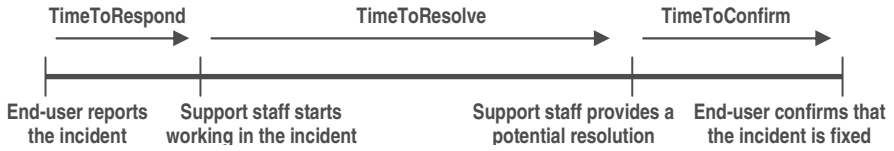


Fig. 2. Incidents' lifecycle timing variables

3.4 Research Questions

To understand incident management we must find answers for these two questions:

Q1: Which factors influence the lifecycle of incidents?

Q2: Are there patterns in the occurrence of incidents?

Regarding Q1, the set of variables that best describe incidents lifecycle at a macroscopic level are *TimeToRespond*, *TimeToResolve* and *TimeToConfirm*. The answer to Q1 is important both to clients and service providers. For clients, particularly for large organizations operating in several countries, it will allow taking decisions in the formulation and negotiation of Service Level Agreements (SLAs). For service providers it will also help in finding the adequate level of staffing.

Regarding the possible factors influencing the incidents lifecycle, we can consider the following variables inscribed in Table 4: *Product*, *Company*, *Country*, *Zone*, *Language*, *Category*, *Impact* and *Priority*. We have selected the following research questions within the scope of this paper:

- **Has the impact of an incident an influence on its lifecycle?**
- **Has the priority of an incident an influence on its lifecycle?**
- **Has the originating country of an incident an influence on its lifecycle?**
- **Has the originating geographical zone of an incident an influence on its lifecycle?**
- **Has the language spoken in the country where the incident was reported, an influence on its lifecycle?**
- **Has the incident category an influence on its lifecycle?**

Regarding Q2, the occurrence of incidents can be measured by a simple counting or a weighted sum (e.g. taking the *Impact* or *Priority* as a weight) of incidents matching one of the possible values of the variable under consideration. For instance, if we were concerned with the identification of seasonal patterns, we can consider the day within the week (*WeekdayOfCreation*) or the week within the year (*WeekOfCreation*) when the incidents were reported. Again, the answer to Q2 will bring benefits to client and service provider. Both will become aware of worst and best-case scenarios and thus take appropriate actions.

Due to the lack of space, we have just considered here a possible pattern, which is the distribution of critical incidents, the ones which give more headaches to all stakeholders. In this case, since the incidents were recorded using the same incident management system and supposedly using similar classification criteria, we would expect

the proportion of critical incidents to be the same across countries. In other words, the corresponding research question is simply:

- **Is the distribution of critical incidents the same across countries?**

3.5 Hypotheses Identification and Testing

In this section we identify which are the statistical hypotheses that must be tested in order to answer the previously stated research questions. We then apply the adequate statistical tests and interpret their results. Research questions are prefixed by “**RQ**”.

RQ: Has the Impact of an Incident an Influence on Its Lifecycle?

In other words, we want to know if incidents with different assigned impacts differ in the corresponding lifecycle schedules (*TimeToRespond*, *TimeToResolve*, *TimeToConfirm*). Notice that the *Impact* category is assigned by the person that records the incident in the incident management system at the time of its creation.

Due to the fact that those schedules are not normally distributed, we can only perform a non-parametric analysis of variance. We will use the Kruskal-Wallis one-way analysis of variance, an extension of the Mann-Whitney U test, which is the nonparametric analog of one-way ANOVA test. The Kruskal-Wallis H test allows assessing whether several independent samples are from the same population (i.e. if they have similar statistical distributions). In our case those independent samples are the groups of incidents for each of the four *Impact* categories.

Let T be a schedule and i and j two different impact categories. Then, the underlying hypotheses for this test are the following:

$$H_0: \forall_{i,j}: T_i \sim T_j \quad \text{vs.} \quad H_1: \neg \forall_{i,j}: T_i \sim T_j$$

Table 5. Testing the influence of the impact on incident schedules with the Kruskal-Wallis one-way analysis of variance test

	TimeToRespond	TimeToResolve	TimeToConfirm
Chi-Square	352.381	77.532	18.487
df	3	3	3
Asymp. Sig.	.000	.000	.000

The Kruskal-Wallis H test statistic is distributed approximately as chi-square. Consulting a chi-square table with $df = 3$ (degrees of freedom) and for a significance of $\alpha = 0.01$ (probability of Type I error of 1%) we obtain a critical value of chi-square of 11.3. Since this value is less than the computed H values (for each of the schedule variables in Table 5), we reject the null hypothesis that the samples do not differ on the criterion variable (the *Impact*). In other words, given any of the schedule variables, we cannot sustain that the statistical distributions of the groups of incidents corresponding to each of the *Impact* categories are the same. This means that we accept the alternative hypothesis that **the impact of an incident has influence on all the schedule variables.**

RQ: Has the Priority of an Incident an Influence on Its Lifecycle?

Here we want know if incidents with different assigned priorities differ in the corresponding lifecycle schedules (*TimeToRespond*, *TimeToResolve*, *TimeToConfirm*). We will follow the same rationale as for the previous research question, regarding the applicable statistic and its interpretation.

Table 6. Testing the influence of the priority on incident schedules with the Kruskal-Wallis one-way analysis of variance test

	TimeToRespond	TimeToResolve	TimeToConfirm
Chi-Square	298.918	80.868	13.210
df	3	3	3
Asymp. Sig.	.000	.000	.004

Again the critical value of chi-square for ($df = 3, \alpha = 0.01$) = 11.3. Since this value is less than the computed H values for each of the schedule variables in Table 6, we reject the null hypothesis that the samples do not differ on the criterion variable (the *Priority*). In other words, given any of the schedule variables, we cannot sustain that the statistical distributions of the groups of incidents corresponding to each of the *Priority* categories are the same. This means that we accept the alternative hypothesis that **the priority of an incident has influence on all the schedule variables.**

RQ: Has the Originating Country of an Incident an Influence on Its Lifecycle?

The rational for answering this research question is the same as for the previous one. To enable the application of the Kruskal-Wallis test, we have automatically recoded the *Country* variable from string categories into numerical categories.

Table 7. Testing the influence of the originating country on incident schedules with the Kruskal-Wallis one-way analysis of variance test

	TimeToRespond	TimeToResolve	TimeToConfirm
Chi-Square	1666.912	337.181	44.877
df	5	5	5
Asymp. Sig.	.000	.000	.000

Given that the critical value of chi-square for ($df = 5, \alpha = 0.01$) = 15.1. Since this value is less than the computed H values for each of the schedule variables in Table 7, we reject the null hypothesis that the samples do not differ on the criterion variable (the *Country*). In other words, given any of the schedule variables, we cannot sustain that the statistical distributions of the groups of incidents corresponding to each of the countries are the same. This means that we accept the alternative hypothesis that **the country of an incident has influence on all the schedule variables.**

RQ: Has the Originating Geographical Zone of an Incident an Influence on Its Lifecycle?

The rational for answering this research question is again the same as for the previous one. To enable the application of the Kruskal-Wallis test, we have automatically recoded the *Zone* variable from string categories into numerical categories.

Table 8. Testing the influence of the originating zone on incident schedules with the Kruskal-Wallis one-way analysis of variance test

	TimeToRespond	TimeToResolve	TimeToConfirm
Chi-Square	1546.415	139.297	17.727
df	2	2	2
Asymp. Sig.	.000	.000	.000

Given that the critical value of chi-square for $(df = 2, \alpha = 0.01) = 9.21$, we reject the null hypothesis that the samples do not differ on the criterion variable (the *Zone*). In other words, given any of the schedule variables, we cannot sustain that the statistical distributions of the groups of incidents corresponding to each of the geographical zones are the same. Then we accept the alternative hypothesis that **the geographical zone where the incident was reported has influence on all the schedule variables.**

RQ: Has the Incident Category an Influence on Its Lifecycle?

Again, after performing an automatic recode (for the *Category* variable), we obtained the following summary table:

Table 9. Testing the influence of the category on incident schedules with the Kruskal-Wallis one-way analysis of variance test

	TimeToRespond	TimeToResolve	TimeToConfirm
Chi-Square	837.595	1258.178	612.215
df	15	15	15
Asymp. Sig.	.000	.000	.000

Given that the critical value of chi-square for $(df = 15, \alpha = 0.01) = 30.6$, we reject the null hypothesis that the samples do not differ on the criterion variable (the incident *Category*). In other words, given any of the schedule variables, we cannot sustain that the statistical distributions of the groups of incidents corresponding to each category are the same. This means that we accept the alternative hypothesis that **the incident category has influence on all the schedule variables.**

RQ: Is the Distribution of Critical Priority Incidents the Same Across Countries?

Since we know the proportion of total incident reports originated in each country (see Fig. 3) we can expect that the incidents with critical priority per country follow the same proportion of values. For this purpose we will use the Chi-Square Test procedure that tabulates a variable into categories and computes a chi-square statistic. This non-parametric goodness-of-fit test compares the observed and expected frequencies in each country to test if each one contains the same proportion of values.

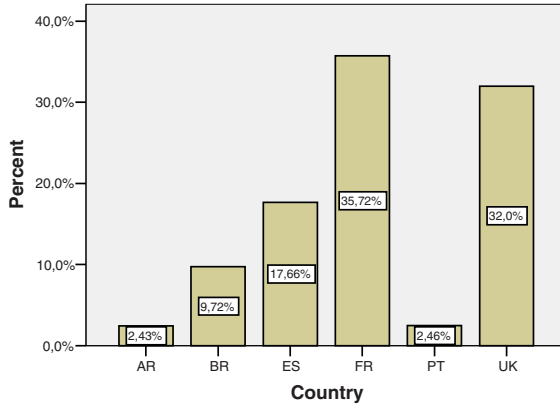


Fig. 3. Percentage of incident reports per country

To apply this test we only selected the critical incidents and obtained the results displayed in Table 10. Since the critical value of the chi-square for (df = 5, $\alpha = 0.01$) = 15.1, we reject the null hypothesis that the proportion of critical priority incidents is the same across countries. This means that we accept the alternative hypothesis that **the proportion of critical priority incidents is different across countries.**

Table 10. Results of applying the Chi-Square Test procedure to assess if the distribution of critical priority incidents is the same across countries

	Observed N	Expected N	Residual
AR	12	17.8	-5.8
BR	39	71.2	-32.2
ES	154	129.3	24.7
FR	198	261.5	-63.5
PT	15	18.0	-3.0
UK	314	234.3	79.7
Total	732		

	Country
Chi-Square	64.203
df	5
Asymp. Sig.	.000

4 Conclusions and Future Work

4.1 Conclusions

In this paper we obtained statistically significant evidence that several independent variables (*Impact, Priority, Country, Zone* and *Category*) have an influence on incidents lifecycle, as characterized by three dependent variables (*TimeToRespond, TimeToResolve* and *TimeToConfirm*). To assess the intensity of the relationship among the independent and dependent variables we must use appropriate measures of association, but that analysis could not be included in this paper due to space restrictions.

There is no surprise on the influence of incident’s business criticality (the *Impact*) and incident’s correction prioritization recorded by the support (the *Priority*) on incidents lifecycle. After all, those incident descriptors were proposed with that same aim.

Not so obvious is the observed fact that either the country or the geographical zone of an organization reporting an incident, has influence on all descriptive variables that characterize incidents lifecycle. This means that organizations from different countries (or geographical zones) do not receive the same kind of support, although they are using the same products and, in principle, paying approximately the same for it. Several reasons, which we have not been explored yet, may explain this phenomenon:

- exigency on SLAs formalization and compliance verification by clients may somehow differ from country to country;
- cultural differences that cause a distinction on the tolerance to failure by final users (e.g. not complaining because an incident was yet solved);
- language differences that somehow influence the relationship between final users and the international support that is provided by the software vendor worldwide,

The incident category also has a direct influence on the three schedule variables. However, we have many kinds of recorded incidents, ranging from those occurring at software installation, to those related to software functionalities. The incidents can also go from enhancement requests to “true” bugs. This diversity requires a careful study before any interpretation of value can be performed.

Another apparent surprise was the fact that the proportion of critical incidents is not the same across countries. In all countries, except the UK and Spain, the actual number of critical incidents was below the expectation. This may indicate that end-users in those countries are causing an over-grading in incidents critically assessment by the support. Sometimes, end-users/customers tend to think that their incidents have always higher impact, simply because it affects the way they do their work and not based on the impact the incident has on the business. Again, this issue deserves further study before sensible conclusions can be drawn.

We have taken a view of the incident management process inspired by the ITIL approach, thus highlighting the importance of combining efforts to link engineering and management areas.

4.2 Threats to the Validity

The main threats to this empirical study are related with data quality and the incident management process itself.

The main data quality related threats are:

- Data missing and/or wrong data (product name, version, etc) provided from the end-users/customers;
- Wrong data entered by the support staff (priority, impact, categorization, resolution codes, etc).

The main Incident Management process threats are:

- Lack of skills about the support tool can make some information non reliable (time to respond to incidents, time to resolve, etc);
- Customer non-response to a provided solution can cause incidents to be open when in fact they could be closed.

As an external threat to this empirical study, we can point that there is data missing from the software development process (resources allocated, activities, development tools, development methodology, etc.) which could help us to better evaluate and understand some of the results.

4.3 Future Work

This empirical study was built upon a large sample of real-life data on incidents across a large period of time, on a long list of commercial products and customers in different countries. We are conscious that we have only scratched the surface. We plan to continue this work by deeply analyzing all the incidents, their categories, software errors and their causes.

Besides understanding the incident management process, our final aim is proposing some guidelines to cost-effectively improve software quality, based on incident management optimization. These guidelines can be focused on the products that appear to have more reported incidents or simply based on the most frequent incident categories. For this to be done accurately, we plan to collect more data, such as information about software development resources and activities performed during the overall development process.

References

1. Humphrey, W.: *Managing the Software Process*. Addison-Wesley Publishing Company, Boston (1989)
2. El-Eman, K., Drouin, J.-N., Melo, W. (eds.): *SPICE: The Theory and Practice of Software Process Improvement and Capability Determination*. IEEE Computer Society Press, Los Alamitos (1997)
3. Sjøberg, D.I.K., Hannay, J.E., Hansen, O., Kampenes, V.B., Karahasanovic, A., Liborg, N.-K., Rekdal, A.: A survey of controlled experiments in software engineering. *IEEE Transactions on Software Engineering* 9, 733–753 (2005)
4. Jedlitschka, A., Ciolkowski, M.: Towards Evidence in Software Engineering. In: *Proc. of International Symposium on Empirical Software Engineering (ISESE 2004)*, pp. 261–270. IEEE Computer Society Press, Washington (2004)
5. OGC: *The Official Introduction to the ITIL Service Lifecycle Book*. TSO, London (2007)
6. Cannon, D., Wheeldon, D.: *ITIL Service Operation*. TSO, London (2007)
7. Case, G., Spalding, G.: *ITIL Continual Service Improvement*. TSO, London (2007)
8. Iqbal, M., Nieves, M.: *ITIL Service Strategy*. TSO, London (2007)
9. Lacy, S., MacFarlane, I.: *ITIL Service Transition*. TSO, London (2007)
10. Loyd, V., Ruud, C.: *ITIL Service Design*. TSO, London (2007)
11. Abran, A., Moore, J.W., Bourque, P., Dupuis, R. (eds.): *Guide to the Software Engineering Body of Knowledge (SWEBOK)*. IEEE Computer Society Press, Los Alamitos (2004)
12. Abe, T.: *What is Service Science?* Research report nr. 246, Fujitsu Research Institute, Tokyo (2005)
13. Poole, G.: A new academic discipline needed for the 21st century. *Triangle Business Journal* (2007)

14. Barash, G., Bartolini, C., Wu, L.: Measuring and Improving the Performance of an IT Support Organization in Managing Service Incidents. In: Proc. of 2nd IEEE/IFIP International Workshop on Business-Driven IT Management (BDIM 2007), pp. 11–18. IEEE Computer Society Press, Los Alamitos (2007)
15. Niessink, F., Vliet, H.v.: Software Maintenance from a Service Perspective. *Journal of Software Maintenance: Research and Practice* 12(2), 103–120 (2000)
16. Jansen, S., Brinkkemper, S.: Evaluating the Release, Delivery and Development Processes of Eight Large Product Software Vendors applying the Customer Configuration Update Model. In: Proc. of International Workshop on Interdisciplinary Software Engineering Research (WISER 2006) @ ICSE 2006, Shangai, China (2006)
17. Mohagheghi, P., Conradi, R.: Quality, productivity and economic benefits of software reuse: a review of industrial studies. In: *Empirical Software Engineering*, vol. 12(5), pp. 471–516. Kluwer Academic Publishers, Hingham (2007)

Pitfalls in Remote Team Coordination: Lessons Learned from a Case Study

Darja Šmite¹, Nils Brede Moe², and Richard Torkar³

¹ University of Latvia, Latvia

² SINTEF, Norway

³ Blekinge Institute of Technology, Sweden

Darja.Smite@lu.lv, Nils.B.Moe@sintef.no, Richard.Torkar@bth.se

Abstract. As companies become more and more distributed, multi-site development is becoming a norm. However along with the new opportunities, geographic distribution is proven to increase the complexity of software engineering introducing challenges for remote team communication, coordination and control. In this article we present an illustrative single-case study with an intra-organizational intra-national context focussing on the effect of geographic distribution on team coordination practices and how this influences remote team performance. Based on our findings we conclude that a) distribution significantly influences the nature of coordination; b) remote team coordination mechanisms can't be chosen disregarding the complexity of the given tasks and c) the distribution of work on complex software development tasks shall be avoided.

1 Introduction

Distributed software teams, in which members interact with one another across geographic, organizational, and other boundaries, are becoming commonplace in software organizations. Distributed software teams can be composed of the best individuals for the task regardless of their physical or organizational location, thus enhancing the quality of software development. However, problems of geographic distribution, such as decreased speed of work [1, 2], loss of communication richness [2, 3] and coordination breakdown [2, 3] experienced in highly distributed global software engineering projects may also be faced in an intra-organizational intra-national software development environment.

While most opportunities of work distribution are found on the business level, most challenges are introduced at the level of development practice [4]. Lacking knowledge and expertise, managers tend to underestimate the complexity of remote team coordination. Lack of proximity and ability to use well-known proven practices for team coordination makes project managers uncomfortable and insecure. Subsequently, missing trust in a distributed project may even lead to termination of further collaboration [5].

Coordination of work is an important aspect of teamwork and team leadership [19]. Coordination together with communication and collaboration are recognized as the key enablers of software development processes [20].

Motivated by the importance of distributed work and software development teams, the objective of this paper is to explore and discuss the interrelation between geographic distribution, team coordination styles and team performance. The core research questions are therefore:

- RQ1: What are the consequences of geographic distribution on how work is coordinated in a distributed software team?*
- RQ2: What characterizes the effect of different coordinating mechanisms on distributed team performance?*

We first illustrate the nature of remote team coordination with a case study and then use related literature to understand the effect of different coordination mechanisms on the remote team performance.

The rest of the paper is structured in the following way: in section 2 we describe related literature. Section 3 describes our research method in detail. In Section 4, we present results from a case study on work coordination in a distributed environment. We discuss our results and research questions in section 5. Finally, recommendations on how to coordinate work in a distributed project conclude the paper.

2 Related Research Overview

2.1 Coordination of Software Work

Software development consists of solving tasks with different complexity, and different task complexities require different coordination mechanisms [6]. Mintzberg [6] proposes the following coordination mechanisms:

1. Mutual adjustment—based on the simple process of informal communication, achieved by a continuous exchange of information among participants;
2. Direct supervision—one person takes responsibility for the work of others by issuing instructions and monitoring their actions;
3. Standardization—of which there are four types: work processes, output, skills (as well as knowledge), and norms.

The coordinating mechanisms may be considered as the most basic elements of structure, the glue that holds the organization together [6]. The mechanisms may act as substitutes for each other to some degree, but all will typically be found in a reasonably well-developed organization.

Simple tasks are easily coordinated by mutual adjustment, but when work becomes more complex, direct supervision tends to be added and takes over as the primary means of coordination. When things get even more complicated, standardization of work processes (or outputs) take over as the primary coordinating mechanism, in combination with the other two. Then, when things become really complex, mutual adjustment tends to become primary again, but in combination with the others (Fig. 1). This means that even though mutual adjustment should be the most important coordinating mechanism when developing software even in a global context, the need for the different coordinating mechanisms depends on the different tasks.

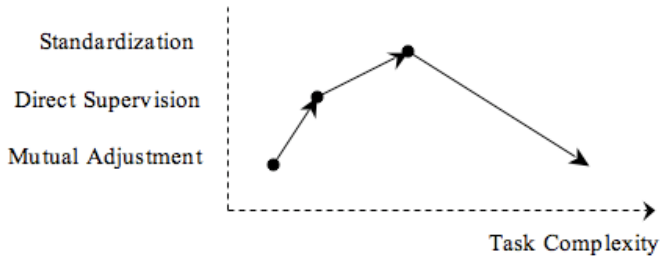


Fig. 1. Complexity of task [6]

Mutual adjustment and direct supervision can be categorized as coordinating by feedback [18], where coordination is adjusted continually as people observe the effects of their own and others' actions. However, geographic distribution is associated with reduction of informal contact, lack of proximity and task awareness [4], which are crucial for applying these coordination mechanisms. One dominant perspective on software development is rooted in the rationalistic paradigm, which promotes a product-line approach to software development using a standardized, controllable, and predictable software engineering process [19]. This perspective is inspired by a mechanistic worldview [20], making standardization of work processes the most important coordinating mechanism. Unfortunately, organizational diversity and disparities in work practices make it difficult to only rely on standardization [5]. These problems are often underestimated and discovered too late in the projects [5].

2.2 Coordination of Distributed Software Work

Research in globally distributed software development (GSD) is in the phase, where case studies of various kinds (in most cases having a qualitative focus) provide input for researchers and organizations to, in the end, improve upon activities in this context. The same applies to this paper and hence relevant work can be found in different contributions where lessons learned, models, problem descriptions and communication patterns are analyzed using a case study approach.

Coordination in distributed development and especially in GSD has caught the attention of several researchers. Cataldo et al. [7] present four case studies, where they exemplify coordination breakdown problems in GSD focusing on how these problems could still occur in spite of supporting tools and processes (one of the findings was to emphasize lateral communication in GSD projects). Holmström et al. (a) [8] present a number of challenges with respect to temporal, geographical and socio-cultural distance and combined with the conclusion that GSD should be trimmed down towards nearshoring. Additionally, Holmström et al. (b) [4] (containing two case studies) investigate how agile processes can help in reducing different types of "distances", i.e. they found some indications that agile practices may assist in alleviating some of the problems in GSD.

Crowston et al. [9] try to explain the performance of teams working for Free/Libre Open Source Software (FLOSS) from a GSD perspective. The conclusion is that the

problems faced by these teams can be recognized as human-centred. In this paper we further examine this issue from an intra-organizational intra-national industry context.

Herbsleb et al. [10] present a number of lessons learned from nine distributed projects. They propose future research on different dimensions of coordination mechanisms and how they play together. In this paper we look at communication patterns, project management issues, the effect of traveling and different communities of practice, i.e. issues which Herbsleb et al. introduced in [10].

Finally, at least three other types of research contributions can be found in the field of coordination research in GSD. First, the quantitative studies (which are fairly uncommon in GSD research) of Herbsleb and Mockus [1], where they discovered that distributed work is significantly slower than co-located through mining source code repositories for further analysis. Second, contributions covering tool support in a GSD context; here Boden et al. [11] and Fomseca et al. [12] can serve as good examples. Third, and final, we find contributions focusing on coordination analysis [13] and predictions [14] that help in better understanding coordination issues in GSD.

3 Research Design and Methodology

3.1 Study Context

To contribute to the existing literature on remote team coordination we exemplify the nature of coordination mechanisms in a geographically distributed environment by a single-case study run in a Northern European software organization nationally distributed across two locations. The development project involved a customer from Norway that engaged the case company that in its turn outsourced some work to its remote location within the national borders. In this study we focused on the intra-organizational relationship, where the central team with several years of experience as a supplier, was now put into an engaging partner role.

The company made a decision to employ developers in one of the poorest regions in the same company for decreasing their development costs. The company rented an office and employed a group of local software engineers all working as developers.

The goal of the project was to deliver a web-based e-commerce application system built around modern technology. The project started in May 2005 and ended in April 2006. In total 1,460 man-days were used exceeding the planned effort by 12%.

The project suffered from ambiguity in the initial set of requirements because a great amount of changes was reported by the end customer during the project. Therefore, schedule deviations were foreseen in advance and negotiated with the customer without extending the deadlines. However this caused extra work during the weekends for the development teams. Additional expenses were not planned by the customer and resulted in the scope of the project being narrowed. As a result, testing activities were limited and several requirements were not implemented. This sequentially decreased the end customer satisfaction.

The quality of the delivered software was perceived as poor by the project manager due to a high amount and severity of bugs uncovered during system testing (1,144 bugs) and acceptance testing (220 bugs). The manager regularly expressed his low satisfaction with the performance of the remote team, which served as a motivation

for our investigation. In addition to the project goal, the project manager aimed to verify the suitability of the remote team for future globally distributed projects.

Process Distribution: The entire project was divided into a set of subsequential phases. Most of the activities were performed on site by the central unit. Development tasks were distributed between the on-site and off-site (central and remote) teams. However, the tasks were allocated according to the independent software architecture modules and primarily did not request close dependency between the distributed programmers. Table 1 shows the lifecycle process distribution between the teams:

- Customer unit consisting of 2 people (overall project manager and systems analyst)
- Central supplier unit consisting of 6 people (project manager, 2 system analysts, 2 testers and a programmer)
- Remote supplier unit consisting of 5 people (a team leader and 4 programmers).

The project manager and the remote team leader have worked in the studied company for 3 years. System analysts and testers had more than 10 years of experience in the same company. Programmers' experience varied between 2-5 years.

Table 1. Lifecycle process distribution

Lifecycle activities	Customer	Supplier: Central unit	Supplier: Remote unit
Requirements Analysis	X	X	
High Level Design		X	
Detail Architecture		X	
Development		X	X
Unit Testing		X	X
System Testing		X	
Acceptance Testing	X	X	

Thus, software processes in this project were distributed between the members of an intra-organizational and intra-national team. According to Prikladnicki et al. in [17] the project can also be characterized as an internal domestic supply.

Task Complexity: The developed software product was perceived as a complex system from both an architectural and technological perspectives. This was particularly discussed when the project was summed up in the end during the post mortem meeting. Another reason for work complexity was lack of experience of the developers with the new technologies the project was built on and unfamiliarity with the business domain. This required close cooperation and joint problem solving.

Process Quality Assurance: Since the supplier organization implemented a quality system and certified its processes according to the ISO 9001:2000 standard, the project followed standardized guidelines for requirement specification, task management, progress reporting and monitoring, and other activities. The project manager established a specifically tailored quality plan describing the procedures to follow. Weekly teleconferences were organized to discuss urgent problems and plans. A special tool was used for allocating tasks and monitoring progress. This tool was also used to monitor the workload of each developer.

3.2 Data Sources and Analysis

In this case study we have used multiple data sources: individual and group interviews, post mortem analysis, risk survey and participant observations; all collected by the first author, to identify problems related to distribution and the effect of different coordination styles on team performance.

We conducted four individual qualitative interviews, which were from 30 to 60 minutes long in the central location and one a 90 minutes group interview with the remote development unit in their location. We also organized a post mortem analysis meeting [16] at the end of the project. The meeting involved all project members in the discussion of what went well and what did not work in the project, concluded with a root-cause analysis of the major issues. In addition, a risk checklist was distributed to the central project manager and the remote team leader in order to identify problems related to collaboration. The first author of this paper was also involved in the quality assurance and process improvement activities during the last 6 months of the project. This enabled the possibility to use participant observation [16] when observing both teams. In this study we are mainly relying on qualitative interviews [15] and participant observation [16].

Multiple sources of evidence enabled triangulation when analysing the material. This helped us increased validity and allowed to create a broad view of the project, also collecting the “unsaid issues”; the remote team reported many problems and was open to the researcher during the group interview performed at their premises, however, they did not to report any problems related to collaboration, communication or coordination during the post-mortem meeting.

The qualitative analysis of the data was performed in several steps. First we analyzed the problems reported and observed from both teams separately. This allowed us to see the diversity of evidence dependent on the chosen investigation method and the team perspective. Next we analyzed the interrelation between the problems, geographic distribution and related them to the three coordination mechanisms. At the end, we tracked the influence of different coordination mechanisms on the team behaviour.

We believe that the selected case is internally representative, since the studied project involved all of the employees in the remote location. The internal validity could be improved by observing several collaborative projects, but this was not possible to address, since the selected project was the only ongoing project between the central and the remote locations at the time of investigation. We also believe that such factors as poor working environment that may have affected the remote team motivation shall be seen as a part of the supply chain management and coordination.. Our study is limited because it is only validated in the context of a Northern European intra-organizational intra-national collaboration and therefore has potential threats to external validity. Hence, the findings cannot be generalized widely to practitioners from other countries. To improve the potential for external validity of our single-case study, we applied existing theory as recommended by Yin (2003) and emphasize our contribution as exploratory-illustrative rather than theory-building. Reliability of the study can be judged upon the description of the methodology used and the context of the intra-organizational intra-national collaboration.

4 Coordination of Work in a Distributed Project

We now present how the different coordinating mechanisms were used in the case study and discuss the reasons of coordination pitfalls.

4.1 Standardization

Face-to-face contact is the richest communication channel we have, and any electronic channel is significantly poorer [18]. Hence, if coordination is based on coordination by programme (standardization) in a distributed team, where coordination is affected through instructions and plans generated beforehand, this will reduce the level of face-to-face communication, which again tends to hinder effective communication and the possibility for coordination by mutual adjustment.

Case-study: Standardization was selected as a prior ground for project coordination, following processes certified according to the ISO 9001:2000 standard. The project manager established guidelines (project quality assurance plans, communication plans, different software development related process handling guidelines) for the team and expected everybody to follow them.

As the project went on, the project manager realized that even though the work processes were standardized, there were disparities in work practices between the central and remote units. Not working according to the established processes as expected, resulted in a lack of understanding of the context of decision-making, and decreased the level of trust between the distributed team members. These disparities considerably decreased the predictability of the team member performance and mutual cohesion.

The project manager reported that the remote team performance was unexpectedly low. Although the collocated team members in the remote location were cohesive, their informality caused problems for team coordination: the systems analysts from the central location reported that the remote team unit acted “as a joint body”, independently shifting their work tasks and interpreting unclear requirements. The remote team was able to act independently, which was good, but making operational decisions without sufficient information often caused subsequent rework. Coordination by standardization didn’t encourage frequent feedback from the remote location, therefore, it caused delays and absence of a joint problem resolution, resulting in the remote team making operational decisions on insufficient grounds. In addition, due to the remote team’s independent behaviour, the project manager felt he had a limited visibility of what was done by whom and when, which he felt caused an inability to effectively coordinate and plan the remote team’s work load. This illustrates that despite the official project guidelines, according to the project manager the remote team acted differently than expected.

Often lacking awareness of the distant activities, the project manager perceived the remote team to be less productive than expected. The amount of defects uncovered during testing was perceived as too high and decreased his cognition even more. However, there could be another explanation for this. Despite the standard technological infrastructure, the remote team suffered from old-fashioned computers and slow communication lines. This required several hours for remote code compilation per

day. Thereby, unjustified application of new unstable architectural solutions caused additional rework. These problems were reported to the project manager only at the end of the project during the post-mortem analysis meeting. However, uncertainty of the remote activities and not satisfied with the quality delivered, resulted in the desire of the project manager to increase the use of direct supervision for team coordination during the project.

Discussion: Relying on coordination by standardization has led to a list of problems in relation to a) disparities in work practices, b) little feedback, and c) lack of transparency of the remote activities.

The project manager chose standardization as the primary coordinating mechanism independent of the tasks to be solved, and their complexity. While the remote development team struggled with technological issues because of an unknown development platform in addition to an increased number of changes, lack of mutual adjustment reduced the possibility for solving these issues effectively and a joint decision-making process between the distributed teams. Change-driven agile practices that were recognized as key for the remote team cohesion went into conflict with the plan-driven standardized work practices of the central unit. These disparities caused misunderstandings, e.g. project guidelines prescribed each developer to report their effort individually, however, the reported progress could be misleading, since there was a high level of workload shift between the remote unit members.

4.2 Direct Supervision

According to Takeuchi and Nonaka [21] management should establish enough checkpoints to prevent instability, ambiguity, and tension from turning into chaos. At the same time, managers should avoid the kind of rigid control that impairs creativity and spontaneity [21]. Therefore direct supervision must be used with care.

Case-Study: Because the project manager lacked previous experience with remote team coordination and coordination by standardization was problematic because of the limited feedback and disparities in work practices, he then started relying more on direct supervision as the most important coordination strategy.

The remote unit communicated directly with the systems analysts and testers whenever necessary. Some of the tasks and most of the problem reported were also coordinated directly without the project manager's and remote team leader's involvement. This produced a situation where the local project manager had a feeling that he was not in charge of all the activities, which together with the geographic distribution and lacking transparency caused him a "headache" because he felt he lacked control of the project. The project manager also reported that it was difficult to spread awareness of everyday activities and more importantly rapid changes across the distance. Feeling he had no control over the remote team, he increased monitoring and started requiring daily progress reports. At the end of the project, he even suggested to install a video camera on the remote site.

Feeling not trusted and afraid of collaboration determination, the remote team started to report even fewer problems. This again resulted in the project manager perceived the remote team as inactive and lacking initiative in the weekly teleconferences, often

remaining silent. This again limited his ability to supervise the remote team and decrease the cognition even more.

Discussion: Geographic distribution made remote team coordination a complex task and led the project to a chain of problems in relation to a) lacking proximity and transparency, b) fear of losing control, c) lack of trust and d) decreased feedback.

Our case illustrates a closed loop that starts with a lack of trust and belief of the project manager in the remote team's ability to perform, leading to increased monitoring and supervision, which eventually leads to a lowered morale of the remote team, unwillingness to collaborate and little feedback. The remote team's silence and delays given the lowered levels of trust are misinterpreted by the project manager and trust decreases even more. Again, this increases the project manager's desire to monitor. It was a deadlocked situation.

When a manager in whom the employee has little trust gives negative feedback, it is likely that the employee will doubt the accuracy of the feedback [23, 24]. This hinders the team leader from managing the team effectively.

4.3 Mutual Adjustment

Mutual adjustment in its pure form requires everyone to communicate with everyone [18]. Therefore to employ mutual adjustment as the prime coordinating mechanisms the team or network need to be dense and co-located, and since our communication abilities are limited, that means they also have to be small [18]. Usually there is a limited possibility for face-to-face communication in a GSD project.

Case-Study: Because the project managers first relied on standardization, mutual adjustment was not seen as an option for coordination with the remote team. When standardization did not work as expected, he started relying on direct supervision was increased, but because of problems with trust and giving feedback, the level of mutual adjustment was reduced even more. Achieving effective mutual adjustment was also difficult because of the geographic distribution, no travelling to the remote office and only relying on instant messaging tools and phone as primary communication means.

As a consequence of heavy monitoring, both sides reported the decreased level of trust; which again reduced the possibly for mutual adjustment even more. Though the remote unit had previous work experience with the system analysts from the central location.

Geographic distribution and lack of transparency caused psychological discomfort for the project manager. He claimed: *"I am not sure if they are working over there"*. The project manager recognized he lacked a belief in the remote team's ability to perform and this was also felt by the remote team lead and reported through the risk survey. A number of contributions in other research disciplines has reported on numerous occasions that studied and analyzed this type of organizational behaviour e.g. [3, 25]. Trust was also affected by the lack of face-to-face meetings, poor socialization, too little communication, misunderstood silence and unwillingness to discuss collaborative problems. In the project this was a significant impediment to apply mutual adjustment for team coordination.

The remote team claimed that there was an increased amount of seemingly unimportant questions that were never communicated through distance, however thei

would have been resolved if the systems analysts travelled on a regular basis to the remote location.

The diversity of social situation in the capital city and the small town increased the gap between the two distributed teams. The benefits that were organized for the central location were not offered for the remote location. Their office got low technological infrastructure, old computers and communication lines. The remote-team leader complained: *“It took some time to even organize the supply of drinking water. It was not easy to convince the management to order the service and it wasn’t easy to find the service supplier in our region”*. After feeling the lack of trust and belief in their performance from the central location, they were afraid to complain about their problems. This subsequently decreased the team’s psychological comfort and ability to rely on mutual adjustment.

Discussion: Several of the problems described are examples of impediments to establishing mutual adjustment in geographically distributed projects. The case shows that geographic distribution and missing trust affected the project manager’s and the team’s behaviour. While the project manager didn’t trust and believe in success of collaboration, the team tried to hide their problems. Given development tasks that were perceived as complex, in addition to frequent changes, stressed by the deadlines and the usage of unfamiliar technologies, the remote team relied on mutual adjustment to coordinate and leverage their work within the team. Because face-to-face meetings were costly, and the project was distributed to save costs, neither systems analysts nor the project manager ever travelled and, therefore, lack of proximity of the remote team’s activity caused coordination breakdown.

5 Discussion

5.1 Pitfalls in Remote Team Coordination

In this paper we try to understand, what is the effect of geographic distribution on how the work is coordinated in a distributed environment. The area of distributed software work is relatively new, and the majority of research conducted in this area is exploratory in nature. Contributing to related research [26, 27, 2] our case study serves as an example, where distributed software teams rely on formal mechanisms (standardization), such as detailed architectural design and plans, to address impediments to team communication that result from geographical separation. However, in contradiction to a common view that the most effective way to manage global software teams is reliance on methodological standardization [2], the results of our case study indicate that coordination by program didn’t work as intended, due to problems with disparities in work practices and limited feedback from the remote location.

Trying to improve the situation, direct supervision became the next dominating coordination mechanism. However, lack of proximity and trust, troubled the ability for direct supervision. Trying to control the remote team, the project manager from the central unit stumbled upon the problem of lacking trust and subsequently missing feedback again.

Our case also proves that coordination by mutual adjustment is troublesome because of the geographic distribution. Lacking trust, limited opportunities for face-to-face

contact and constant feedback in the distributed environment make it difficult to use mutual adjustment when coordinating work, which again hinders effective communication. This is one of the reasons for distributed organizations having problems to display the same cohesiveness, resilience, and endurance as a “physical” organization, and it is likely that a distributed organization will therefore experience a handicap that must be outweighed by other factors [18]. Frequent communication is important in distributed teams for providing constant confirmation that team members are still there and still working [28]. If feedback is provided on a regular basis, communication improves, which in turn leads to greater trust and improve team performance [29, 30]. Lacking mutual adjustment results in choosing either direct supervision or standardization as the dominating coordinating mechanisms.

5.2 The Effect of Coordination Mechanisms on Remote Team Performance

Exploring the characteristics of the effect of different coordination mechanisms on distributed team performance, we have gathered the observations that indicate that lacking balance in selecting coordination mechanisms not only leads to coordination pitfalls, but also affects the remote team performance.

Relying on standardization as the primary coordinating mechanisms one should be sure that there are no disparities in work practices, since it is difficult to discover such disparities using standardization. Moreover, disparities in work practices can lead to troubled understanding of the manager’s decisions, misunderstanding of requirements and misbehaviour during implementation and testing [5].

The effect of direct supervision, can have devastating effects (such as decreased team performance or a total project breakdown) when people do not trust each other. Trust is a premise for getting necessary feedback, which is needed for direct supervision to be successful [5].

By not fulfilling the most basic needs [25], e.g. existence and relatedness, or the most basic hygiene factors [31], e.g. working conditions, it is not likely that the employees will enjoy their work and therefore perform accordingly (a paycheck is not enough [32]). Additionally, a manager must always adapt the leadership style no matter what type of coordinating mechanisms might come into play. Employing a task-oriented and authoritarian style [3], when it is not appropriate, will seriously influence work throughput in most teams.

In the case study described in this paper the manager would most likely have benefited by applying situational leadership [33],[34]. The manager, who tried to empower and delegate responsibility to the team [35], ended up with a role where he was defining and instruct the team what to do, and then making it impossible to empower the team Furthermore, the attempts of the remote team to cope with the complexity of the given tasks by applying mutual adjustment on their site were not appreciated.

5.3 Implications for Practitioners

With our case study we exemplify the behaviour of the project manager who didn’t trust the remote team and the effect of his coordination style on the remote team performance and psychological comfort. We therefore recommend:

Adjust Your Coordination Mechanisms: Consider the nature and perceived complexity of the tasks (both technical and business related) given to the distributed team and adjust the coordinating mechanisms taking this into account. Since the remote team may have problems understanding the tasks it is important to use a mechanism that encourage frequent communication and feedback.

Furthermore, estimate the level of organizational diversity and disparities in work practices before relying on standardization. In addition, managers should avoid the kind of rigid control that impairs creativity and spontaneity [21]. Therefore direct supervision must be used with care. A premise for being able to adjust the coordinating mechanism is frequent feedback and trust. This can be achieved by using the measures suggested by Moe and Smite [5].

Make it Possible to Apply Mutual Adjustment When Needed: A software organization often deploys experts in multi-disciplinary teams that carry out projects in a complex and dynamic environment. Such organizations can be classified as innovative, where mutual adjustment is the most important coordinating mechanism [6]. The managers should avoid rigid control (direct supervision), which impairs creativity and spontaneity [21]. Mutual adjustment is also important when solving complex tasks. Therefore there is a need for mutual adjustment in any software development project. Relying too much on standardization and direct supervision, together with having problems achieving trust will make it difficult to apply mutual adjustment.

Avoid Complex Task Distribution: Carmel claims that the cost of coordinating work increases when either the tasks are new or uncertain, or when the work units become more interdependent [2]. Distribution of interdependent and tasks perceived as complex exacerbates the problems related even more.

In addition, making the remote team responsible for entire modules is important, i.e. from planning to testing, the team gets a deeper understanding of the tasks it is working on. Training and investment in a long time relationship is also important. Extending the existing studies on globally distributed software development (GSD) that recommend to ‘trim down GSD towards nearshoring’ [8], we emphasize that distributing software development within national and organizational borders still, in some ways, faced the same difficulties as those faced in the GSD context. Therefore, we recommend evaluating all pros and cons before starting a distributed project and avoiding pure cost reduction deals.

Keep the Team Small: Given that mutual adjustment in its pure form requires everyone to communicate with everyone, the team or network needs to be compact [18].

5.4 Implications for Future Research

While this case study illustrates the effect of coordination by standardization and direct supervision, our future work will focus on exploring the effect of coordination by mutual adjustment. Holmström et al. state [4]: despite the fact that the more common view is that agile methods are not applicable for GSD, agile practices may assist in alleviating some of the distribution problems. We would therefore be interested in exploring distributed software teams that mainly rely on mutual adjustment while performing tasks of different complexity. Future research should also study how work

is coordinated when there exist a more mature relationship between the remote and local team.

Along with the research in the area of global software development, we also emphasize the importance of empirical research in the intra-organizational intra-national context to supplement the understanding of internal domestic supply chains.

6 Conclusions

While organizations become more distributed, software development tasks of different complexity are often performed by distributed software teams. Team coordination necessary to make sure that it contributes to the overall objective faces new challenges with respect to geographic distribution of project managers and their teams. The overhead of control and coordination associated with any software project is astounding [2]. It is therefore important to understand the pitfalls of team coordination in order to make a distributed project successful.

We have found that:

- Failure applying standardization may result in direct supervision becoming the dominating coordination mechanism, but this only increases the problems related to decreased communication and missing feedback.
- Trust and a common understanding of the work processes is a premise for succeeding with standardization, direct supervision and mutual adjustment.
- Application of frequent communication and feedback through mutual adjustment is essential in overcoming the complex tasks.

Although theory suggests that different coordination mechanisms shall dominate in different situations dependent on task complexities [6], geographic distribution introduces certain impediments in applying each of the mechanisms discussed in this paper. In particular, disparities in work practices put under threat the path of standardization, lack of proximity troubles direct supervision and distribution makes it difficult to apply mutual adjustment, which is important for complex tasks. Thus, coordinating mechanisms shall be chosen thoroughly and in balance.

Acknowledgments. This research is supported by the Research Council of Norway under Grant 181658/I30, the Knowledge Foundation in Sweden under a research grant for the project *BESQ*, European Social Fund under grant “Doctoral student research and post doctoral research support for university of Latvia” and the Latvian Council of Science within project Nr. 02.2002.

References

1. Herbsleb, J.D., Mockus, A.: An Empirical Study of Speed and Communication in Globally Distributed Software Development. *IEEE Transactions on Software Engineering* 29(3), 481–494 (2003)
2. Carmel, E.: *Global software teams: collaborating across borders and time zones*. Prentice-Hall, Englewood Cliffs (1999)

3. Tannenbaum, R., Schmidt, W.H.: How to Choose a Leadership Pattern. *Harvard Business Review* 51, 162–174 (1973)
4. Holmström, H., Fitzgerald, B., Ågerfalk, P.J., Conchúir, E.Ó.: Agile Practices Reduce Distance in Global Software Development. *Information Systems Management* 23(3), 7–18 (2006)
5. Moe, N.B., Šmite, D.: Understanding a Lack of Trust in Global Software Teams: A Multiple-Case Study. In: *Software Process Improvement and Practice*. John Wiley & Sons (in press, 2008)
6. Mintzberg, H.: *Mintzberg on Management: Inside Our Strange World of Organizations*. Free Press, New York (1989)
7. Cataldo, M., Bass, M., Herbsleb, J.D., Bass, L.: On Coordination Mechanisms in Global Software Development. In: *Proceedings of the International Conference on Global Software Engineering (ICGSE 2007)*, pp. 71–80. IEEE Computer Society Press, Munich, Germany (2007)
8. Holmström, H., Conchúir, E.Ó., Ågerfalk, P.J., Fitzgerald, B.: Global Software Development Challenges: A Case Study on Temporal, Geographical and Socio-Cultural Distance. In: *Proceedings of the International Conference on Global Software Engineering*, pp. 3–11. IEEE Computer Society Press, Costão do Santinho, Florianópolis, Brazil (2006)
9. Crowston, K., Annabi, H., Howison, J., Masango, C.: Effective Work Practices for Software Engineering: Free/libre Open Source Software Development. In: *Proceedings of the 2004 ACM Workshop on Interdisciplinary Software Engineering Research*, pp. 18–26. ACM Press, Newport Beach (2004)
10. Herbsleb, J.D., Paulish, D.J., Bass, M.: Global Software Development at Siemens: Experience from Nine Projects. In: *Proceedings of the 27th International Conference on Software Engineering (ICSE 2005)*, pp. 524–533. ACM Press, St. Louis (2005)
11. Boden, A., Nett, B., Wulf, V.: Coordination Practices in Distributed Software Development of Small Enterprises. In: *Proceedings of the International Conference on Global Software Engineering (ICGSE 2007)*, pp. 235–246. IEEE Computer Society Press, Munich, Germany (2007)
12. Fonseca, S.B., Souza, C.R.B.d., Redmiles, D.F.: Exploring the Relationship between Dependencies and Coordination to Support Global Software Development Projects. In: *Proceedings of the International Conference on Global Software Engineering (ICGSE 2006)*, p. 243. IEEE Computer Society, Costão do Santinho, Florianópolis, Brazil (2006)
13. Wiredu, G.O.: A Framework for the Analysis of Coordination in Global Software Development. In: *Proceedings of the 2006 International Workshop on Global Software Development for the Practitioner at International Conference on Software Engineering*, pp. 38–44. ACM Press, Shanghai, China (2006)
14. Herbsleb, J.D., Mockus, A.: Formulation and Preliminary Test of an Empirical Theory of Coordination in Software Engineering. In: *Proceedings of ACM Symposium on the Foundations of Software Engineering (FSE)*, Helsinki, Finland, pp. 112–121 (2003)
15. Myers, M.D., Newman, M.: The Qualitative Interview in IS Research: Examining the Craft. *Information and Organization* 17(1), 2–26 (2007)
16. Dingsøyr, T.: Postmortem Reviews: Purpose and Approaches in Software Engineering. *Information and Software Technology* 47(5), 293–303 (2005)
17. Prikladnicki, R., Audy, J.L.N., Damian, D., de Oliveira, T.C.: Distributed Software Development: Practices and Challenges in Different Business Strategies of Offshoring and Onshoring. In: *Proceedings of the International Conference on Global Software Engineering (ICGSE 2007)*, pp. 262–274. IEEE Computer Society Press, Munich, Germany (2007)

18. Groth, L.: *Future Organizational Design: The Scope for the IT-based Enterprise*. John Wiley & Sons, New York (1999)
19. Dybå, T.: Improvisation in Small Software Organizations. *IEEE Software* 17(5), 82–87 (2000)
20. Nerur, S., Balijepally, V.: Theoretical reflections on agile development methodologies - The traditional goal of optimization and control is making way for learning and innovation. *Communications of the ACM* 50, 79–83 (2007)
21. Takeuchi, H., Nonaka, I.: The New New Product Development Game. *Harvard Business Review* 64, 137–146 (1986)
22. McGregor, D.: *The Human Side of Enterprise*. McGraw Hill, New York (1960)
23. Dirks, K.T., Ferrin, D.L.: The role of trust in organizational settings. *Organization Science* 12, 450–467 (2001)
24. Salas, E., Sims, D.E., Burke, C.S.: Is there a big five in teamwork? *Small Group Research* 36, 555–599 (2005)
25. Alderfer, C.P.: An Empirical Test of a New Theory of Human Needs. *Organizational Behavior & Human Performance* 4, 142–176 (1969)
26. Ramesh, B., Cao, L., Mohan, K., Xu, P.: Can distributed software development be agile? *Communications of the ACM* 49, 41–46 (2006)
27. Ågerfalk, P.J., Fitzgerald, B.: Flexible and distributed software processes: Old petunias in new bowls? *Communications of the ACM* 49, 26–34 (2006)
28. Jarvenpaa, S.L., Shaw, T.R., Staples, D.S.: Toward contextualized theories of trust: The role of trust in global virtual teams. *Information Systems Research* 15, 250–267 (2004)
29. Jarvenpaa, S.L., Knoll, K., Leidner, D.E.: Is anybody out there? Antecedents of trust in global virtual teams. *Journal of Management Information Systems* 14, 29–64 (1998)
30. Jarvenpaa, S.L., Leidner, D.E.: Communication and trust in global virtual teams. *Organization Science* 10, 791–815 (1999)
31. Tagiuri, R.: Managing people: Ten essential behaviors. *Harvard Business Review* 73, 10–10 (1995)
32. Whyte, W.F.: *Money and Motivation*. Harper & Row, New York (1955)
33. Hersey, P., Blanchard, K.H.: Life Cycle Theory of Leadership. *Training and Development Journal* 33, 94–94 (1979)
34. Hersey, P., Blanchard, K.H.: Great ideas revisited: Revisiting the life-cycle theory of leadership. *Training & Development* 50, 42–48 (1996)
35. Hersey, P., Blanchard, K.H., Johnson, D.E.: *Management of Organizational Behavior: Leading Human Resources*. Prentice Hall, New Jersey (2001)

A Model to Identify Refactoring Effort during Maintenance by Mining Source Code Repositories

Raimund Moser¹, Witold Pedrycz², Alberto Sillitti¹, and Giancarlo Succi¹

¹ Center for Applied Software Engineering, Free University of Bolzano-Bozen, Italy

² Department of Electrical and Computer Engineering, University of Alberta, Canada
{raimund.moser, alberto.sillitti, giancarlo.succi}@unibz.it,
pedrycz@ee.ualberta.ca

Abstract. The use of refactoring as a way to continuously improve the design and quality of software and prevent its aging is mostly limited to Agile Methodologies and to a lower amount to software reengineering. In these communities refactoring is supposed to improve in the long-term the structure of existing code in order to make it easier to modify and maintain. To sustain such claims and analyze the impact of refactoring on maintenance we need to know how much refactoring developers do. In few cases such information is directly available for example from CVS log messages. In this study we propose a model on how to mine software repositories in order to obtain information of refactoring effort throughout the evolution of a software system. Moreover, we have developed a prototype that implements our model and validate our approach with two small case studies.

Keywords: refactoring, software metrics, software evolution, Agile Methodologies.

1 Introduction

Refactoring is defined as a behavior-preserving source-code transformation [3] that increases the quality of the code and reduces its complexity. Agile developers claim that refactoring has a positive impact on the evolution of the overall design of a software system [2]: frequently refactored code is easier to understand, to maintain, to modify, and to adjust to new requirements. However, nowadays such claims are not yet validated sufficiently by quantitative data coming from real projects in industry. Data collection in software industry is hard: developers and managers do not want to waste time for “non productive” activities and additional resources dedicated to manual data collection are very costly. Therefore, only automatic and non-invasive data collection processes and data analysis [18] [10] can leverage metric programs in the software industry. With this work we follow this approach: we aim at identifying in an automatic way refactoring effort during maintenance of a software system in order to enable automatic monitoring and analysis of the impact of refactoring for example on software maintainability and software maintenance costs.

Most of the research done in refactoring can be classified into three different categories [13]:

1. Refactoring and the code structure: how does refactoring change the internal structure of the software? Such approaches use different techniques such as abstract syntax tree representation of source code to analyze which kind of graph transformations are achieved by refactoring and how different software metrics (coupling, cohesion, complexity, etc.) are changed by it [3].
2. Refactoring and code smells: other researchers focus on the identification of code smells by mining change history, software metrics and by using source code visualization in order to identify which refactoring to apply for which code smell and more in general when to apply refactoring during development [17] [7] [19]. Bois et al. [4] for example analyze how refactoring manipulates coupling and cohesion metrics and how to identify refactoring opportunities that improve these metrics.
3. Refactoring and reverse engineering: in reverse engineering people analyze how refactoring can be used to reconstruct the design flow of a software system and how to support reverse engineering [6]. In this context refactoring is often used to convert legacy code in to a more structured form [15].

While some of the issues regarding the relationship between refactoring and the design and quality of source code are addressed by the research listed above many questions remain still open [12]. One reason for the limited empirical validation of the claimed benefits of refactoring throughout the evolution of a software system is clearly the lack of quantitative data: industrial studies where data about refactoring is collected within the scope of a metrics program are rare and open source projects usually do not provide such information. However, the availability of such data is a precondition for assessing the benefits of refactoring for maintenance costs and the long-term evolution of software design, code quality, and testability [4] [20].

The idea of this research is to propose a model for mining automatically source code repositories of a software system in order to get quantitative data on refactoring effort throughout its evolution. We follow in part the approach proposed by Demeyer et al. [6]: we use changes of source code metrics as basis for identifying refactorings. Whereas Demeyer et al. focus on the evolution of the design of a software system in order to support reverse engineering we extend the work of Demeyer et al. in several ways:

- We focus on how to identify refactoring activities throughout maintenance of a software system rather than detecting single refactorings and their influence on software design.
- We develop a metric independent model that takes into account a larger number of refactorings than the method used by Demeyer et al.
- We introduce the notion of “refactoring activity”, which is a kind of probability for the occurrence of refactorings during software development.

The main contributions of this paper are twofold: first, we propose a general model for detecting refactoring activities during maintenance of a software system by mining

its source code repository. And second, we instantiate such model and perform a case study on one industrial and one open source project for a first validation.

The paper is organized as follows. In Section 2, we define our model for identifying “refactoring activity”. In Section 3, we discuss a prototype implementation of the model, and Section 4 describes two small case studies. In Section 5, we discuss threats to validity and we briefly outline some possible directions for future work. Finally, in Section 6, we draw some conclusions of this research.

2 A Model for Computing “Refactoring Activity”

The basic idea we use in this research is simple: a refactoring may be detected by analyzing the changes of several source code metrics, which are affected by it. Let us make a simple example: the “Extract Method” refactoring [8] extracts a part of a long and complex method and puts it into its own method. Thus, such refactoring will for sure increase the number of methods in a class by one and probably also lower the lines of code of its longest method. If we find such change pattern in a software repository we may conclude that it originates from an “Extract Method” refactoring. However, things are not that easy: many other activities (bug fixes, adding functionality, etc.) could lead to the same change pattern. In general it is not possible to associate in a one-to-one relation software refactorings with change patterns of source code metrics. At most we can group “refactoring change patterns” – changes of source code metrics induced by a refactoring R_i - into two categories:

1. Necessary changes: Those that are induced by refactoring R_i always in the same (deterministic) way. The “Extract Method” refactoring for example will always increase the number of methods in a class by one. Thus, such changes are always observed after a refactoring of this type has been applied.
2. Likely changes: these changes may or may not be visible as a consequence of a refactoring R_i . An “Extract Method” refactoring for example will most often be applied to the largest method of a class. The likelihood to observe such changes depends on the specific refactoring and several “soft” factors (developers, coding standards, etc.).

Taking into consideration the observations above we propose a model that is based on the following two assumptions:

- First, a refactoring changes some code metrics in a determined way while others can be changed with a likelihood that depends on the specific type of refactoring, metric, and several “soft” factors.
- Second, “refactoring change patterns” in general are induced with a higher probability by refactoring than by other coding activities.

More formally we can define a model for detecting “refactoring activity” in the following way:

Let $R = \{R_i, R_i \text{ is a refactoring}\}$ be a set of refactorings. Furthermore, we denote the set of source code metrics we consider by $M = \{M_i, \text{ where } M_i \text{ is source code metric}\}$. We assume that we can decompose our software system at any specific point in

time t into a set of entities $E_t = \{e(i)t, e(i)t \text{ is source code entity } i \text{ at time } t\}$. Then we compute the change metrics for one entity e at time t in the following way:

$\Delta M_i(et) = M_i(et) - M_i(et-\Delta t)$, where Δt is a time interval we choose for computing the changes of metric M_i .

By ΔM_t we denote the set of all change metrics at time t , i.e., the changes of all metrics M_i from $t - \Delta t$ to t . Next we define a probability function that gives the probability that a set of change metrics ΔM_t originates from a refactoring R_i :

$$p_t : R \times \Delta M_t \rightarrow [0,1] \text{ with } p_t(R_i, \Delta M_1(e_t), \Delta M_2(e_t), \dots) \in [0,1] \tag{1}$$

If p_t evaluates to 0 we can exclude for sure that refactoring R_i has been applied on e while a resulting value of 1 means that e has been definitely refactored (a refactoring R_i has been applied) during the time interval $[t-\Delta, t]$.

We sum up all entities of a software system and all considered refactorings and get a general formula for a measure of “refactoring activity” in the time interval $[t-\Delta t, t]$:

$$RA_t = \frac{\sum_{e \in E} \sum_{r \in R} p_t(r, \Delta M_t(e))}{|ER|} \times 100\%, \text{ where } ER = \{e \in E, p_t(e) \neq 0\} \tag{2}$$

Formula (2) is not normalized and therefore not a probability in a strict sense. It says the following: if some entities have been refactored and we are able to identify such refactorings with a probability of 1 we get a refactoring activity RA_t of 100%. Values less than 100% indicate a smaller probability for refactoring while values higher than 100% mean that more than one refactoring has been applied to the same entity. We divide formula (2) by the number of entities that show a refactoring probability greater than zero in order to prevent that a very small refactoring probability for many entities of a software system does sum up to a big overall refactoring activity. In this way if for example we find in a software system 100 entities, which show a change pattern that gives a very small refactoring probability of 0.01 for each entity, then the overall refactoring activity is 1% and not 100% as it would be if we omit the denominator in formula (2).

3 An Implementation of the Model

The model described in Section 2 depends on 5 parameters we have to choose for a real implementation:

1. A set of refactorings: we consider 20 refactorings, at least two for each category, from Fowler’s book [8].
2. A set of source code metrics: we choose the Chidamber and Kemerer set of object-oriented metrics [5], McCabes’s cyclomatic complexity [11] of a method, non-commented lines of code of a method and some other size or complexity-related measures (see Table 1).
3. A time interval for calculating the changes of source code metrics: we use a constant time interval of one day.

4. A suitable probability function for modeling the relationship between a refactoring and specific change patterns.
5. A decomposition of a software system into distinct entities: we consider Java classes as entities.

For the implementation of a first prototype we have chosen a set of parameters, which turns out to be reasonable for the two case studies we present in the next section. For the decomposition aspect our approach is simple and intuitive: we consider single classes of an object-oriented software system as basic entities. All metrics calculations are based on classes and for the moment we do not consider for example the method or package level. The prototype is easily extendible and adjustable to new metrics, refactorings, and probability functions. At the moment it implements about 20 refactorings described in Fowler's book [8]. Fowler divides refactorings into 6 different groups: "Composing Methods", "Moving Features Between Objects", "Organizing Data", "Simplifying Conditional Expressions", "Making Method Calls Simpler", and "Dealing With Generalization". For each of these groups we consider at least the two refactorings, which are most common among developers [6] [3].

For collecting source code metrics we use the PROM tool [18]. It enables in an easy and non-invasive way the daily extraction of several object-oriented and procedural source code metrics from a source code repository. For the case study to be presented in the next Section we collect the metrics listed in Table 1. These are also implemented in our tool.

Table 1. Selected metrics for computing change patterns

Metric name	Definition
CK metrics	Chidamber and Kemerer set of object-oriented design metrics
LOC	Number of Java statements per class
NOM	Number of methods declared in a class
NOA	Number of attributes declared in a class
LOC_PER_METHOD	Average number of Java statements per method in a class
MAX_LOC_PER_METHOD	Maximum number of Java statements per method of all methods in a class
PARAM_PER_METHOD	Average number of parameters per method in a class
MAX_PARAM_PER_METHOD	Maximum number of parameters per method of all methods in a class
MCC	Average McCabe's cyclomatic complexity per method in a class
MAX_MCC	Maximum McCabe's complexity per method of all methods in a class
NUMBER_OF_CLASSES	Number of classes

We do not claim that the metrics in Table 1 are the most suitable ones for detecting refactorings. We employ them in this first study because many industrial and open source metric tools provide them; moreover, other researchers use a similar set of metrics [6] [20]. Clearly, in the future we plan to experiment with a different set of

metrics in order to find out which ones may be the most powerful refactoring detectors. Moreover, at the moment we compute only daily changes of the source code metrics. We think that our model could be improved significantly if instead we compute changes between single CVS log groups [16]. At the moment we are implementing this feature and in the future we plan to do a larger experiment using CVS log groups as time windows for change patterns.

For the case studies we implemented for each refactoring a very simple probability function, which we illustrate with the “Extract Method” (EM) refactoring:

$$p_i(EM, \Delta M_i(class)) = \begin{cases} p \leftarrow \text{necessary condition} = \text{true} \wedge \text{likely condition} = \text{true} \\ q \leftarrow \text{necessary condition} = \text{true} \wedge \text{likely condition} = \text{false} \\ 0 \leftarrow \text{otherwise} \end{cases} \quad (3)$$

A necessary condition is for example: $\Delta M=1$ and $\Delta LOC_PER_METHOD < 0$ and $\Delta RFC > 0$ and $\Delta CBO=0$ and inheritance hierarchy is not changed. A likely condition could be: $\Delta MAX_LOC_PER_METHOD < 0$.

We have derived the necessary and likely conditions for the change patterns in (3) by carefully analyzing the description and examples of the refactorings given in Fowler’s book [8]. However, we can for sure not provide a complete and exact set of conditions, but rather some heuristic rules that may be appropriate only under certain circumstances. The weighting factors p and q allow us to adjust the importance of a single rule and may vary for different environments. For the case studies we tried different values for p and q and found that if we set for all refactorings $p=0.9$ and $q=0.1$ we obtain fairly good results. This is only a first, very simple approach for a probability function and for the future we plan to investigate more in depth, which kind of probability functions would improve our results. However, as we see in the next Section such simple probability function already provides satisfactory results.

4 Two Case Studies

We have evaluated our model on two development projects: an agile, close-to industrial software project developed at VTT, Finland, and an open source software project, PMD (<http://pmd.sourceforge.net>).

In the close-to industrial software development project developers have documented refactoring activities using “user stories”. This facilitates the evaluation, as we do not have to browse the source code to check whether our method is successful or not. In the following we provide a brief description of the characteristics of the first case study used for evaluation.

The object under study is a commercial software project at VTT in Oulu, Finland. The programming language in use was Java and the final product consists of 33 Java classes. The development process followed a tailored version of the Extreme Programming practices [1]: two pairs of programmers (four people) have worked for a total of eight weeks (1- and 2-week iterations). Throughout the project mentoring was provided on XP and other programming issues according to the XP approach. Three of the four developers had an education equivalent to a BSc and limited industrial experience. The fourth developer was an experienced industrial software engineer. The team worked in a collocated environment. From the project plan we know that 3 user stories have been developed in order to refactor part of the system. A first

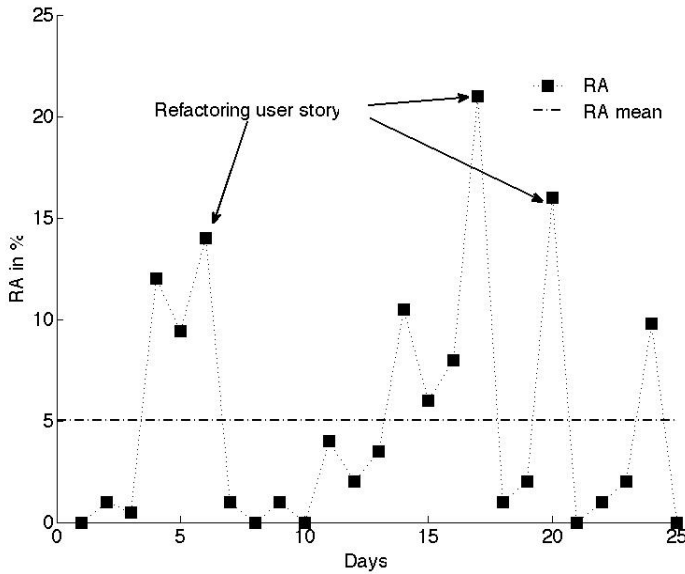


Fig. 1. Temporal evolution of the refactoring activity. *Days* means the number of development days excluding days used for planning and other activities. The *Refactoring Activity RA* is given in percentage as defined in (2).

validation of our proposed method is to check whether we are able to detect these 3 user stories by using our model.

Figure 1 shows the resulting refactoring activity per day and additionally the days when the 3 user stories on refactoring have been implemented: each data point represents the outcome of our model, i.e., the refactoring activity as defined in (2), for one development day. Our model is able to predict fairly well the days when a “refactoring user story” was implemented, as the refactoring activity of those days is located above one standard deviation of the overall mean. However, a few more days show a similar high refactoring activity and in absolute numbers the refactoring activity is not very high (in between 13% and 20%). We could obtain higher numbers by choosing different model parameters. However, in order to be able to compare the two case studies we use the same set of parameters for both; our results allow us to discriminate fairly well refactoring from other development activities, for which it is sufficient to look at the relative differences of daily refactoring activities. To conclude we can state that our model predicts those days when developers did larger refactorings and that for the project under scrutiny the overall refactoring activity is rather low. This could be explained by the fact that developers were exposed for the first time to an Agile process and did not have prior experience with refactoring, which was confirmed by the project manager.

In the PMD project (<http://pmd.sourceforge.net>) developers have documented refactoring activities in the CVS log messages. Again, this facilitates the validation since we do not have to browse manually the source code for identifying potential refactorings. PMD is an open source tool that scans Java source code and looks for potential coding problems. Its development started around summer 2002; the total

number of developers is 13 and it has one lead developer. Unfortunately we cannot obtain further information about the team structure and development methodology from the project website. As we see from the number of unit tests and CVS log messages PMD has been extensively unit tested and refactored during its evolution. We analyze only the main module *pmd*: it has about 770 Java classes and 60000 lines of non-commented source code. Over a period of three years we choose randomly 10 CVS log groups both for refactoring and other maintenance activities. We identify days with commits related to refactoring by their respective CVS log messages, which have to contain the word “refactoring”. The ten non-refactoring related commits deal – as it can be seen from the CSV log messages – with bug fixing or adding new functionality. For each day we compute the changes of source code metrics by extracting them once in the morning and once in the evening from the CVS repository. The results we obtain by running our tool on the 10 refactoring change patterns are reported in Table 2:

Table 2. Refactoring activity for CVS commits related to refactoring

Date	CVS log message	Refactoring activity
2002-07-02	More refactoring	90%
2003-03-04	Some refactoring ...	74%
2003-03-24	Minor refactoring	80%
2003-03-28	Cleaned up the GUI a bit; refactoring ...	90%
2005-06-14	Minor refactoring	80%
2005-08-03	... refactored away one of the symbol table passes	10%
2005-08-14	A bit of refactoring and renaming ...	80%
2005-09-24	More refactoring ...	10%
2005-10-01	A big refactoring ...	90%
2006-02-03	Minor refactoring	70%

For eight out of the ten “refactoring days” we obtain a refactoring activity of more than 70% and for the remaining days we get an activity of 10%. For the “non refactoring days” the computed refactoring activity is in all but one case (for which we find a refactoring activity of 42%) less than 30%. Thus, our model is able to separate fairly well refactoring activities from other maintenance activities; a two-sample Wilcoxon test [9] confirms at a 95% level. In comparison with the industrial case study we find a much higher overall refactoring activity for the open source project. We explain this by the fact that some of the contributors and the lead developer of the PMD project are experienced software engineers and familiar with the practice of refactoring.

5 Threats to Validity and Future Work

As stated several times the validity and power of our model depends on different factors: the set of source code metrics, the time interval used for computation of their changes, the selection of refactorings, and a suitable probability function. The choice

of these factors determines to a high amount the models accuracy and the number of both false positives and false negatives.

This research is at an early stage and we have to investigate more in-depth the impact of the choice of the parameters on the model's accuracy. Moreover, the heuristics we use is subjective and preliminary and we have to control how stable both the number of false negatives and false positives are if we vary them. In particular, we would like to analyze which set of metrics is most suitable as refactoring indicators and how to model a powerful probability function. For such analysis we plan to replicate our experiment with different parameters (metrics, probability functions, refactorings) and with a larger set of CVS transactions both for PMD and other open source and possible industrial projects. We expect to confirm the results obtained from the small samples presented in this study and to be able to gain some insights into how model parameters affect prediction accuracy and change in different development environments and projects.

We are aware of the fact that the individual "refactoring experience" of developers has a big impact on the predictive capability of our model. In order to assess such impact quantitatively and to evaluate whether it is a serious threat to the external validity of our approach, we plan to analyze and compare two similar projects, but which are different in terms of developers' refactoring experience.

Finally, only after such larger experimentation we may conclude **(a)** if we can generalize our approach and **(b)** in general how valid and usable it is for collecting automatically refactoring effort during maintenance of a software system.

6 Conclusions

This research proposes a model for identifying automatically refactoring activities during development or maintenance of a software system by mining its source code repository. We developed a prototype, which implements our model and applied it to one close-to industrial software project and one open source project.

Overall, the results we obtain are promising: in most cases we are able to distinguish fairly well refactoring from other maintenance activities. Such information is valuable as it can be used for identifying:

- The amount of refactoring done during maintenance: to few refactoring could indicate that developers are not aware of/do not have time for refactoring and the code dies the early death of entropy [2].
- The developers, who do most of the refactoring work and those who do not refactor at all. Thus, developers with a lot of refactoring experience can be identified and train other developers on refactoring techniques.
- The parts of a software system that are never or rarely refactored. This could indicate that a) those parts are very well designed and stable and do not need any further refactoring or b) that they are potential future "troublemakers" in terms of bugs and lack of understandability and modifiability.
- The impact of refactoring on the long-term evolution of software design, code quality, and testability [4] [20].

Finally, in a recent study [14] we found some evidence that refactoring has a positive impact on both the quality of the final software product and developers productivity. In such light the amount of refactoring dedicated to a project can be used as additional indicator for the quality of the product. This could be exploited particularly in the context of open source development where it is difficult to estimate final product quality: among other indicators we can extract the refactoring activity from a project's code repository and use it for quality estimation.

In the future we plan to analyze also the use of different kind of refactorings by developers. We think that it would be very valuable – especially for junior programmers – to identify those refactorings that are used most often by experienced software engineers and are probably most effective for improving software maintainability.

References

1. Abrahamsson, P., Hanhineva, A., Hulkko, H., Ihme, T., Jääliñoja, J., Korkala, M., Koskela, J., Kyllönen, P., Salo, O.: *Mobile-D: An Agile Approach for Mobile Application Development*. In: *Proceedings 19th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2004*, Vancouver, British Columbia, Canada (2004)
2. Beck, K.: *Extreme Programming Explained: Embrace Change*. Addison-Wesley, Reading (2000)
3. Bois, B.D., Mens, T.: *Describing the impact of refactoring on internal program quality*. In: *Proceedings of the International Workshop on Evolution of Large-scale Industrial Software Applications (ELISA)*, Amsterdam, The Netherlands (2003)
4. Bois, B.D., Demeyer, S., Verelst, J.: *Refactoring – Improving Coupling and Cohesion of Existing Code*. In: *Belgian Symposium on Software Restructuring*, Gent, Belgium (2005)
5. Chidamber, S., Kemerer, C.F.: *Metrics suite for object-oriented design*. *IEEE Transactions on Software Engineering* 20(6), 476–493 (1994)
6. Demeyer, S., Ducasse, S., Nierstrasz, O.: *Finding Refactorings via Change Metrics*. In: *Proceedings 15th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2000*, Minneapolis, USA (2000)
7. van Emden, E., Moonen, L.: *Java Quality Assurance by Detecting Code Smells*. In: *Proceedings of the 9th Working Conference on Reverse Engineering*. IEEE Computer Society Press, Los Alamitos (2002)
8. Fowler, M.: *Refactoring Improving the Design of Existing Code*. Addison-Wesley, Reading (2000)
9. Hollander, M., Wolfe, D.A.: *Nonparametric statistical inference*, pp. 68–75. John Wiley & Sons, New York (1999)
10. Johnson, P.M., Disney, A.M.: *Investigating Data Quality Problems in the PSP*. In: *Proceedings of 6th International Symposium on the Foundations of Software Engineering (SIGSOFT 1998)* (1998)
11. McCabe, T.: *Complexity Measure*. *IEEE Transactions on Software Engineering* 2(4), 308–320 (1976)
12. Mens, T., Demeyer, S., Bois, B.D., Stenten, H., van Gorp, P.: *Refactoring: Current Research and Future Trends*. *Electronic Notes in Theoretical Computer Science*, vol. 82(3) (2003)
13. Mens, T., Tourwé, T.: *A Survey of Software Refactoring*. *IEEE Transactions on Software Engineering* 30(2), 126–139 (2004)

14. Moser, R., Abrahamsson, P., Pedrycz, W., Sillitti, A., Succi, G.: A case study on the impact of refactoring on quality and productivity in an agile team. In: Proc. of the 2nd IFIP Central and East European Conference on Software Engineering Techniques CEE-SET 2007, Poznan, Poland (2007)
15. Pizka, M.: Straightening spaghetti-code with refactoring? In: Proceedings of the Int. Conf. on Software Engineering Research and Practice - SERP, Las Vegas, NV, pp. 846–852 (2004)
16. Ratzinger, J., Fischer, M., Gall, H.: Improving Evolvability through Refactoring. In: Proceedings 2nd International Workshop on Mining Software Repositories, MSR 2005, Saint Louis, Missouri, USA (2005)
17. Ratzinger, J., Fischer, M., Gall, H.: EvoLens: Lens-View Visualizations of Evolution Data. In: Proceedings of 8th International Workshop on Principles of Software Evolution (IWPSE 2005), Lisbon, Portugal, 5-7 September (2005)
18. Sillitti, A., Janes, A., Succi, G., Vernazza, T.: Collecting, Integrating and Analyzing Software Metrics and Personal Software Process Data. In: Proceedings of the EUROMICRO, Belek-Antalya, Turkey, September 3-5 (2003)
19. Simon, F., Steinbruckner, F., Lewerentz, C.: Metrics based refactoring. In: Proc. European Conf. Software Maintenance and Reengineering, pp. 30–38. IEEE Computer Society Press, Los Alamitos (2001)
20. Stroulia, E., Kapoor, R.V.: Metrics of Refactoring-based Development: An Experience Report. In: The 7th International Conference on Object-Oriented Information Systems, Calgary, AB, Canada, pp. 113–122. Springer, Heidelberg (2001)

The Application of ISO 9001 to Agile Software Development

Tor Stålhane¹ and Geir Kjetil Hanssen^{1,2}

¹The Norwegian University of Science and Technology

²SINTEF ICT

Abstract. In this paper we discuss how to reconcile agile development's focus on speed and lean development with ISO 9001's need for documentation, traceability and control. We see no need to change neither ISO 9001 nor the agile concept. Instead, we see a need to be flexible when using terms such as planning and evidence of conformance. It is true that we can include everything in agile development by making it a requirement but there is a limit to how many documents we can require from an agile process without destroying the very concept of agility.

Keywords: quality assurance, agile software development, ISO 9001.

1 Introduction

With the quick advance of agile methods, some developers feel that ISO 9001 and other quality assurance standards have become irrelevant or not needed any more. The idea seems to be that an ISO 9001 conformant process is incompatible with an agile development process. Our goal is to show that there in reality is more that unite than that separate the two strategies and that both will bring benefits to a project if they are combined.

Many potential customers require that the development company has an ISO certificate before they will award it a contract. This holds both for government agencies and for private companies. The main reason for this is the level of trust created by an ISO 9001 certificate. It is much easier to check that the company has an ISO 9001 certificate than it is to check that they have a good development process and, if they have one, that they really follow it. In addition, there are many companies that are already ISO 9001 certified and want to keep their certificate while at the same time be able to introduce agile development.

2 Related Work

The idea of reconciling agile development and ISO 9001 is probably almost as old as agile development itself. Even though the inventors of agile development did not consider this a problem, quite a lot of managers and quality assurance persons did.

The papers published in this area are many and varied. A problem is that some of the authors do not understand the ISO 9001 or are not aware of the fact that the standard has changed to a process oriented view with the new ISO 9001:2000. A case in point is a paper by Mnkandla and Dwolatzky [1]. Their main argument all too often boils down to statements like “the application of Object-Oriented design principles lead to maintainable systems”.

Another simple but in this case workable solution is suggested by Namioka and Bran [15]. By looking at each time box or increment as a separate project, the problem of making the process ISO 9001 conformant disappears. This solution will, however, create some extra time boxes that are only concerned with developing documentation.

McMichael and Lombardi discuss problems pertaining to aligning ISO 9001 and agile development [2] in a paper from 2007. Their main claim is that XP and Scrum together will fulfill all of ISO 9001’s requirements. Their discussion is a bit sketchy, but they are on the right tack when they state that “ISO 9001 does not equate quality. It simply helps ensure that your agile practices are being followed”. Boehm and Turner point their fingers at the same problem in [3] when they discuss the need to balance agility and discipline and observe that “Every successful venture in a changing world requires both agility and discipline”.

Vriens [4] has published a paper where he discusses the full range of CMM, ISO 9001 and their relationships to XP and Scrum. He observes that most of the ISO 9001 requirements are independent of development methods used and are covered by the existing processes.

One author who has done a really thorough job on agile development and ISO 9001 is Wright [5]. He has used an approach that has many ideas in common with the approach that we will use later in this paper – go through the ISO 9001 requirements item by item and see what XP and Scrum have to offer in line of conformance. We do, however, disagree with some of his statements and the overall conclusion that none of XP’s practices needed to be changed. We will look at two points that underline some of the problems with agile development when it comes to ISO 9001 – one is taken from the table on XP versus ISO 9001 and one is taken from the table on XP versus TickIT [6].

In his ISO 9001 versus XP table Wright has looked at the ISO 9001 item 7.3.4 “At suitable stages, systematic reviews of design and development shall be performed in accordance with planned arrangements”. The author claims that pair programming is a continuous code review. This claim does, however, not hold up against most of the available definitions of a code review – see for instance [7]. The design is not *systematically* reviewed in pair programming since the focus is on the other person’s coding. In addition, pair programming does not include documentation, which makes a later review difficult.

In Wright’s table of TickIT versus XP, he claims that “customer stories and acceptance tests fully define the software requirements”. There are two problems with this statement. Firstly, that the acceptance test defines the requirements is manifestly wrong. The acceptance test is written based on the requirements, not the other way around, although the new trend of automated acceptance testing may change this [27]. Secondly, the customer stories are way too imprecise to serve as requirements. It is

the stories *plus* the customer's acceptance – often not in writing – that define the requirements.

An approach similar to the one used by Wright is used by Ilieva et al [16]. They had a process that was already ISO 9001 conformant. Their problem was to identify how they could change the process in an agile direction and still stay ISO 900 conformant. They called this a gap analysis and the approach seemed successful – they introduced agile development in e-business development and management to stay ISO 9001 conformant.

Melis et al. [9] focus on part seven of the ISO 9001 – product realization, since this is the part of the ISO 9001 that it most heavily touched by agile development. The paper gives a good overview of the relation between agile development and ISO 9001 for this part of the standard but leave the rest untouched. The authors identify ISO 9001 items 7.3.2 – 7.3.7 as the most important challenges for making an ISO conformant agile process.

Keenan [10] has studied ISO 9001 and XP in order to use ideas from both in a process tailoring project. He states that “the desire to support an agile development philosophy is one of the main motivators” for looking at process tailoring.

A paper by Nawrocki et al. [11] is important because the authors have performed an experiment with XP and parts of ISO 9001. The main results, as reported in the paper, are that the XP projects in the experiment suffered from such problems as low maintainability and late delivery. In addition to part seven of the ISO 9001 standard, Nawrocki also studied the effect of agile methods on part eight - measurement, analysis and improvement.

The TickIT International has also looked into ISO 9001 and agile development. Southwell sums up his observations as follows [12]: (1) many of these principles address issues which are not really covered by ISO 9001 and TickIT and are therefore not in conflict with them, (2) several principles addresses similar concerns to those of ISO 9001 but goes further and (3) some of the principles are in complete agreement even if the approaches are rather different.

We have also found two master theses [13, 14] that treat the problem of agile vs. ISO 9001. Both contain reports from case studies, which make their works important. In addition, they have done a complete review of the ISO 9001 requirements. Their goal, however, was not to check the additions needed in agile development to stay conformant to ISO 9001 but to see how well the agile development projects in the case studies adhered to ISO 9001.

Vitoria, in [13] looks at the whole TickIT standard and analyzes how it has been used in two case study projects. For the two projects in question he found that 33% of TickIT could not be applied in an XP project, 24% could be partly applied, 20% could be applied in full, while 23% were not relevant since the two projects were student project.

Erharuyi [14] looks at part seven and eight of ISO 9001, just as Nawrocki [11]. As should be expected, his conclusions are pretty much the same as those of Nawrocki. However, his paper contains some blatant misunderstandings, such as the claim that test plan updates is part of corrective actions – ISO 9001, item 8.5.2.

Another interesting case study is presented by Stephen Sykes in [17]. This case study includes an auditor's report of all findings when auditing a company using the agile method Crystal. The main conclusion is that all nonconformities identified during the ISO 9001 audit can be solved with a little flexibility from all parties involved.

3 Agile Development

Agile software development is a way of organizing the development process, emphasizing direct and frequent communication – preferably face-to-face, frequent deliveries of working software increments, short iterations, active customer engagement throughout the whole development life-cycle and change responsiveness rather than change avoidance. This can be seen as a contrast to waterfall-like processes which emphasize thorough and detailed planning and design upfront and consecutive plan conformance. Over the past ten years or so agile methods have gained great interest and popularity as they seem to address recurring problems such as budget overruns, delivering the wrong features and generating a lot of overhead in the form of reporting, formalism, re-planning and extensive management. The basic concepts of agile software development are concisely described in the agile manifesto¹. Agile software development can be seen as a philosophy and several defined methods based on these ideas are in use, all sharing a common set of values and principles. The best known and most used agile methods are Extreme Programming (XP) [22] and Scrum [23].

The main constructs used in agile development are:

- Iteration: a short (2-4 weeks) period of analysis, design, development and testing. In Scrum, iterations are called sprints.
- Product backlog: a list of prioritized requirements for the product
- Sprint or iteration backlog: a selection of items from the product backlog being developed in an iteration
- Sprint review: an evaluation of the outcome of a sprint, done in cooperation with the customer to identify fulfilled requirements and requirements needing further improvement. This can also be viewed as a retrospective. Thus, the term review here refers to review of the software being developed.
- Sprint planning: is done in the start of an iteration or a sprint and results in a sprint backlog with items that in total can be developed within the timeframe of an iteration by the current team of developers.
- Standup-meeting: a daily short meeting where each team member reports on progress, plans and problems. This can include both product and process-related problems.

Compared to a strict water-fall model, an agile process involves and engages the customer both initially, in each iteration and in the finalization of the product. In each iteration, the customer collaborates with the development team for requirement specification, knowledge transfer and acceptance testing – see Fig. 1.

¹ www.agilemanifesto.org

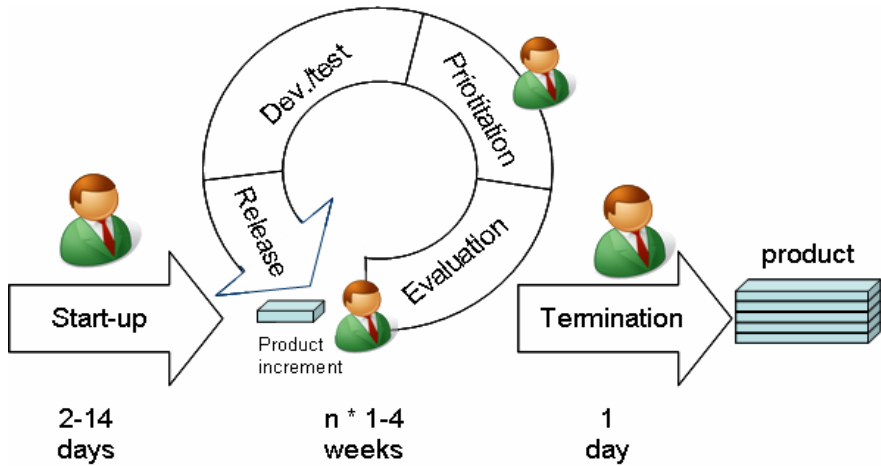


Fig. 1. Customer involvement in agile development

The ideas behind agile software development are not new [19] as they clearly are inspired by agile and lean manufacturing which have been in use in many types of industries for decades, the radical innovations in the Japanese post-war industry is probably the best known example [20]. Yet, some important changes need to be made to make this fit software development [21]. The most fundamental principle from lean development being applied is the principle of waste reduction: all work and work products not directly contributing to the development of software should be considered as waste and thus avoided or minimized.

Since the first book on Extreme Programming by Kent Beck was published in 1999 the interest and industrial use has grown surprisingly fast. The huge interest seen in industry does in most cases stem from the developers and can be explained by the simple and human-centric values carried out by agile methods which may be appealing to practitioners but threatening to management. The basic principles are easy to grasp and seems to address the most fundamental problems bugging developers. However, among this interest and willingness to radically change the development process, several critical voices have emerged and many experience reports indicate that it is not straight forward - in most cases it is an act of balancing agility and discipline [24].

One type of critique against agile methods is the deliberate avoidance of documentation as this may be considered as waste; documents are not software and software development should develop software, not documents. This is a strict concept and probably explains the common perception that agile software development is incompatible with well known quality assurance standards such as ISO 9001. As we will show later, this does, however, not necessarily have to be true.

4 The ISO 9001 Requirements

The requirements of ISO 9001 are, at the top level, summarized in a few points:

- The company must have a quality assurance management process – part 4.

- The product's quality is the responsibility of the management – part 5. As a result of this, the company's management must make the necessary resources available for quality assurance work and training – part 6.
- The company shall have one or more documented processes for product realization – part 7. The process must produce documents that can be (1) reviewed for acceptance and (2) used as proof of conformance.
- All reports of non-conformances, both of the product or the process, shall be reported and analyzed and should lead to a corrective action – part 8.

In addition to ISO 9001, the document ISO 90003 is also important. This is not a standard but a guideline for applying ISO 9001 to software development and maintenance.

ISO 9001 focuses on situations where we have or are about to sign a contract. The contract is signed on the basis of a defined process. One of the roles of this process is to give the customer confidence that the quality will be as specified and it is thus important that the process is followed.

ISO 9001 uses document review as its main control and verification mechanism. Many companies that use agile development claim that pair programming makes reviews unnecessary. In addition, there exist both experiments and case studies that show that pair programming, given the right conditions, give a lower failure rate than traditional software development – see for instance [18, 28] We can, however, not dispose of document reviews and still claim ISO 9001 conformance. A statement from DnV is enlightening here: “The main point is that verification shall be performed according to plan – see ISO 9001, item 7.3.5. The amount of verification needed must be adapted to the importance of each artifact and our confidence in the process that produced it. If the company has developed a strong confidence in the results from e.g. pair programming, it will be reasonable to move the resources somewhere else where there is a larger probability that the verification process will contribute to a better product quality. Thus, pair programming will influence the verification plan but cannot be used as an argument to suspend verification and verification planning”.

5 Comparing ISO 9001 with Agile Development

When comparing ISO 9001 and agile development, we have defined agile development as adherence to the basic principles as formulated in [19]. In order to compare agile development with ISO 9001, we used the same approach as we have earlier used when introducing ISO 90001 to a Norwegian company [8]. The approach is simple and pragmatic – use a table containing all the ISO 9001 items aligned with the corresponding item in the ISO 90003 guidelines. Have a free column where we can indicate whether agile development meets the requirements of ISO 9001 as explained in ISO 90003. All items not ticked off are given a closer scrutiny to see whether we need to add or change something in order to achieve conformance.

For the main activity for this paper – deciding what is conformant to ISO 9001 and what is not – we used three approaches: (1) what has been accepted by other auditors

[17], (2) our own expert judgment, based on developing an ISO 9001 conformant development process [8] and (3) the experience of two auditors from DnV – “Det norske Veritas”.

In addition to this, we compared our assessment to the assessments in [5], [13], [14] and [17] and did a closer investigation where there was a disagreement. This process left us with 15 items where agile methods only partly were able to fulfill the ISO 9001 requirements and four items where agile methods could not meet the requirements at all. As should be expected, part seven of the ISO 9001 standard dominates in both cases – nine out of 15 of the partly fulfilled items and two of the four items that were not fulfilled at all. On the positive side – of the 50 items in ISO 9001, 31 items will not need any changes or enhancements whatsoever by either side.

The results of the comparison process were used as our starting point for a three step process: (1) identify the reasons for the lack of full conformance, (2) see how these lacks can be amended, either by extending the development process or by augmenting the ISO 90003 guidelines. For a discussion of the confidence that can be placed in this approach, see section 7 – Threats to validity.

6 What Can Be Done to Achieve Conformance

This section is organized as follows: each ISO 9001 requirement that is not clearly conformant with agile development is quoted in italics. It is then followed by an explanation, change or adaptation to agile development that in our opinion is needed in order to meet the ISO 9001 requirement.

4.2.1.d: The quality management system documentation shall include documents needed by the organization to ensure the effective planning, operation and control of its processes.

As for any kind of development methodology, an agile development project always starts by defining how the methodology shall be used in the given project. The planning of an agile project can easily be documented in a simple form specifying for instance iteration length, how to record and track requirements etc.

4.2.4: Records shall be established and maintained to provide evidence of conformity to requirements and of the effective operation of the quality management system. Records shall remain legible, readily identifiable and retrievable. A documented procedure shall be established to define the controls needed for the identification, storage, protection, retrieval, retention time and disposition of records.

In between iterations, conformity to requirements is evaluated by the product owner and records of the results are kept as evidence of conformity.

5.3: Top management shall ensure that the quality policy

- *5.3 a: is appropriate to the purpose of the organization*
- *5.3 b: includes a commitment to comply with requirements and continually improve the effectiveness of the quality management system*

A well implemented agile process, with the necessary conditions in place will support the purpose of the organization. That is, delivering well functioning software within the compromised time- and cost frame. A well working agile process will ensure a dedicated commitment to requirements as these are continuously evaluated based on experience from development and testing.

5.4.1: Top management shall ensure that quality objectives, including those needed to meet the requirements for the product, are established at relevant functions and levels within the organization. The quality objectives shall be measurable and consistent with the quality policy.

The agile method Evo [29] emphasizes measurable quality objectives and this practice can easily be applied in other agile methods and meets the ISO 9001 requirement.

5.6.2: The input to management review shall include information on

- *5.6.2 a: results of audits*
- *5.6.2 b: customer feedback*
- *5.6.2 c: process performance and product conformity*

After each iteration, the process performance is reviewed and potential improvements are implemented in the following iteration. Reviews are based on input from developers and customer feedback. Output from such retrospectives [25] can in the context of ISO 9001 be used as input to management review.

7.1: The organization shall plan and develop the processes needed for product realization. Planning product realization shall be consistent with the requirements of the other processes of the quality management system (see 4.1). In planning product realization, the organization shall determine the following as appropriate

7.1a: quality objectives and requirements for the product

7.1 b: the need to establish processes, documents and provide resources specific to the product

The adaptation of an agile process at the start of the development project covers this requirement. Adaptation, or process planning, may include deciding iteration length, strategies for requirements documentation, staffing etc.

7.2.1a: The organization shall determine requirements specified by the customer, including the requirements for delivery and post-delivery activities.

Agile processes include practices for determining requirements but this is usually focused on features and qualities of the product itself. To cover the requirement of determining requirements for delivery and post-delivery activities this needs to be included. One way of dealing with this is to add additional sprints after delivery to deal with any post-delivery activities.

7.2.2: The organization shall review the requirements related to the product. The review shall be conducted prior to the organization's commitment to supply a product to the customer (e.g. submission to tenders, acceptance of contracts or orders, acceptance of changes to contracts or orders) and shall ensure that

- 7.2.2a: *product requirements are defined.*
- 7.2.2 c: *the organization has the ability to meet the defined requirements*
- 7.2.2 x2: *where the customer provides no documented statement of requirements, the customer requirements shall be confirmed by the organization before acceptance*

This ISO 9001 requirement is the most problematic if it is to be interpreted strictly. It requires the development organization to have the complete requirements defined upfront. However, agile methods actually do specify that requirements should be gathered upfront. They will, however, not be complete and will not contain all the details. This is based on the assumption that it is impossible to get a complete overview of all details up front; instead the most important aspects should be documented. In Scrum this is documented in the product backlog which is set up prior to the first iteration - at this time it constitutes the best possible understanding of the requirements. Compared to traditional requirements specifications it differs in the way that it is anticipated to change, based on experience from development. The conclusion of this issue is that if an auditor accepts this initial overview of requirements, agile methods fulfil this requirement. If not, we find that the fundamental principle of requirements evolution in agile methods is in conflict with ISO 9001.

7.3.1a: The organization shall plan and control the design and development of the product. During the design and development planning, the organization shall determine the design and development stages.

Agile methods cover planning and control of the product design. It differs from the traditional approach in that this is done iteratively and incrementally, yet it is handled. It will still produce documents that can be used as proof of conformance for the activities mentioned in 7.3.1a. Examples of documents that have been accepted as proof of conformance are e.g. pictures of the whiteboard showing requirements planned, in work or completed.

7.3.2: Inputs related to product requirements shall be determined and records maintained (see 4.2.4). These inputs shall include

- 7.3.2 a: *functional and performance requirements.*
- 7.3.2 x1: *these inputs shall be reviewed for adequacy. Requirements shall be complete, unambiguous and not in conflict with each other*

This is handled through the requirements process in agile methods. In front of each iteration, new requirements are gathered or existing requirements altered due to customer feedback. These requirements are then reviewed and recorded through cooperation between the customer and the development team. This includes both functional and performance requirements.

7.3.3: The outputs from design and development shall be provided in a form that enables verification against the design and development input and shall be approved prior to release. Design and development output shall

- 7.3.3 a: *meet the input requirements for design and development*

There seems to be a certain amount of disagreement in the agile camp as to whether e.g. XP requires design but all this aside, it is no problem to include a high level design activity in the first planning game and a low-level design at the start of each iteration.

7.3.4: At suitable stages, systematic reviews of design and development shall be performed in accordance with planned arrangements (see 7.3.1)

7.3.4 a: to evaluate the usability of the results of design and development to meet requirements

Design and development is reviewed in every transition between iterations as a joint effort between the development team and the customer. The customer is given a particular responsibility of evaluating usability and requirements conformance.

7.3.5: Verification shall be performed in accordance with planned arrangements (see 7.3.1) to ensure that the design and development outputs have meet the design and development requirements. Records of the results of the verification and any necessary actions shall be maintained

It is not a problem to include a high level design activity in the first planning game and a low-level design at the start of each iteration.

7.3.7: Design and development changes shall be identified and records maintained. The changes shall be reviewed, verified and validated, as appropriate, and approved before implementation. The review of design and development changes shall include evaluation of the effect of the changes on constituent parts and products already delivered. Records of the results of the review of changes and any necessary actions shall be maintained (see 4.2.4)

The review done after each iteration takes care of this. This is done as a joint effort between the development team, the customer or product owner and other possible stakeholders which amongst other issues consider changes to design and development. Decisions are documented in the form of an updated product backlog. If necessary, formal and signed minutes of meeting can be made to keep track of the design and development history.

8.1: The organization shall plan and implement the monitoring, measurement, analysis and improvement processes needed

- *8.1 a: to demonstrate conformity of the product*
- *8.1 b: to ensure conformity of the quality management system*

This is covered by the planning and adoption of the agile method being used. A central part of all agile methods is close monitoring of progress to early discover potential problems. The reviews done between iterations also include an evaluation of the development process itself to potentially identify software process improvement initiatives.

8.2.3: The organization shall apply suitable methods for monitoring and, where applicable, measurement of the quality management system process. These methods shall demonstrate the ability of the process to achieve planned results. When planned

results are not achieved, correction and corrective action shall be taken, as appropriate, to ensure conformity of the product.

Besides evaluation of product increments, the review in between iterations also may include a retrospective [25]. This has the same function as a traditional assessment of the process performance, potentially leading to process improvement actions to be implemented in the following iterations.

8.2.4: The organization shall monitor and measure the characteristics of the product to verify that product requirements have been met. This shall be carried out at appropriate stages of the product realization process in accordance with the planned arrangements (see 7.1)

Evidence of conformity with the acceptance criteria shall be maintained. Records shall indicate the person(s) authorizing release of product (see 4.2.4)

This is handled by the iteration reviews. Acceptance of requirements are documented e.g. in the product backlog or similar.

8.5.2: The organization shall take action to eliminate the cause of nonconformity in order to prevent recurrence. Corrective actions shall be appropriate to the effects of the nonconformity encountered.

A documented procedure shall be established to define requirements for

- *8.5.2 c: evaluating the need for actions to ensure that nonconformities do not recur*
- *8.5.2 d: determining and implementing action needed*
- *8.5.2 e: records of the results of action taken (see 4.2.4)*
- *8.5.2 f: reviewing corrective action taken*

The iteration reviews intend to discover nonconformity with requirements. This is either caused by too little resources in the previous iteration due to unforeseen difficulties, insufficient understanding of the requirements or a bad process. Only the latter case is of interest here. The process causes are registered and will later be used as input to a process improvement activity.

8.5.3: The organization shall take action to eliminate the cause of potential nonconformities in order to prevent their occurrence. Preventive actions shall be appropriate to the effects of the potential problems.

A documented procedure shall be established to define requirements for

- *8.5.3 a: determining potential nonconformities and their causes*
- *8.5.3 b: evaluating the need for action to prevent occurrence of nonconformities*
- *8.5.3 c: determining and implementing action needed*
- *8.5.3 d: records of results of action taken (see 4.2.4)*
- *8.5.3 e: reviewing preventive action taken*

The intention of having frequent reviews of development progress, requirements and process performance in cooperation with the customer is, among other things, to eliminate causes of potential nonconformities. As both the software product and related knowledge grows, the development team and the customer continuously

improve their ability to discover potential sources of nonconformity. Such reviews may, if relevant, produce software process improvement actions – both to reactively take immediate action and as a mean to proactively improve the development process for later development projects.

7 Threats to Validity

There are three threats to validity for our conclusion – have we (1) understood ISO 9001, (2) have we touched all relevant ISO 9001 items and (3) have we understood agile development in general and Scrum and XP in special?

7.1 Have We Understood ISO 9001

One of the authors has experience with helping a company becoming ISO 9001 certified and has a through knowledge of ISO 9001. Whenever we have been in doubt, we have consulted personnel at DnV who certify Norwegian companies. They have a large amount of ISO 9001 experience and have been able to clear up any misunderstandings that we might have had.

An ISO 9001 certification audit is, however, not an exact science. Different auditors may have different standards for what they find acceptable. Thus, there is always a possibility that what we have found acceptable – e.g. the Scrum planning process – may not be accepted as a planning process by some auditors.

7.2 Have We Touched All Relevant ISO 9001 Items

By using the standard itself plus its guideline, we went through the whole standard, item by item. All items that concerned documents, documentation or acceptance of documents, together with all issues pertaining to the implementation, validation and verification of a software system were assessed – see chapter 5.

In addition, we have coordinated our findings with four independent sources and all ISO 9001 items we identified were also identified by at least one of these sources. Thus, we are confident that all relevant ISO 9001 items are identified and assessed.

7.3 Have We Understood Agile Development

Agile development is not an exact defined methodology and there exist a handful of agile methods that varies [26]. Yet they are all based on the few common principles described in chapter 3. In our assessment we have tried to apply these common and fundamental principles to reduce a potential bias from our own interpretations of what agile development is, yet a certain level of subjective interpretation is inevitable.

7.4 Our Claims to Validity

Based on the discussion in the sections 7.1 to 7.3 we claim that our conclusions regarding ISO 9001 and agile development will be valid for a wide range of companies and auditing authorities.

8 Conclusion and Future Work

Based on the discussions in the chapter on threats to validity, we feel that our observations and conclusions are relevant for the topic.

The main difference between ISO 9001 and agile methods is that ISO 9001 insists on documentation for reviews and to demonstrate process conformity. Agile methods try to avoid writing documents that does not contribute to the finished system. On the other hand – if the customer requires a certain document, the use of agile methods are no hindrance for developing them.

There are ways to deal with many of the documents that ISO 9001 requires. We can add such activities as review meetings, writing design documents and so on. The process will still keep the most important agile ideas, such as short iterations, building in increments, including the customer, reprioritizing requirements whenever need, and constantly adjusting scope, time and cost within the bounds of the project contract. One often used slogan in the agile community is “Do the Simplest Thing that Could Possibly Work”. The term “simplest” does not mean it is forbidden to add extra process artifacts or activities. There are, however, limits to how many new artifacts that can be added to an agile method and still insist on labeling it agile. The changes necessary to be conformant to ISO 9001 are, however, well inside those limits.

We see from the discussions above that the differences between agile development and an ISO 9001 conformant development process are not insurmountable. Some changes are, however, needed. We suggest the following actions:

ISO: the ISO 90003 guidelines should include some guidelines concerning (1) what is accepted as a review (2) several types of reviews and (3) when each type of reviews is considered necessary.

Agile development: given the suggestions to the ISO 90003 guidelines, there remains two problems – that an agile process produce documents that can be used (1) as proof of conformance and that (2) can be reviewed as part of ISO 9001’s verification and validation.

When the abovementioned problems are solved, there will be no problems whatsoever when a company wants to use agile development and still keep its ISO 9001 certificate.

Acknowledgements

We gratefully acknowledge the help from T. Skramstad and F. Prytz, DnV for important input and fruitful discussions on the interpretation of ISO 9001.

References

- [1] Mnkandla, E., Dwolatzky, B.: Defining Agile Quality Assurance. In: The proceedings of the International Conference on Software Engineering Advances – ICSEA 2006 (2006)
- [2] McMichael, M., Lombardi, M.: ISO 9001 and Agile development. In: AGILE 2007 (2007)

- [3] Boehm, B., Turner, R.: Balancing Agility and Discipline: Evaluating and Integrating Agile and Plan-Driven Methods. In: the proceedings of 26th International Conference on Software Engineering – ICSE 2004(2004)
- [4] Vriens, C.: Certifying for CMM Level and ISO 9001 with XP@SCRUM. In: The proceedings of the Agile development Conference – ADC 2003 (2003)
- [5] Wright, G.: Achieving ISO 9001 Certification for an XP Company. In: Maurer, F., Wells, D. (eds.) XP/Agile Universe 2003. LNCS, vol. 2753. Springer, Heidelberg (2003)
- [6] The TickIT Guide, British Standards Institute, London, UK (2001)
- [7] IEEE Standard Glossary of Software Engineering Terminology, IEEE Std 610.12-1990, Standards Coordinating Committee of the Computer Society of the IEEE (1990)
- [8] Stålhane, T.: Implementing an ISO 9001 certified process. In: Proceedings of the EuroSPI Conference, Joensuu, Finland (2006)
- [9] Melis, M., et al.: Requirements for an ISO Compliant XP Tool. In: Eckstein, J., Baumeister, H. (eds.) XP 2004. LNCS, vol. 3092. Springer, Heidelberg (2004)
- [10] Keenan, F.: Agile Process Tailoring and Problem Analysis (APTLY). In: The proceedings of 26th International Conference on Software Engineering – ICSE 2004 (2004)
- [11] Nawrocki, J.R., et al.: Combining Extreme Programming with ISO 9000
- [12] Southwell, K.: Agile Process Improvement. In: TickIT International, Firm Focus on behalf of BSI-DISC ISSN 1354-588
- [13] Vitoria, D.: Aligning XP with ISO 9001:2000 – TickIT Guide 5.0 – A case study in two academic software projects, Master Thesis, School of Engineering, Blekinge Institute of Technology, Ronneby, Sweden (2004)
- [14] Erharuyi, E.: Combining eXtreme Programming with ISO 9000:2000 to Improve Nigerian Software Development Processes, Master Thesis, School of Engineering, Blekinge Institute of Technology, Ronneby, Sweden (2007)
- [15] Namioka, A., Bran, C.: eXtreme ISO. In: Proceedings of the OOPSL 2004, Vancouver, Canada, October 24-28 (2004)
- [16] Ilieva, S., et al.: Analysis of an agile methodology implementation. In: Proceedings of the 30th EUROMICRO conference (2004)
- [17] Cockburn, A.: Crystal Clear – A Human-Powered methodology for Small Teams. Addison-Wesley Longman, Amsterdam ISBN 0201 699478
- [18] Aiken, J.: Technical and Human Perspective on Pair Programming. ACM SIGSOFT Software Engineering Notes 29(5) (2004)
- [19] Merisalo-Rantanen, H., Rossi, M.: Is Extreme Programming Just Old Wine in New Bottles: A Comparison of Two Cases. Journal of Database Management (2005)
- [20] Takeuchi, H., Nonaka, I.: The New Product Development Game. Harvard Business Review (1986)
- [21] Poppendieck, M., Poppendieck, T.: Lean Software Development: An Agile Toolkit for Software development Mangers. In: Cockburn, A., Highsmith, J. (eds.) The Software Development Series. Addison Wesley, Reading (2003)
- [22] Beck, K.: Extreme programming explained: embrace change. Addison-Wesley, Reading (2000)
- [23] Schwaber, K., Beedle, M.: Agile Software Development with Scrum. Prentice-Hall, Englewood Cliffs (2001)
- [24] Boehm, B., Turner, R.: Balancing Agility and Discipline – A Guide for the Perplexed. Addison-Wesley, Reading (2004)
- [25] Derby, E. and Larsen, D.: Agile Retrospectives: Making Good Teams Great, 20067, Pragmatic Bookshelf

- [26] Abrahamsson, P., et al.: Agile software development methods – review and analysis, VTT Electronics (2003)
- [27] Mugridge, R., Cunningham, W.: Fit for Developing Software: Framework for Integrated Tests (R. Prentice Hall, Upper Saddle River (2005)
- [28] Arisholm, E., et al.: Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise. IEEE Transactions on Software Engineering 33(2), 65–86 (2007)
- [29] Gilb, T.: Competitive Engineering: A handbook for systems engineering, requirements engineering, and software engineering using Planguage. Elsevier, Butterworth-Heinemann (2005)

Study of the Evolution of an Agile Project Featuring a Web Application Using Software Metrics

Giulio Concas^{1,2}, Marco Di Francesco³, Michele Marchesi^{1,2},
Roberta Quaresima¹, and Sandro Pinna¹

¹ DIEE, Università di Cagliari, Piazza d'Armi,
09123 Cagliari, Italy

{concas, michele, roberta.quaresima, pinnasandro}@diee.unica.it

² FlossLab s.r.l., viale Elmas, 142
09122 Cagliari, Italy

³ Lab for Open Source Software, ICT District, Sardegna Ricerche, Piazza d'Armi,
09123 Cagliari, Italy
difrancesco80@gmail.com

Abstract. We present an agile process used for the development of a Web application written in Java, devised by choosing a set of proven agile practices taken by existing popular agile methodologies. During the project, we regularly measured the software using Chidamber and Kemerer object-oriented metrics suite, and other metrics. The application development evolved through phases, characterized by a different level of adoption of some key agile practices – such as pair programming, test-based development and refactoring. The evolution of the OO metrics of the system, and their behavior related to the agile practices adoption level is presented and discussed, showing that software quality, as measured using standard OO metrics, looks directly related to agile practices adoption.

Keywords: Software metrics, agile methodologies, object-oriented languages.

1 Introduction

Agile methodologies (AMs) have gained adoption in a wide variety of software development domains, so we can state that they have become mainstream in software engineering. Several experience reports have been reported in the literature regarding the successful adoption of AMs in different contexts. In fact the context, expressed in terms of criticality, size, culture, dynamism and personnel must be taken into account while choosing the practices, principles and values that represent the best trade off between discipline and agility for a specific project [4]. AMs have been adopted by small software firms and by large organizations, by co-located and by distributed teams, and in general they have been used for developing a wide range of software products.

However, the availability of software data enabling empirical software engineering studies about AMs is still scarce. To be valuable, these data should include, for each examined project, a detailed description of the context and a set of process and

product metrics. Classical software engineering evolved studying the typical software projects of the seventies and eighties, where users interact with a centralized system through specific user interfaces running on a terminal or on a client workstation. The first applications of agile methodologies, like the Chrysler C3 project [12] where Extreme Programming was born, were of this kind.

Nowadays, many software firms and open source communities have adopted agile values, principles and practices for developing applications that interact with the web using technologies like J2EE or .NET. The availability of qualitative and quantitative empirical data about Web application projects using agile methodologies is even more scarce, because this kind of development became popular only in the last years.

In this work we present in detail a software project consisting in the implementation of FlossAr, a Register of Research software for universities and research institutes, developed with a complete object-oriented (OO) approach and released with an open source license [10]. This is a Web application, which has been implemented through a specialization of the open source software project, jAPS (Java Agile Portal System) [13], a Java framework for Web portal creation released with GNU GPL 2 open source license. Throughout the project we collected metrics about the software being developed. Since Java is an OO language, we used the Chidamber and Kemerer (CK) OO metrics suite [5]. The adoption level of some key agile practices had been recorded as well during the project.

The goals of this paper are two: to present the principles and practices of the specific agile process we devised for the project; and to present some quantitative measurements performed on the system under development, and relate them with the adoption of some key agile practices. We found that some key software quality metrics show significantly different mean values and trends during different phases of the project, and that these changes can be positively related with the adoption of some agile practices, namely pair programming, test-based development and refactoring.

The paper is organized as follows: in section 2 we present the agile process we used; in section 3 we present the OO metrics computed on the software; in section 4 we present the software project, the development team and the phases of its development; in section 5 we present and discuss the results, relating software quality – as resulting from the metrics measurements – with the adoption of agile practices; section 6 concludes the paper.

2 Agile Practices

As FlossAr project had to be developed quickly, in front of requirements not well understood at the beginning, and with a high probability of changing, the management of FlossLab, the firm behind the development of FlossAr, decided to use an agile approach to develop the system. With the aid of the software engineering group of the University of Cagliari, the state of the art of agile software development was discussed, and a set of practices deemed most suited to our Web application development was devised.

Agile Methodologies are a recent approach to software development, introduced at the end of the nineties and now widely accepted worldwide as “mainstream” software engineering. AMs offer a viable solution when the software to be developed has fuzzy

or changing requirements, being able to cope with changing requirements throughout the life cycle of a project. Several AMs have been formalized, the most popular being Extreme Programming (XP) [3], Scrum [15], Feature Driven Development [8], DSDM [9] and others. All AMs follow the principles presented in the Agile Manifesto [1].

Very often, software teams willing to pursue an agile approach do not follow “by the book” a specific AM, but discuss and decide a set of agile practices to be used, and from time to time review the project and make adjustments to these practices.

Web application development is relatively new, and it lacks the many consolidated programming practices applied in traditional software development. One of the main peculiarities of this kind of development is a heterogeneous team, composed by graphic designers, programmers, Web developers, testers. Moreover, the application has typically to be run on different platforms, and to interact with legacy systems.

Consequently, the choice of the development practices to use is of paramount importance for the success of the project. The team and its advisors first defined some principles to be followed:

- **Code Reuse:** Not only the underlying framework, but also the specialized software produced must be reusable. In fact, there is no such thing as a general purpose register of research, but each university or research institution wish to customize such a system to cope with its specific features and requirements.
- **Evolvability:** It is very important that the software can be easily modified and upgraded, in a context where requirements were unclear since the beginning, and new requirements might be added continuously.
- **Maintainability:** Errors and failures must be fixed quickly, in every software module composing the system, minimizing the probability to introduce new bugs when old bugs are fixed.
- **Modularity:** The functional blocks of the underlying framework must be kept, and new modules must be added on each of them, with no intervention on the framework itself.
- **Portability:** This was a specific requirement of FlossAr, to be able to propose it to several research institutions, with different hardware and software contexts. For other specialization projects it might not be as important.

Following the above principles, the team chose and defined a set of agile practices, most of them derived from XP methodology [3]. They were:

- **Pair Programming:** This practice might be considered difficult to apply in the context of a heterogeneous team. In our case, however, it was one of the keys to the success of the project. All the development tasks were assigned to pairs and not to single programmers. Given a task, each pair decided which part of it to develop together, and which part to develop separately. The integration was in any case made working together. Sometimes, the developers paired with external programmers belonging to jAPS development community, and this helped to grasp quickly the needed knowledge of the framework.
- **On Site Customer:** A customer’s representative was always available to the team. This customer-driven software development led to a deep redefinition of the structure and features of the system, particularly in the first months of the project.

- **Continuous Integration:** The written code was integrated several times a day.
- **Small Releases:** Taking advantage of the customer on site, the development was divided in a sequence of small features, each separately testable by the customer, guaranteeing a high feedback level. There were three major releases, at a distance of two months each other.
- **Test-Driven Development (TDD):** All code must have automated unit tests and acceptance tests, and must pass all tests before it can be released. In its most extreme definition, tests are even written before the code. In the presented project, the choice whether to write tests before or after the code was left to programmers. However, they had a strong requirement that all code must be provided of tests.
- **Refactoring:** A continuous refactoring was practiced throughout the project, to eliminate code duplications and improve hierarchies and abstractions.
- **Coding Standards:** The same coding standards of the original jAPS project were kept to increase code readability.
- **Collective Code Ownership:** The code repository was freely accessible to all programmers, and each pair had the ability to make changes wherever needed. This was eased by the uniformity of technical skills of all team members.
- **Sustainable Pace:** This practice was enforced throughout the project, with the exception of the week before the main releases, when the team had to work more than forty hours to complete all the needed features in time.
- **Stand-up Meeting:** Every day, before starting the work, an informal short meeting was held by the team, to highlight issues and to organize the daily activities.
- **Feature List and Build by Feature:** These practices were inspired by FDD agile methodology [4]. The Feature List is also called “Backlog” in the terminology of Scrum. A list of the features to implement, ordered by their relevance, was kept in a Wiki, and the system development was driven implementing them. These features are user-oriented – meaning that most of them describe how the system reacts to user inputs – and have a priority, agreed with the on site customer. Each feature must be completed at most in one week of pair programming; longer features are divided in shorter sub-features satisfying this constraint. The Feature Design activity of FDD was seldom performed before programming, except in the first interactions of the development, when architectural choices were made, and UML diagrams were created to document them.

The resulting software process is a standard agile one, proceeding by short iterations and taking advantage of many “classical” agile practices, mainly taken from XP. Its main peculiarity is the control process, which is less structured than Scrum Sprint and XP Planning Game, with less meetings and standard artifacts.

2.1 Agile Practices and Software Quality

The goal of some of the agile practices quoted above is to enable the team to successfully answer to changes in the requirements, and to maximize feedback with the customer and among the team. On Site Customer, Small Releases, Stand-up Meeting, Feature List, Build by Feature and Sustainable Pace are all practices of this kind. These practices were adopted during the whole development process, and moreover do not directly prescribe how code is written. Therefore, it is impossible to assess

their impact on the quality of the code, expressed using the quality metrics described in the followings.

The other practices concern how the code is written and upgraded, and have a more direct impact on the quality of the code. Among them, we deem that the most effective to improve code quality are pair programming, TDD and refactoring.

Pair programming, by forcing two developers to work simultaneously at the same piece of code on the same computer, allows the team to share the knowledge of the system. The effectiveness of pair programming also depends on the pair rotation strategy, that allows the maximization of communication among developers.

A software system can be described as a network of interconnected components and the collective knowledge of such a system allows the team to easily add new functionalities and fix the bugs. On the other hand, the reduction of the amount of communication leads to specialization, i.e. the fact that a single developer has knowledge of, and is comfortable with, only a piece of the entire system. Given that in Object Oriented programming a class will cooperate with other classes, issues can arise when a programmer needs to use a module developed by another programmer. This gap of knowledge can lead to an increased fault proneness of the class, with consequent impact on its quality metrics.

TDD means that the production code should be adequately covered with automatic tests. Tests are an excellent documentation tool because they describe the behavior of a piece of code in term of assertions that compare actual and expected values. Tests may also be used as a design tool by writing them before the production code. Following a rigorous testing strategy, programmers are forced to write testable code and this encourages the production of quality code. In the case of classes with many collaborations with other classes, testing is difficult because these dependencies lead to other objects that must be properly initialized, or simulated with mock objects. It is reasonable to assume that a reduction of the testing level has an impact on coupling metrics and on the method length.

Refactoring is a practice which aims to simplify the system, without changing its functionalities. A software system can be represented as a network of interconnected entities and its readability can be improved by applying some refactoring practices. Some of these practices need a global knowledge of the system, and perform better if they are combined with pair programming and testing. For example, reducing the coupling among classes requires not only a local knowledge of them, but also the knowledge of the complex network of relations among these classes. Other refactoring practices are finalized to the reduction of local complexity, and can be performed by programmers with no global picture of the system.

3 The Measured Software Metrics

Throughout the project, we computed and analyzed the evolution of a set of source code metrics including the Chidamber and Kemerer suite of quality metrics (CK) [5], the total number of classes, the lines of code of classes (CLOCs) and methods (MLOCs). The quality of a project is usually measured in terms of lack of defects, or of maintainability. It has been found that these quality attributes are often correlated

with specific metrics. For Object Oriented systems, the CK metrics suite is the most validated in the literature. The CK suite is composed of six metrics:

- **Weighted Methods of a Class (WMC):** A weighted sum of all the methods defined in a class. Chidamber and Kemerer suggest assigning weights to the methods based on the degree of difficulty involved in implementing them [5]. In our case, we simply computed the number of methods. WMC is a measure of the complexity of a class.
- **Coupling Between Objects (CBO):** A count of the number of other classes with which a given class is coupled. To be more precise, class A is coupled with class B when at least one method of A invokes a method of B, or accesses a field (instance or class variable) of B. CBO denotes the dependency of one class on other classes in the system.
- **Response For a Class (RFC):** A count of the methods that are potentially invoked in response to a message received by an object of a particular class. It is computed as the sum of the number of methods of a class and the number of external methods called by them. RFC is both a measure of the complexity of a class, and of the potential communication between the class and other classes. In fact, in empirical measurements, RFC is often found to be correlated with both WMC and CBO – though WMC and CBO are not correlated with each other.
- **Lack of Cohesion in Methods (LCOM):** A count of the number of method pairs with zero similarity, minus the count of method pairs with non-zero similarity. Two methods are similar if they use at least one shared field (for example they use the same instance variable). LCOM measures the cohesiveness of methods within a class.
- **Depth of Inheritance Tree (DIT):** The length of the longest path from a given class to the root class in the inheritance hierarchy. DIT is the total number of superclasses of a given class, at all levels, and measures how many classes can influence the class through the inheritance mechanism.
- **Number of Children (NOC):** A count of the number of immediate subclasses inherited by a given class. It is a measure of the width of the inheritance tree whose root is the class.

CK metrics have been largely validated in the literature. In a study of two commercial systems, Li and Henry studied the link between CK metrics and the maintenance effort [14]. Basili et al. found, in another study, that many of the CK metrics were associated with fault-proneness of classes [2]. In another study, Chidamber et al. reported that higher values of CK coupling and the cohesion metrics were associated with reduced productivity and increased rework/design effort [6].

Among CK metrics, RFC, CBO and WMC are those that have been found most correlated with software quality in the literature [16]. LCOM was not always proved to be correlated with fault proneness or with maintenance effort related to a class, but this was the case in some key researches [6], [14]. DIT and NOC are usually considered less important than other CK metrics [11], [16]. In general, the lower is the value of CK metrics, the better the quality of the system.

We also consider the LOCs of classes and methods metrics. It is good OO programming practice to create small, cohesive classes, and to keep short the method LOCs, because every method should concentrate on just one task, and should delegate

a substantial part of its behavior to other methods. So, also LOC metrics should be kept reasonably low in a “good” system.

In this paper, for the sake of brevity we consider just the average values of CK metrics and LOCs metrics, averaged on all the classes or methods of the system. The average is just a rough measure of the metrics, because it is well known that the distributions of CK and LOCs metrics follow a power-law [7]. However, also given that the number of classes of the system is of the order of some hundreds, the average of these metrics should suffice to give an idea of the average quality of the system.

4 The Project and Its Phases

In this work we present in detail a software project consisting in the implementation of FlossAr, a Register of Research software for universities and research institutes, developed with a complete object-oriented (OO) approach and released with an Open Source license [10]. FlossAr manages a repository of data about research groups and research results – papers, reports, patents, prototypes – aimed to help research evaluation and matching between firms looking for technologies and knowledge, and researchers supplying them. It is a Web application, because both researchers who input their profiles and products, and people looking for information access the system through a standard Web browser.

FlossAr has been implemented through a specialization of an open source software project. We define specialization as the process of creating a software application customized for a specific business, starting from an existing, more general software application or framework. The general framework we customized is jAPS (Java Agile Portal System) [13], a Java framework for Web portal creation released with GNU GPL 2 open source license. jAPS comes equipped with basic infrastructural services and a simple and customizable content management system (CMS). It is able to integrate different applications, offering a common access point.

FlossAr has been developed by a co-located team of four junior programmers, coordinated by a team leader, adopting the agile process that has been defined in section 2. All developers had a master degree in computer engineering. The four junior programmers were just graduated and had experiences of OO design, OO programming and Web programming in Java and other languages, in projects carried on during their studies. One of them was fairly skilled in Web site design. They were taught during university courses about agile practices, but had almost no experience in applying them. This was their first project carried on in a team. The team leader had a two year experience in leading software teams and using XP practices.

The project evolved through five main phases, each one characterized by an adoption level of the key agile practices of pair programming, TDD and refactoring. These phases are summarized below:

- **Phase 1:** An exploratory phase where the team studied both the functionalities of, and the way to extend the underlying system (jAPS). It lasted three weeks, at the beginning of the project, and did not produce code. We will not consider this phase in the measurements and in the subsequent discussion.

- **Phase 2:** A phase characterized by the full adoption of all practices, including testing, refactoring and pair programming. It lasted ten weeks, leading to the implementation of a key set of the system features.
- **Phase 3:** This is a critical phase, characterized by a minimal adoption of pair programming, testing and refactoring, because a public presentation was approaching, and the system still lacked many of the features of competitors' products. So, the team rushed to implement them, compromising the quality. This phase lasted seven weeks, and included the first release of the system after two weeks.
- **Phase 4:** An important refactoring phase, characterized by the full adoption of testing and refactoring practices and by the adoption of a rigorous pair programming rotation strategy. This phase was needed to fix the bugs and the bad design that resulted from the previous phase. It lasted four weeks and ended with the second release of the system.
- **Phase 5:** Like phase 2, this is a development phase characterized by the full adoption of the entire set of practices, until the final release, after seven weeks. This phase includes three holiday weeks, that are not considered.

5 Results and Discussion

In this section we analyze the evolution of FlossAr source code metrics. At regular intervals of one week, the source code has been checked out from the CVS repository and analyzed by a parser that calculates the metrics. The parser and the analyzer have been developed by our research group as a plug-in for the Eclipse IDE.

The total number of classes of the system (including abstract classes and interfaces), which is a good indicator of its size, is shown in Fig. 1. The number of classes generally increases over time, though not linearly. The project started with 362 classes – those of jAPS release 1.6. At the end of the project, after 28 weeks (excluding those of phase 1 and the holidays), the system had grown to 514 classes, due to the development of new features that constituted the specialized system. In this and

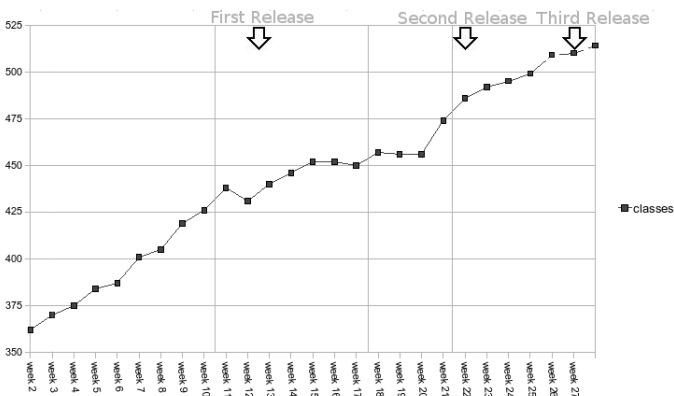


Fig. 1. The evolution of the number of classes

all the subsequent figures, we delimit with vertical lines the four main phases of development (from phase 2 to phase 5), and report as well with arrows the times of the three releases of the system.

In Table 1 we report the cross-correlation values between all pairs of metrics, computed using the complete time series related to the project, highlighting in bold those whose value is above 0.8. As you can see, many metrics are fairly correlated with each other. The most correlated with other metrics are the LOC ones. Also CBO and NOC metrics are quite correlated with others. WMC is the least correlated metric, being strongly correlated only with RFC. The only pair that is totally uncorrelated is WMC and DIT, and in general there is no metrics pair showing anti-correlation, even at a very low level. These results reflect the fact that, as the system grows, all considered metrics tend to grow. This grow is in part natural – because a bigger system tends to be more complex than a smaller one. However, we can also attribute this growth, that happened mostly during phase 2 and, above all, phase 3, to poor usage – or no usage – of pair programming, TDD and refactoring during these phases.

Table 1. Cross-correlation values between all measured metrics

Metrics	WMC	RFC	LCOM	CBO	DIT	NOC	CLOCs	MLOCs
WMC	1	0.94	0.46	0.58	0.09	0.41	0.76	0.53
RFC	0.94	1	0.57	0.81	0.32	0.64	0.91	0.77
LCOM	0.46	0.57	1	0.77	0.81	0.80	0.79	0.81
CBO	0.58	0.81	0.77	1	0.73	0.89	0.96	0.98
DIT	0.09	0.32	0.81	0.73	1	0.90	0.67	0.82
NOC	0.41	0.64	0.80	0.89	0.90	1	0.87	0.94
CLOCs	0.76	0.91	0.79	0.96	0.67	0.87	1	0.96
MLOCs	0.53	0.77	0.81	0.98	0.82	0.94	0.96	1

Table 2 shows the means and standard deviations of the six CK metrics and the two LOC metrics we computed, in the four relevant phases of the project, just to give an idea of their magnitude and variation.

Table 2. Statistics related to measured metrics, for the relevant project phases. The number in italics shown within brakes after the phase is the number of observations of the phase.

Metrics	Phase 2 [<i>10</i>]		Phase3 [<i>7</i>]		Phase 4 [<i>4</i>]		Phase 5 [<i>6</i>]	
	mean	st.dev.	mean	st.dev.	mean	st.dev.	mean	st.dev.
WMC	6.7	0.11	7.0	0.15	6.8	0.10	6.7	0.04
RFC	14.5	0.34	16.0	0.43	15.3	0.16	15.1	0.11
LCOM	25.3	0.4	30.0	5.2	34.7	1.3	33.0	0.4
CBO	3.98	0.10	4.64	0.16	4.52	0.06	4.68	0.03
DIT	0.75	0.006	0.77	0.002	0.81	0.003	0.81	0.005
NOC	0.59	0.006	0.61	0.002	0.62	0.005	0.62	0.005
CLOCs	67.9	1.92	80.2	3.45	78.8	0.78	77.4	0.82
MLOCs	9.53	0.15	10.7	0.27	10.8	0.05	10.9	0.07

In Fig. 2 we show the behavior of the mean values of the four CK metrics not related to inheritance – WMC, RFC, LCOM and CBO. All the values are normalized to

the maximum value reached by the metrics. During phase 2, all these metrics tend to grow, though to different extents. This growth continues during phase 3, up to a peak, that happens at the end of phase 3 or, in the case of LCOM, at the beginning of phase 4. During phases 4 and 5 these metrics tend to decline, except in the case of CBO, that shows a slow increase after a dip occurring at the beginning of phase 4.

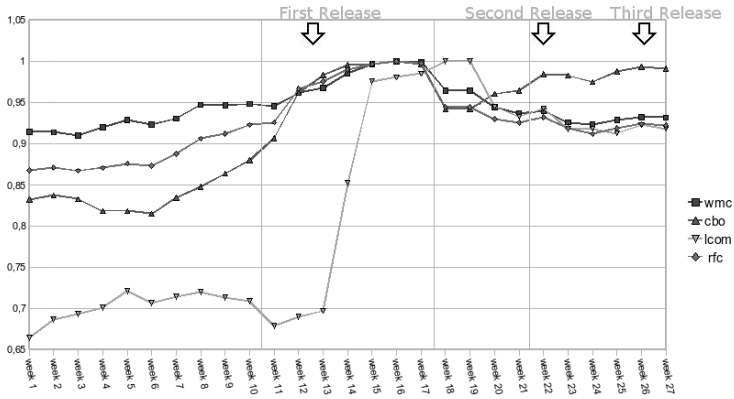


Fig. 2. The evolution of the mean value of WMC, RFC, LCOM and CBO metrics

In Fig. 3 we show the behavior of the mean values of the CK metrics related to inheritance – DIT and NOC – and of LOC metrics. Also in this case, the values are normalized to the maximum value. NOC metric shows a slow, steady increase throughout the project, looking unaffected by the coding practices used. Note that in literature this metric is often neglected, or found poorly correlated with software defects [16]. DIT metric behavior is quite stable too, except in the transition from phase 3 to phase 4, where there is a sudden increase of about 0.5. This is due to the fact that one of the first refactoring activities made in phase 4 was to restructure several inheritance hierarchies, factoring out common features in an added abstract superclass. This activity

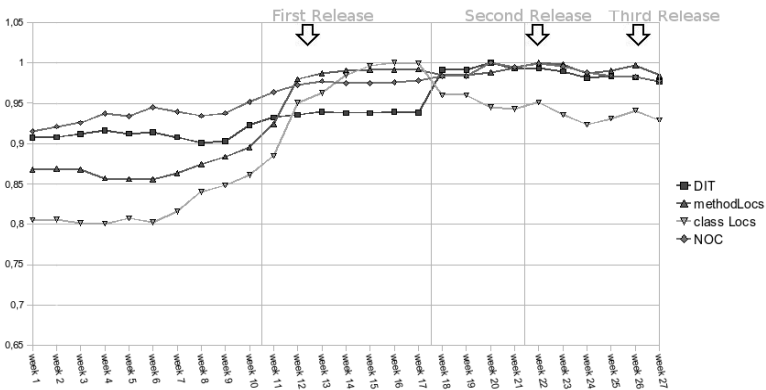


Fig. 3. The evolution of the mean value of DIT, NOC, CLOCs and MLOCs metrics

led to an increase by one of the depth of inheritance tree of about one half of the system classes, explaining the jump of DIT metric.

The evolution of both class and method LOCs shows an increasing trend during phase 2 and the beginning of phase 3, until just before the first release. Then, CLOCs metric tend to slowly decrease during phases 4 and 5, while MLOCs shows a relative stabilization, with some minor fluctuations, during the remaining of the project. Both LOCs metrics are strongly correlated with CBO and, to a lesser extent, with RFC metrics, as shown in Table 1.

Now, let us proceed with a deeper discussion of the results shown so far. As described in section 4, the level of adoption of key agile practices, and namely pair programming, testing and refactoring was highly variable in the different phases of the project. Since these practices were always applied or not applied together, it is not possible to discriminate among them, or to assess their relative usefulness, using the data gathered in this case study. Consequently, we will talk of “key agile practices” as applied together.

The evolution of many of the studied metrics in conjunction with the process phases, as shown in Figs. 1, 2 and 3, shows significantly different values and trends, depending on the specific phase. Our hypothesis is that this variability is due precisely to the different level of adoption of the key agile practices, because, to our knowledge, this is the only difference among the various phases, as regards external factors that might have an impact on the project. Moreover, the only relevant internal factor in play is the team experience, both regarding working together applying agile practices, and about the knowledge of the system itself. Since the project duration was relatively short, we estimate that that the latter factor affected significantly only phase 2.

We performed a Kolmogorov-Smirnov (KS) two-sample test to assess how these measurements differ from one phase to the next. This KS test determines if two datasets differ significantly, i.e. belong to different distributions, making no assumption on the distribution of the data¹. For each computed metric, we compared the measurements belonging to a phase to those belonging to the next. The results are shown in Table 3, showing in bold the cases with significance levels greater than 95%. Phase 2

Table 3. Confidence level that the measurements taken in two consecutive phases significantly differ, according to K-S two-sample test

Metrics	Phases 2-3	Phases 3-4	Phases 4-5
WMC	99.8%	85.1%	92.9%
RFC	99.99%	95.3%	92.9%
LCOM	91.5%	62.3%	92.9%
CBO	99.99%	85.1%	98.3%
DIT	99.95%	98.7%	92.3%
NOC	99.95%	98.7%	4.75%
CLOCs	99.99%	85.1%	92.9%
MLOCs	99.97%	90.3%	56.4%

¹ Since all the metrics computed at a given time depend also on the state of the system in the previous measurement, the assumption underlying KS test that the samples are mutually independent random samples could be challenged. However, we used KS test to assess the difference between measurements in different phases as if they were independent sets of points, and we believe that at a first approximation the KS test result is still valid.

metrics differ very significantly from phase 3 in all cases but for LCOM – but getting a significance higher than 90% also for LCOM. The difference of the metrics of other consecutive phases are lower, though in several cases there is a significance greater than 95%, and in most cases greater than 90%. These results in fact confirm the difference in trends and values of the various metrics in the various phases that are patent in Figs. 2 and 3. Now, let us discuss the metrics trends during the various phases.

Phase 2 is characterized by a steady growing trend of the number of classes. All metrics, but LCOM, are stable during the first five weeks of this phase, and then tend to grow – in particular RFC, CBO, CLOCs and MLOCs. LCOM, on the contrary, tends to increase during the first four weeks, and then stabilizes. The starting values of all these metrics are those of the original jAPS framework, constituted by 367 classes and evaluated by code inspection as a project with a good OO architecture. The increase of RFC and CBO denotes a worsening of software quality. Note that phase 2 is characterized by a rigorous adoption of agile practices, but we should consider two factors:

- The knowledge of the original framework was initially quite low, so the first addition of new classes to it in the initial phase had a sub-optimal structure, and it took time to evolve towards an optimal configuration.
- Some agile practices require a time to be mastered, and our developers were junior;

In general, we might conclude that in phase 2 the team steadily added new features, and consequently new classes to the system. In the first half of the phase, however, these classes substantially kept the structure of the original system they were added to. As the system grew, this structure was slowly impaired, due to the factors quoted above.

Phase 3 is characterized by a strong pressure for releasing new features and by a minimal adoption of pair programming, testing and refactoring practices. In this phase we observe a growth in most metrics, namely all CK metrics related to coupling and complexity – with an explosive growth of LCOM – and in the LOC metrics. Only inheritance-related CK metrics are not affected. This shows that in this phase the quality has been sacrificed for adding several new features.

Phase 4, that follows phase 3, is a refactoring phase where the team, adopting a rigorous pair programming rotation strategy together with testing and refactoring, were able to reduce the values of many important quality metrics, such as WMC, RFC, LCOM, CBO and CLOCs. In this phase, no new features were added to the system. However, the number of classes increased during this phase, because refactoring required to split classes that had grown too much, and to refactor hierarchies, adding abstract classes and interfaces. In particular, WMC and RFC complexity metrics were very significantly reduced since the beginning of phase 4, and LCOM was reduced as well, mainly at the end of the phase. CBO shows a strong decrease in the first weeks, followed by an increase at the end of the phase. The increase of DIT and corresponding increase of CLOCs, as discussed before, are due to the addition of abstract classes to the hierarchies, that factor out common features, thus reducing the code of many classes. Method LOCs was not reduced, but it stopped to grow. Note that the values of the metrics at the end of phase 4 seem to reach an equilibrium.

The last development phase (phase 5) is characterized by the adoption of pair programming, testing and refactoring practices, and by the addition of further classes associated to new features. In this phase the metrics don't change significantly – although in the end the values of most of them are slightly lower than at the beginning of the phase – maybe because the team has become more effective in the adoption of the agile practices compared to the initial phase 2.

In conclusion, in phase 2 we observed a deterioration of quality metrics, that significantly worsened during phase 3; phase 4 led to a significant improvement in quality, and phase 5 kept this improvement. The only external factors that changed during the phases were adoption of pair programming, TDD and refactoring, that was abandoned during phase 3, and systematic use of these practices during phase 4, aiming to improve the quality of the system and with no new feature addition. As regards internal factors, in phase 2 the team was clearly less skilled in the use of agile practices and in the knowledge of jAPS framework than in subsequent phases.

Although it is not possible to draw definitive conclusions observing a single, medium-sized project, these observations quantitatively relate software quality metrics with the use of key agile practices, and this relation is positive – when pair programming, TDD and refactoring are applied, the quality metrics improve, when they are discontinued, these metrics become significantly worse.

6 Conclusions

In this paper we presented an agile process supporting the development of FlossAr, a Web application for managing a Register of Research. The process was devised having in mind the specificities of the project and of the team; it uses several agile practices, taken from XP, Scrum and FDD.

During the development we systematically performed measurements on the source code, using standard software metrics that have been proved to be correlated with software quality. Moreover, the development itself evolved through phases, characterized by a different adoption level of some key agile practices such as pair programming, TDD and refactoring, and by different team skills in using these practices and in the knowledge of the software framework which the system was built upon.

We were able to correlate several relevant quality metrics with the agile practices adoption, showing a significant difference in quality metrics of software developed in the various phases, and a systematic improvement of software quality metrics when agile practices are thoroughly used by skilled developers. Clearly, these results represent just a first step toward a more rigorous and systematic assessment of the effect of the use of agile practices on software quality, as measured using standard metrics.

Future work will be performed measuring other software projects, taking into account also the number of bugs and the effort to fix them. We also plan to use metrics more sophisticated than the simple computation of the average of metrics computed on the classes of the system. Such metrics will be related to behavior of the distributions of CK metrics on the class population, especially in the tail of the distribution, or could be complexity metrics computed using the complex network approach [7].

References

1. Agile Manifesto, <http://www.agilemanifesto.org>
2. Basili, V., Melo, L.B.: A validation of object oriented design metrics as quality indicators. *IEEE Trans. Software Eng.* 22, 751–761 (1996)
3. Beck, K., Andres, C.: *Extreme Programming Explained: Embrace Change*, 2nd edn. Addison-Wesley, Reading (2004)
4. Boehm, B., Turner, R.: *Balancing Agility and Discipline*. Addison-Wesley Professional, Reading (2003)
5. Chidamber, S., Kemerer, C.: A metrics suite for object-oriented design. *IEEE Trans. Software Eng.* 20, 476–493 (1994)
6. Chidamber, S., Kemerer, C.: Managerial use of metrics for object oriented software: An exploratory analysis. *IEEE Trans. Software Eng.* 24, 629–639 (1998)
7. Concas, G., Marchesi, M., Pinna, S., Serra, N.: Power-Laws in a Large Object-Oriented Software System. *IEEE Trans. Software Eng.* 33, 687–708 (2007)
8. De Luca, J.: *A Practical Guide to Feature-Driven Development*. Prentice-Hall, Englewood Cliffs (2002)
9. DSDM Consortium and Stapleton, J. *DSDM: Business Focused Development*, Pearson Education (2003)
10. FlossAr site, <http://www.flosslab.it/flosslab/en/flossar.wp>
11. Gyimothy, T., Ferenc, R., Siket, I.: Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction. *IEEE Trans. Software Eng.* 31, 897–910 (2005)
12. Haungs, J.: Pair Programming on the C3 Project. *IEEE Computer* 34(2), 118–119 (2001)
13. JAPS: Java agile portal system, <http://www.japsportal.org>
14. Li, W., Henry, S.: Object oriented metrics that predict maintainability. *J. Systems and Software* 23, 111–122 (1993)
15. Schwaber, K.: *Agile Project Management with Scrum*. Prentice-Hall, Englewood Cliffs (2001)
16. Subramanyam, R., Krishnan, M.S.: Empirical Analysis of CK Metrics for Object-Oriented Design Complexity: Implications for Software Defects. *IEEE Trans. Software Eng.* 33, 687–708 (2007)

Identifying and Understanding Architectural Risks in Software Evolution: An Empirical Study

Odd Petter Nord Slyngstad¹, Jingyue Li¹, Reidar Conradi¹, and M. Ali Babar²

¹ Department of Computer and Information Science (IDI), Norwegian University of Science and Technology (NTNU), Trondheim, Norway

{oslyngst, jingyue, conradi}@idi.ntnu.no

² LERO– The Irish Software Engineering Centre, University of Limerick, Limerick, Ireland
malibaba@lero.ie

Abstract. Software risk management studies commonly focus on project level risks and strategies. Software architecture investigations are often concerned with the design, implementation and maintenance of the architecture. However, there has been little effort to study risk management in the context of software architecture. We have identified risks and corresponding management strategies specific to software architecture evolution as they occur in industry, from interviews with 16 Norwegian IT-professionals. The most influential (and frequent) risk was “Lack of stakeholder communication affected implementation of new and changed architectural requirements negatively”. The second most frequent risk was “Poor clustering of functionality affected performance negatively”. Architects focus mainly on architecture creation. However, their awareness of needed improvements in architecture evaluation and documentation is increasing. Most have no formally defined/documented architecture evaluation method, nor mention it as a mitigation strategy. Instead, problems are fixed as they occur, e.g. to obtain the missing artefacts.

Keywords: software architecture, software evolution, risk management, software architecture evaluation.

1 Introduction

Modern software systems are commonly built by acquiring and integrating various components developed by commercial or open source entities. The software engineering community has enabled several processes for developing and maintaining component-based systems. Proper handling of software architecture is one of the most important factors towards successful development and evolution of component-based systems. However, there has been little effort to identify and understand the architectural risks in software evolution and potential strategies to deal with those risks. We assert that it is important to obtain and disseminate the information about potential risks (i.e. problems) in architecture evolution, as the architecture constitutes the central part of a software system [1]. Knowledge and understanding about architecture evolution risks should facilitate the development of improved strategies to mitigate these risks.

We have decided to obtain such knowledge from practicing IT-professionals working with software architecture, as they are expected to encounter risks (i.e. problems that may occur) in evolving software architectures on a regular basis. Our research here is concerned with Component-Based Software Engineering (CBSE) development, where there has been architectural evolution during the systems' lifetime.

Using a convenience sample of respondents, we carry out a preliminary investigation of architectural risks and management strategies in software evolution. This means changes to the structure(s) of a system of software elements, their external properties and mutual relationships, all viewed from a perspective of risk analysis and risk mitigation. This exploratory study is targeted at Norwegian IT-professionals who hold significant knowledge and experience in designing and evolving software architectures.

We have identified architectural risks (i.e. problems identified in planning or experienced during the maintenance/evolution) and associated risk management strategies (i.e. methods to mitigate these issues) as they occur in industry. "Lack of stakeholder communication affected implementation of new and changed architectural requirements negatively" was the most influential as well as the most frequent risk. This risk was most effectively mitigated by extending the time used towards communication with stakeholders. "Poor clustering of functionality affected performance negatively" was the next most frequent risk. This risk was in turn most successfully mitigated by refactoring or improving the modifiability of the architecture.

Furthermore, architects easily handle anticipated or experienced risks. However, their focus is usually on "forward engineering", not on reengineering (i.e. the architecture solution rather than the suitable steps to get there [14] in advance). Despite this, some of the findings also show that awareness of software documentation and evaluation issues and practices is increasing. Also, most of the respondents have no formally defined or documented architecture evaluation method in place. Rather, challenges are met as they appear, and the main focus is on obtaining the missing artifact. Finally, none of our respondents mentioned using formally defined or documented architecture evaluation as a risk mitigation strategy.

The remainder of the paper is organized as follows: Section 2 holds Background. Research Design is in Section 3. Section 4 contains information on our data collection, and the results of our study are in Section 5. Discussion and Threats to Validity are located in Section 6, and Conclusions and future work are in section 7.

2 Background and Related Work

Software Architecture [1] can be defined as the discipline dealing with the structure or structures of a system, comprising software elements, the externally visible properties ("interface" of in-going and out-going calls) of those elements, and the relationships between them. Well-defined software architecture is one of the key factors in successfully developing and evolving a non-trivial system or a family of systems. A well-defined software architecture provides a framework for the earliest design decisions to achieve functional and quality requirements. In addition, it has a profound influence on

project decomposition and coordination. Poor architecture often leads to project inefficiencies, poor communication, and inaccurate decision making [1]. The above definition of software architecture refers to software elements, which can be seen as components of the given software system. Hence software architecture is closely related to CBSE [2].

Clerc et al. [14] conducted a study to understand architects' attitudes towards software architecture knowledge. They found that architects are aiming more at creation and communication instead of review and maintenance of a system's architecture. Bass et al. [21] analyzed the output from 18 ATAM evaluations to discover risk themes specifically for software architecture. Besides a set of risk categories, they found that the more prevalent risks are those of omission (i.e. of not taking action on a particular issue). They also did not find a link between the risk categories and the business/mission goals or the domain of a system. Bass et al. further comment that the similarities to their study shown in [23] indicate the industrial relevance of the risk categories [21], as well as the ability of ATAM analysis to discover architectural risks. Another risk categorization from ATAM evaluations is presented in O'Connell [22], using 8 evaluation results. Although study [22] was analyzed independently from [21], the resulting themes are similar in content. It should be noted though that neither of these studies deal explicitly with the evolution of software architecture.

The architecture of a system will evolve as architectural changes are accumulated over time. There are diverging views in the research community about how software evolution should be defined. These include considering maintenance as a broader term [5], seeing evolution as a step in the software lifecycle [4], and regarding evolution as software systems' dynamic behavior through maintenance and enhancements [3]. Some [9] consider evolution as the enhancement and improvement performed on a system between releases. Based on this description, we define software evolution for this study as: *the systematic and dynamic updating in new/current development or reengineering from past development of component(s) (source code) or other artifact(s) to a accommodate new functionality, b improve the existing functionality, or c enhance the performance or other quality attribute(s) of such artifact(s) between different releases.*

If left unchecked, over time, a system's architecture will naturally decay as new quality and functional requirements are imposed on it. This decay is manifested by the original architectural structure(s) being lost. This is sometimes called "software rot" [20], and is one of the most prevalent reasons behind reengineering the architecture of a software system.

Risk management entails methods to mitigate risks that may occur during a software development project. Boehm [8] describes a framework for risk management consisting of two main steps, namely risk assessment (identification, analysis, and prioritization) and risk control (planning, resolution, and monitoring). Ropponen and Lyytinen [6] have identified six elements of software risk. Their results reveal influence on risk elements by environmental factors (e.g. development method). Also, awareness of risk management importance and method(s) was shown to have an effect. Keil et al. [10] conducted a risk management survey of project managers. They identified several additional important risk factors in comparison with

Boehm [8], contributing these to changes in the industry since Boehm's study. Additionally, they discovered that important risks were commonly out of managers' control. They therefore suggested that project managers widen their attention beyond traditional software risk factors.

Further based on the definition of risk in Boehm's article[8], as well as input from [6][12], we use the following definition for architectural evolution risks: *the issues or problems that can potentially have negative effects on the software architecture of a system as it evolves over time, hence compromising the continued success of the architecture*. The above studies on architectural risks [21][22] have focused on discovering risk categories directly from the output of ATAM [1] analyses. They use analysis outputs from organizations where such evaluation is an established practice. However, they do not comment on how commonly such formal evaluation methods are used in industry. Nor do they take software evolution specifically into account. In [7], the authors found that evaluation practices could range from completely ad-hoc to formally planned, from qualitative to quantitative. They also discovered that the approach depended on the goals of the evaluation. This means that additional risk issues and management strategies could be left undiscovered by looking only at output from structured analysis reports. We therefore decided to employ semi-structured interviews to gather qualitative information on risk issues and risk management strategies.

3 Research Design: Context, Motivation and Research Questions

We observed that risks and risk management strategies are commonly studied in relation to general software development [11][12][13], identifying risks on the project level [6][8][10]. Similarly, software architecture studies often focus on the design, implementation and maintenance of the architecture. While these results are important, there has been little effort to study risk management in the context of software architecture [21][22]. Hence, we decided to carry out an empirical study to help further identify and better understand the risks and risk management strategies in relation to software architecture.

This research is limited to those software systems which have two major characteristics: use of CBSE and changes in the systems' software architectures during their lifetimes. This means projects that have at least delivered the first production release, i.e. can be said to be in the "maintenance" phase.

Our main motivation is to obtain insight into the actual risks (i.e. issues identified and experienced which may affect the software architecture negatively) and associated risk management strategies (i.e. effective mitigation methods), as they occur in industry, in relation to software architecture evolution. We aim to use the results from this exploratory study as basis for more in-depth studies in this area.

This study is aimed at identifying and understanding risks and strategies relevant to software architecture evolution. That is, we investigate the steps of risk identification, analysis and prioritization, as well as risk planning and resolution [8], as they occur in industry. We do not cover issues pertaining to risk assurance or monitoring [8]. The research questions are as follows:

RQ1: What are the relevant architectural risks of software evolution, i.e. what software architecture related risks can be encountered during software evolution?

Any issue that can affect a project adversely if not handled correctly is considered a risk [8]. The first step in Boehm's risk management framework [8] entails risk identification, analysis, and prioritization. We are hence here interested in investigating the state-of-the-practice regarding risk awareness, i.e. to obtain insight on which risks that software architects deem more important in relation to software architecture evolution.

As aforementioned, software architecture is the central part of a software system [1], so failure of the software architecture can easily cause the entire project to fail. Hence a proper focus on the software architecture is needed to ensure the project is kept on budget and schedule. Similarly, changes to the software architecture can cause subsequent changes in many components of a software system [1]. It is therefore imperative to be aware of the possible risks incurred on the software architecture through software evolution.

RQ2: How can these risks best be assessed; through which methods or mechanisms were these risks identified, analyzed and prioritized?

Software architecture evaluation is widely known as an important and effective way to assess architectural risks [1, 7]. In order to identify, analyze and prioritize [8] risks there is the need for effective methods or mechanisms for software architecture evaluation. Such mechanisms help validate architecture design decisions with respect to required quality attributes (such as testability, availability, modifiability, performance, usability, security etc.). Prior architecture analysis studies [21][22] focused on structured analysis outputs as a method to discover risks. However the analysis methods used can range quite widely [7]. Investigating a wider range of analysis methods will help discover risk issues possibly missed by earlier studies.

RQ3: How can these risks best be mitigated: what were the relevant risk management strategies? Were the strategies successful or not?

The second step in Boehm's framework [8] encompasses risk control. This step focuses on problem mitigation; it is aimed at handling problems to minimize their impact. Here, our aim is to obtain the status quo, and suggest possible improvements by enabling a systematic approach to architectural risk management in software evolution. It is therefore imperative that we receive information on both positive and negative aspects of employed risk management strategies, and also on their outcomes.

Again, risks in relation to the central part of a software system (i.e. the architecture [1]) are important. Proper management of these risks on the three levels, technical, process and organization [11][12][13], provides the ability to minimize the potentially far-reaching impacts of these risks [8].

In order to practically explore the three research questions above, we designed an interview guide consisting of six questions. The relation between the questions in the interview guide, the research questions, and Boehm's framework [8] is shown in Table 1.

Question Q6 has been adapted from an earlier empirical study aimed at identifying the factors that can influence software architecture evaluation practices [7]. We also gathered demographic data (e.g. level of experience) about the respondents. The

Table 1. Relation between research questions and the interview guide

	Identification, Analysis, and Prioritization [8]	Assessment [8]	Planning, and Resolution [8]
Questions in the interview guide	RQ1	RQ2	RQ3
Q1.1. Describe architectural problems (indicate influence) and strategies (rate outcome) you identified in planning maintenance/evolution?	X		X
Q1.2. Describe architectural problems (indicate influence) and strategies (rate outcome) experienced and employed during maintenance/evolution?	X		X
Q2. Indicate weighting of and any changes in the following quality attributes[1]: testability, availability, modifiability, performance, usability and security) in your software architecture?	X		
Q3. How has the architecture changed throughout the lifetime of the system?	X		
Q4. Please describe your architecture change process?		X	X
Q5 Which architectural patterns (e.g. layering, task control, AI approach pipe-and-filter etc.) did you use to design the architecture?	X		
Q6. Does your organization use a defined and/or documented method or process to evaluate software architecture?		X	X

interview guide was piloted with 3 researchers to ensure quality and ease of understanding, through which the questions were polished and refined. We aimed to be flexible so as to gain as much qualitative information on each question as possible. Therefore, all the questions (Q1-Q6) were left open-ended. Also, the influence of each risk and the outcome of each strategy were indicated on a 5-point Likert scale. That is, risk Influence was ranked Very High = 5 to Very Low = 1. Similarly, strategy Outcome success was ranked Completely = 5, Mostly = 4, Medium = 3, Somewhat = 2 and Not at all = 1 successful.

4 Data Collection and Analysis

This study was carried out using a convenience sample of participants from the software industry in Norway. Potential respondents were first contacted by email, and sent the invitation letter with interview guide to get an overview. Later the potential respondents were contacted again by phone and signed up for a phone-interview appointment if they agreed to participate. The respondents were 16 IT-professionals in different companies with prior knowledge and experience with software architecture.

The phone interviews took on average 30 minutes to carry out, and we obtained complete responses to all the six questions from all 16 respondents. The data was recorded on paper and transcribed into electronic form. The responses were also summarized and read back to the respondents directly after the interviews, so they could be checked for accuracy.

Nine of the respondents had bachelor level degrees, while seven had master degree level educations. On average, the respondents had 8 years of experience working with software architecture, with six having less than five years of experience, five having 5-10 years of experience and another five having over 10 years of experience.

We analyzed the data as follows: The data was initially analyzed by dividing the data into discrete parts and coding each piece according to risk or strategy theme(s). As an example, for risks this was done as {condition – what may go wrong, consequence(s)}: e.g. “requirements from earlier versions still in effect affected architecture design negatively.” was coded as {earlier version requirements, negative for architecture design}.

We then examined them for commonalities and differences, and grouped related pieces of information based on their coding (e.g. for risks, {earlier version requirements, negative for architecture design} and {required same functionality as before, negative for planning} were grouped as {required backward compatibility, negative for architecture maintenance/evolution planning and design}). Each respondent’s transcript was run through this procedure. The results were checked by a second researcher to ensure reliability. This is similar to the constant comparison method described in [16]. The issues identified in the data analysis were classified into three categories; technical, process and organizational. We believe that risk management is not merely a technical issue; rather, it spans all three categories [11][12][13][21].

5 Results

The results are here divided into categories of (1) technical, (2) process and (3) organizational risks. This means that we have combined the findings from Q1.1 and Q1.2 for RQ1 and RQ3.

Table 2. Most **influential (Influence \geq 4) technical risks (TRs)** and corresponding management strategies performed

Technical	ID	Risk	Influence	Strategy	Outcome
Identified in planning	TR 1	Poor clustering of functionality affected performance negatively	4	Refactoring of the architecture	5
Experienced during	TR 2	Poor original core design prolonged the duration of the maintenance/ evolution cycle	4	Improve modifiability of the architecture	3
	TR 3	Increased focus on modifiability contributed negatively towards system performance	4	Implementation of changes towards modifiability	3
	TR 4	Varying release cycles for COTS/OSS components made it difficult to implement required changes	4	Use own development as potential backup	3
	TR 5	Poor clustering of functionality affected the performance negatively	4	Implement extra architecture add-ons	1

Table 3. Most influential (influence ≥ 4) process risks (PRs) and corresponding management strategies performed

Process	ID	Risk	Influence	Strategy	Outcome
Identified in planning	PR1	Lack of architecture documentation contributed to more effort being used on planning the maintenance/ evolution	4	Recover arch. documentation from current architecture design	5
	PR2	Lack of architecture evaluation delayed important maintenance/ evolution decisions	4	Recover evaluation artefacts where needed	5
				Alter process to capture important details	5
Experienced during	PR3	Lack of stakeholder communication affected implementation of new/ changed architectural requirements negatively	5	Negotiated project extension	3
				Allow additional time for communication/feedback	5
	PR4	Insufficient requirements negotiation contributed to requirement incompatibilities on the architecture	4	Postponed some requirements to next maintenance/evolution cycle	3
	PR5	Poor integration of architecture changes into implementation process affected implementation process and the architecture design negatively	4	Overlay new architecture change process onto implementation process	5
				Integrate architecture considerations into implementation process	3
	PR6	Using Software Change Management (SCM) sys. w/o explicit software architecture description contributed to inaccuracies in communicating the architecture	4	Use separate system for architecture description (using ADL), link to SCM system	3
				Trial use of additional ADL system	3
	PR7	No standard terminology affected internal and external communication efforts negatively	4	Align terminology with literature	1
				Extra communication to clarify terminology	1
PR8	Customer architects being unfamiliar with architecture change process affected maint./ evo. cycle schedule negatively	4	Extra communication effort with own resident architect to clarify	5	

Technical risks: Table 2 shows the most influential technical risks and corresponding management strategies performed. From Table 2, we can see that the strategy applied in planning towards TR1 was Completely successful (Outcome = 5). Furthermore, overall the strategies were also 3 out of 5 of Medium success (Outcome = 3), and 1 out of 5 Not at all successful (Outcome = 1).

Table 4. Most influential (influence ≥ 4) organizational risks (ORs) and corresponding management strategies performed

Organization	ID	Risk	Influence	Strategy	Outcome
Identified in planning	OR 1	Architecture team on a per maintenance/evolution cycle basis contributed to loss of knowledge about the existing architectural design	4	Dedicated personnel to "retrieve" knowledge	3
	OR 2	Cooperative maintenance / evolution with architects from customer organization required extra training and communication efforts	4	Frequent, interactive, scheduled meetings to keep up to date	5
	OR 3	Lack of clear point of contact from customer organization contributed to inconsistencies in communication of the architecture and requirements	4	Involve all "layers" of customer organization as stakeholders, allow extra communication time	5
	OR 4	Not allowed to change OSS as decision mandate external to architecture team, affecting performance and modifiability negatively	4	Ensure compliance with external mandate holder	3
Experienced during	OR 5	Separate architecture team per maint. / evo. cycle contributed to insufficient knowledge about the existing architectural design	4	Regain architecture details from upper management remaining	3
	OR 6	Prior architecture maint./ evo. by other projects due to lack of personnel made it difficult to obtain existing architecture design documentation	4	Merge architecture knowledge / documentation to central location	3
	OR 7	Large architecture team affected division of duties and subsequently implementation of maint./ evolution cycle negatively	4	Divide duties between subgroups	3
	OR 8	Lack of clear lead architect affected implementation progress negatively and contributed to extra effort needed	4	Merge duties and diverge roles more clearly	3

Process risks: Table 3 (below) shows the most influential process risks and corresponding management strategies performed. These results (Table 3) show that all of the strategies used in response to the most influential risks in planning were Completely successful. Towards the risks experienced during the maintenance/evolution, the

strategies were 3 out of 10 Completely successful, 5 out of 10 of Medium success, while 2 out of 10 were Completely successful.

Organizational risks: Table 4 (below) shows the most influential organizational risks and corresponding management strategies performed. Among the strategies used in response to these most influential organizational risks (Table 4) identified in planning, 2 out of 4 were Medium successful, while 2 out of 4 were Completely successful. Towards those experienced during, the strategies were all Medium successful.

Additionally, our results show that the overall most frequent (and most influential) risk was “Lack of stakeholder communication affected implementation of new and changed architectural requirements negatively”. The most successful strategy in response to this risk was “Allow additional time for communication for communication and feedback”. The second most frequent risk was “Poor clustering of functionality affected performance negatively”, with “Refactoring the architecture” and “Improve the modifiability of the architecture” as corresponding most successful strategies. The results from questions Q2, Q3, Q5 (Table 5), and Q4, Q6 are below.

Table 5. Summary of additional findings for RQ1

Q2. Quality attribute foci:	
<ul style="list-style-type: none"> • Focus on any given QA can change during the project. • Only a few projects experienced a lowering of focus on a given QA. • Most frequent QA with increased focus was Modifiability, followed by Usability. 	
Q3. Architecture changes made during system lifetime to:	
<ul style="list-style-type: none"> • Improve processing speed or scale (7 out of 16) • Improve flexibility to accommodate future changes (7 out of 16) • Accommodate new or altered user requirements (5 out of 16) 	<ul style="list-style-type: none"> • Improve system uptime (3 out of 16) • Enable additional access interfaces (1 out of 16) • Increase abstraction level (1 out of 16) • Support additional record types (1 out of 16)
Q5. Architectural patterns used (as means to solve design challenges):	
<ul style="list-style-type: none"> • Inversion of Control (1 out of 16), • Layered (3 out of 16), • Blackboard (3 out of 16), 	<ul style="list-style-type: none"> • Model View Controller (4 out of 16), • Pipeline (3 out of 16), • Task Control (2 out of 16), and • Broker (1 out of 16).

The following are results from Q4 (**RQ2, RQ3**) (architecture change process):

- none used a strictly defined change process,
- 7 out of 16 performed this process informally,
- 4 out of 16 employed loosely defined procedures,
- 3 out of 16 changed the architecture as part of the development process, and
- 2 out of 16 just change the architecture as needed.

In question Q6, none of the respondents answered that they have a defined or documented process for software architecture evaluation. 5 out of 16 of the respondents have a loosely defined process in place if needed. Another 5 out of 16 have knowledge of evaluation processes or methods mentioned in literature. Yet another 5 out of 16 of the respondents carry out a software architecture evaluation informally if needed. Finally, 1 out of 16 of the respondents reports that her/his organization has a process for

software architecture evaluation in place (in this specific case, based on the Architecture Tradeoff Analysis Method – ATAM [1]), but this is not commonly used.

6 Discussion

6.1 Comparison to Related Work

The Technical risks identified by the respondents show a high focus on design and creation of the architecture, supporting [14].

While Ropponen’s [6] focus was overall software development risks, ours is software architecture risks in software evolution. The strategies used in response to the risks we identified as (See Table 6 below) “Architecture Team” and “Requirements” risks were reported as being Medium or Completely successful in outcome. We can hence support the notion that there is at least some success in managing risks related to “Architecture Team” and “Requirements” [6].

A summarized comparison with the above and Bass et al. [21] is also in Table 6.

Table 6. Summary of comparison to related work

ID	Ropponen et al. [6]
	Requirements risks:
PR4	“Insufficient requirements negotiation contributed to requirement incompatibilities”
TR3	“Increased focus on modifiability contributed negatively towards system performance”
	Architecture Team risks:
OR5	“Separate architecture team per maint. / evo. cycle contributed to insufficient knowledge about the existing architectural design“
OR7	“Large architecture team affected division of duties and subsequently implementation of maint./ evo. cycle negatively”
OR8	“Lack of clear lead architect affected implementation progress negatively and contributed to extra effort needed”
	Stakeholder risks (from the subcontractor viewpoint):
PR3	“Lack of stakeholder communication affected implementation of maint./ evo. cycle negatively”
OR2	“Cooperative maint./evo. w/ architects from customer organization required extra training and communication efforts”
OR3	“Lack of clear point of contact from customer organization contributed to inconsistencies in communication of the architecture and requirements”
PR8	“Customer architects being unfamiliar with architecture change process affected maint./evo cycle schedule negatively”
ID	Bass et al. [21]
	Quality Attribute risk:
TR3	“Increased focus on modifiability contributed negatively towards system performance”
	Integration risks:
TR4	“Varying release cycles for COTS/OSS components made it difficult to implement required changes”
OR4	“Not allowed to change OSS as decision mandate external to architecture team, affecting performance and modifiability negatively“
	Requirements risks:
PR4	“Insufficient requirements negotiation contributed to requirement incompatibilities on the architecture”
TR3	“Increased focus on modifiability contributed negatively towards system performance”
	Documentation risks:

Table 6. (continued)

PR1	“Lack of architecture documentation contributed to more effort being used on planning the maintenance/evolution”
PR6	“Using Software Change Management system w/o explicit software architecture description contributed to inaccuracies in communicating the architecture”
	Process and Tools risks:
PR2	“Lack of architecture evaluation delayed important maintenance/evolution decisions”
PR6	“Using Software Change Management system w/o explicit software architecture description contributed to inaccuracies in communicating the architecture”
	Allocation risks:
TR1	“Poor clustering of functionality affected performance negatively”
TR4	“Varying release cycles for COTS/OSS components made it difficult to implement required changes”
	Coordination risks:
PR3	“Lack of stakeholder communication affected implementation of maint./evo. cycle negatively”
PR8	“Customer architects being unfamiliar with architecture change process affected maint./evo cycle schedule negatively”
OR2	“Cooperative maint./evo. with architects from customer organization required extra training and communication efforts”
OR3	“Lack of clear point of contact from customer organization contributed to inconsistencies in communication of the architecture and requirements”
OR4	“Not allowed to change OSS as decision mandate external to architecture team, affecting performance and modifiability negatively”

6.2 Observations on Key Architectural Risks and Promising Risk Management Strategies

The most influential Process risks we identified (Table 3) show that the **main focus is still forward thinking (producing systems according to budget and schedule) rather than hindsight reflection and learning.** Further, from the answers to Q5 we can see that the consequences of using one or more specific patterns are neither explicitly considered, nor evaluated as potential risks (though tactics, packaged by patterns, is a risk issue also discovered from ATAM reports in [21][22]).

The answers from Q4 and Q6 also point towards this main focus. Hence there is no apparent specific focus on discovering potential problems (rather problems are fixed as they are encountered, focussing on the missing artefacts). This is despite the potential benefits (e.g. identifying architecture design errors and potentially conflicting quality requirements early) of defined and documented architecture evaluation described in the literature [1]. However, architects are **becoming aware that their practices around evaluation and documentation need improvement.** This is echoed by the Organizational risks we identified (Table 4), such as “Architecture team on a per maint./evo. cycle basis contributed to loss of knowledge about the existing architectural design” and “Large architecture team affected division of duties and subsequently implementation of maint./evo. cycle negatively”.

A **link to Business Risks** [19] (i.e. those that affect the viability of a software system) can also be seen. The architectural risks identified are influenced by and in turn also affect such elements as e.g. cost, schedule.

Considering the most influential **Technical risks** (table 3), we can see that the majority of them were experienced during the maintenance/evolution, without prior planning. The same appears the case for the most influential **Process risks**, whereas

for the most influential **Organizational risks** half were identified in planning, and another half were experienced during the maintenance/evolution. In terms of management strategies, one overall trend appears to be that those employed in response to risks identified in planning had a more successful outcome. This appears especially to be the case where the same risk was both identified in planning as well as experienced during the maintenance/evolution (e.g. Technical risks TR1 and TR5: “Poor clustering of functionality affected performance negatively”). These findings also emphasize the points about forward engineering and awareness discussed above.

One of the strategies applied towards technical risks, as well as two of the strategies applied towards process risks were Not at all successful. These strategies should be viewed in light of the respective projects’ context (Tables 2, 3). Additionally, improvement is needed in the employed strategies, especially regarding issues encountered during maintenance/evolution. The lack of a strictly defined and documented architecture change process reported by the respondents (Q4) is also an interesting finding. We would expect architecture evaluation to be part of a given change process in order to analyze the consequences of proposed architectural changes.

To improve this situation, we believe that rigorous documentation and evaluation of architecture should be made an integral part of a software architecture change process. Furthermore, management of risks specific to architectural modifications should be given more attention. To achieve these objectives, software architects should be provided appropriate training. Moreover, organizational management should also demonstrate commitment to implement changes to the way software architecture changes are handled.

6.3 Threats to Validity

Threats to validity (using definitions provided by Wohlin et al. [15]):

Construct Validity: The research questions are rooted firmly in the research literature, and the actual questions in the interview guide have direct relations to the research questions. The interview guide was refined through pre-testing among our colleagues to ensure quality. All the terms used in the guide were defined at the beginning to avoid any potential misinterpretations.

External Validity: This study has been conducted by using a convenience sample of 16 IT-professionals, an issue which remains a threat. Nevertheless, obtaining a random sample is almost unachievable in software engineering studies because our community lacks good demographic information about populations of interest [17]. The respondents were chosen by us based on their background and experience with software architecture. Each respondent nevertheless represents a different company.

Internal Validity: The respondents are all knowledgeable and from the software industry, and have expressed an interest in the study. They all have the needed knowledge and background to provide informed answers. We hence believe that they have answered the questions to the best of their ability, truthfully and honestly, drawing on their own experiences, skills and knowledge. We also clarified any ambiguities in the questions or the accompanying definitions during the actual interviews, in addition to the definitions provided in the guide.

Conclusion Validity: This is an exploratory study. The findings are based on analyzing data from a relatively small number of software architects. We plan to implement a large scale study to confirm the results of this study. However, the exploratory nature of the study has identified several issues that may cause architectural risks for evolving systems. The insights gained will also function as background for refining the interview guide towards expansion of the sampling base for the planned larger scale study.

7 Conclusion and Future Work

We conducted phone-based, semi-structured interviews of 16 software architects from Norway for an exploratory study regarding risks and risk management strategies occurring in industry related to software architecture evolution.

Our findings include an initial identification of risks and corresponding risk management strategies as they occur in industry. Our main observations include that “lack of stakeholder communication affected implementation of new and changed architectural requirements negatively” was the most influential and frequent risk. The corresponding most successful strategy was to “Allow additional time for communication and feedback”. In second place concerning most frequent risks came “Poor clustering of functionality affected performance negatively”. The most successful management strategies towards this risk were “Refactoring the architecture”, and “Improve the modifiability of the architecture”.

Furthermore, architects’ main concerns are towards designing and creating the architecture. However, our results also show some awareness towards improvements in relation to how these tasks are performed, as well as towards the importance of retaining knowledge about and performing evaluation of the architecture. As most respondents have no formally defined or documented method to evaluate software architecture, problems are fixed as they occur with focus on the lacking artefacts rather than on the method.

Our results here will be used as input for a larger study in the software industry to survey the state-of-practice on risk and risk management regarding software architecture evolution. In particular, we plan to explore the relation between risks and risk management practices, and project context factors.

Acknowledgements

We thank all parties involved. The study was performed in the SEVO project, a Norwegian R&D project in 2004-2008 with contract number 159916/V30.

References

1. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice, 2nd edn. Addison-Wesley, Reading (2004)
2. Bass, L., Buhman, C., Comella-Dorda, S., Long, F., Robert, J., Seacord, R., Wallnau, K.: Volume I: Market Assessment of Component-based Software Engineering in SEI Technical Report number CMU/SEI-2001-TN-007 (2001)

3. Belady, L.A., Lehman, M.M.: A model of a Large Program Development. *IBM Systems Journal* 15(1), 225–252 (1976)
4. Bennett, K.H., Rajlich, V.: Software Maintenance and Evolution: A Roadmap. In: *ICSE 2000 – Future of Software Engineering*, Limerick, Ireland, pp. 73–87 (2000)
5. Sommerville, I.: *Software Engineering*, 6th edn., p. 728. Addison-Wesley, Reading (2001)
6. Ropponen, J., Lyytinen, K.: Components of Software Development Risk: How to Address Them? A Project Manager Survey. *IEEE Transactions on Software Engineering* 26(2), 98–112 (2000)
7. Ali Babar, M., Bass, L., Gorton, I.: Factors Influencing Industrial Practices of Software Architecture Evaluation: An Empirical Investigation. In: *QoSA 2007*, Medford, Massachusetts, USA, July 12-13 (2007)
8. Boehm, B.W.: Software Risk management: Principles and Practices. *IEEE Software* 8(1), 32–41 (1991)
9. Carr, M., Kondra, S., Monarch, I., Ulrich, F., Walker, C.: Taxonomy-Based Risk Identification, Technical Report SEI-93-TR-006, SEI, Pittsburgh, USA (1993)
10. Keil, M., Kule, P.E., Lyytinen, K., Schmidt, R.C.: A Framework for Identifying Software Project Risks. *Communications of the ACM* 4(11), 76–83 (1998)
11. Boehm, B.W.: A Spiral Model of Software Development and Enhancement. *IEEE Computer* 21(5), 61–72 (1988)
12. Gemmer, A.: Risk Management: Moving Beyond Process. *IEEE Computer* 30(5), 33–41 (1997)
13. Hecht, H.: *Systems Reliability and Failure Prevention*. Artech House Publishers (2004)
14. Clerc, V., Lago, P., van Vliet, H.: The Architect’s Mindset. In: *QoSA 2007*, Medford, Massachusetts, USA, July 12-13 (2007)
15. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: *Experimentation in Software Engineering – An Introduction*. Kluwer Academic Publishers, Dordrecht (2002)
16. Seaman, C.B.: Qualitative Methods in Empirical Studies of Software Engineering. *IEEE Transactions on Software Engineering* 25(4), 557–572 (1999)
17. Lethbridge, T.C., Sim, S.E., Singer, J.: Studying Software Engineers: Data Collection Techniques for Software Field Studies. *Empirical Software Engineering* 10(3), 311–341 (2005)
18. Kitchenham, B., Pfleeger, S.L.: Principles of Survey Research, Parts 1 to 6. *ACM Software Engineering Notes* (2001 – 2002)
19. Messerschmitt, D.G., Szyperski, C.: Marketplace Issues in Software Planning and Design. *IEEE Software* 21(3), 62–70 (2004)
20. Johnson, R.E., Foote, B.: Designing Reusable Classes. *Journal of Object-Oriented Programming* 1(2), 22–35 (1988)
21. Bass, L., Nord, R., Wood, W., Zubrow, D.: Risk Themes Discovered Through Architecture Evaluations. In: *Proc. WICSA 2007* (2007)
22. O’Connell, D.: Boeing’s Experiences using the SEI ATAM® and QAW Processes (April 2006), <http://www.sei.cmu.edu/architecture/saturn/2006/OConnell.pdf>
23. Charette, R.N.: Why software fails. *Spectrum* 42(9), 42–49 (2005)

A Hands-On Approach for Teaching Systematic Review

Maria Teresa Baldassarre, Nicola Boffoli, Danilo Caivano, and Giuseppe Visaggio

Department of Informatics, University of Bari – RCOST Bari
{baldassarre, boffoli, caivano, visaggio}@di.uniba.it

Abstract. An essential part of a software engineering education is technology innovation. Indeed software engineers, as future practitioners, must be able to identify the most appropriate technologies to adopt in projects. As so, it is important to develop the skills that will allow them to evaluate and make decisions on tools, technologies, techniques and methods according to the available empirical evidence reported in literature. In this sense, a rigorous manner for analyzing and critically addressing literature is Systematic Review. It requires formalizing an answerable research question according to the problem or issues to face; search the literature for available evidence according to a systematic protocol and retrieve data from the identified sources; analyze the collected evidence and use it to support decision making and conclusions. In this paper we report on how Systematic Review has been integrated in the “Empirical Software Engineering Methods” course that is taught at the Department of Informatics at the University of Bari, and how students have been introduced to this type of literature review through a hands-on approach. As far as we know, it is the first attempt of including a complex topic like systematic review in a university course on empirical software engineering. We have no empirical evidence on the effectiveness of the approach adopted, other than practice-based experience that we have acquired. Nonetheless, we have collected qualitative data through a questionnaire submitted to the students of the course. Their positive answers and impressions are a first informal confirmation of the successful application of our strategy.

Keywords: Empirical Software Engineering, Systematic Review, Statistical Process Control, Evidence Based Software Engineering.

1 Introduction

Empirical Software Engineering (ESE) is an important component of any software engineer’s curricula as it trains students to evaluate and make decisions on tools, technologies, techniques and methods according to the available evidence reported in literature. Indeed this is pointed out in the “Guidelines for Software Engineering Education” [3] that shows how SEEK (SE Education Knowledge) can be taught according to the volume’s guidelines. For each knowledge area, there is a short description and then a table that delineates the units and topics for that area. For each knowledge unit, recommended contact hours are designated. For each topic, a Bloom taxonomy level [5] (indicating what capability a graduate should possess) and the topic’s relevance (indicating whether the topic is essential, desirable, or optional to

the core) are designated. In this context, ESE can be categorized as part of the Mathematical and Engineering Fundamental. In particular, within this knowledge area, it can be seen as *empirical methods and experimental techniques* topic of the *engineering foundations for software unit*. This area is classified, with an “Essential” relevance, i.e. the topic is part of the core, and according to the Bloom taxonomy level it is considered as “c” comprehension, i.e. students should be capable to understand information and the meaning of material presented. For example, be able to translate knowledge to a new context, interpret facts, compare, contrast, order, group, infer causes, predict consequences, etc. Meyer [19] points out that software engineering education trains professionals for the industry, while ESE develops the *skills for empirically validating* tools, techniques used, developed and aimed for industry.

The above considerations point out how important it is to develop the skills that will allow software engineering students, also future practitioners, to critically and systematically evaluate the best available evidence on a specific issue of interest, may it be a tool, method, technique or other. In order to achieve such goal students must be able to apply the steps involved in evidence based software engineering [14]:

1. convert a problem or need for information into a research question
2. search the literature for evidence to answer the question
3. critically analyze the evidence
4. combine the evidence with previous knowledge and individual experience
5. evaluate performances and eventually make improvements.

This asks for concepts in empirical software engineering, in that students must acquire the fundamental elements of empirical methods and techniques used for validating a set of hypotheses, as well as skills for searching literature and critically addressing research questions. At the Department of Informatics at the University of Bari our graduate course in Informatics includes a mandatory course called “Empirical Software Engineering Methods”. In accordance to the SEEK guidelines, the course introduces students to empirical methods such as surveys, case studies and experiments. It also gives elements of empirical based software engineering (formalized in the above steps) and trains students on how to empirically evaluate software engineering tools, techniques, methods and technologies. Within the course, we have achieved step 2 of the above activities through a “Systematic Review”. A systematic review is a formal approach for reviewing research literature [17]. As reviews are often limited to annotated bibliographies, a systematic review means giving appropriate breadth and depth, rigor and consistency, let alone effective analysis and synthesis of the literature. Furthermore, it can be considered as much more effort prone than an ordinary literature survey. The latter being formally defined as “the selection of available documents (both published and unpublished) on the topic, which contain information, ideas, data and evidence written from a particular standpoint to fulfill certain aims or express certain views on the nature of the topic and how it is to be investigated, and the effective evaluation of these documents in relation to the research being proposed” [12]. More so, it has less scientific value than a systematic review, formally defined as a “means of evaluating and interpreting all available research relevant to a particular research question or topic area or phenomenon of interest” [17].

In this paper we describe how students have been introduced and addressed to carrying out systematic reviews as part of the above listed EBSE process within the

Empirical SE Methods course at the University of Bari. To make things easier and more interesting we have used a hands-on approach and actively involved students in a real systematic review on the topic of Statistical Process Control (SPC) [9]. We have carried out some type of qualitative evaluation to assess the student opinions. Results point out positive answers and impressions and therefore confirm the approach we adopted.

The rest of the paper is organized as follows: section 2 introduces the reader to the basic concepts of systematic review which we consider part of the EBSE process; section 3 illustrates the approach used to involve university graduate students in a review on the topic of statistical process control. Section 4 presents the general comments on the opinions collected; finally conclusions are drawn.

2 Searching Evidence through a Systematic Review

An important step of the EBSE process is the search of evidence for answering the research question. The more evidence found (of both positive and negative results on the topic being investigated), the more support to rational decision making is assured. Search of evidence can be done informally from various information sources such as retrieving customer or software user viewpoints, or asking for expert judgment; or formally as research-based evidence from sources such as scientific journals, books, grey literature. In our course we emphasize systematically searching evidence through a rigorous approach like systematic review. As so, step 2 of the EBSE process consists in searching the appropriate evidence through a systematic review.

Guidelines on systematic review have been defined and are quite stable in contexts such as medicine, social sciences, education and information sciences and used for analyzing and synthesizing existing empirical results on a certain topic. Indeed, there are many existing guidelines in this field that include the Cochrane Reviewer's Handbook [7], Guidelines of the Australian National Health and medical Research Council [1, 2]; CRD Guidelines for those Carrying Out or Commissioning Reviews [16].

Adaptations of these guidelines to software engineering have been made by Kitchenham in [17]. Also, applications of the procedure for performing a systematic review are becoming more and more common to the software engineering context in the past few years [6, 10, 11] and many studies have been carried out on various topics of interest that range from cost-estimation [15], within and cross company estimation models [18] to software process improvement [20], to statistical power [8].

We have defined and begun a systematic review on Statistical Process Control (SPC) [9]. Further details on the study itself can be found in [4].

In this paper we describe how graduate students of our Empirical SE Methods course have been actively involved in conducting the systematic review.

2.1 Systematic Review Concepts

In spite of the growing importance that systematic review has been achieving in the past years, it is still a quite new topic to the software engineering community. As so, before going on, we will provide the reader some preliminary concepts to make the rest of the paper easier to understand. More details on how to organize a systematic

review can be found in [17], this section synthesizes the information extracted from this report.

In general, a systematic review can be seen as a process made up of three main phases: planning the review, conducting the review, reporting the review (Fig. 1).

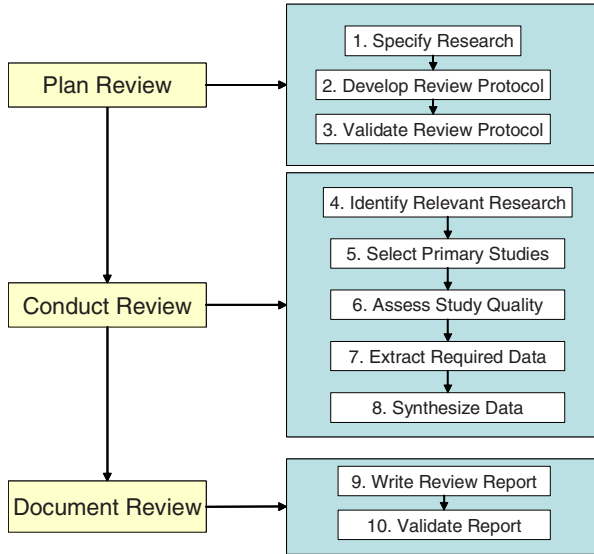


Fig. 1. Systematic Review Process

This first phase of **planning** involves *specifying the research* by motivating the need for information search; *developing the review protocol*: the review protocol is what formally specifies the steps and procedures used for carrying out the systematic review. It is important for the protocol to be defined before starting the review in order to avoid that results are in any way influenced by researcher expectations and desiderata. At this point the research questions are formulated and search strings are defined based on an analysis of the questions. In defining a search string it is important to keep in mind that: major terms are identified from the topic area, intervention and outcomes. Search strings should include synonyms, related terms and alternative spelling for major terms; boolean “OR” is used to incorporate alternative spellings and synonyms; boolean “AND” is used to link major terms. The following details should be used for constructing search strings:

- Population: Software development and maintenance projects and tasks
- Intervention: Statistical process control
- Outcomes: Reported benefits , reported limitations, task type, software attribute controlled (e.g. productivity, defect rate)

The search process is divided into two parts. First primary sources are identified from scientific journals, bibliographical databases, digital libraries, electronic databases on the Internet. Next, secondary sources are searched. The secondary search phase

consists in checking primary sources, identified in the initial search phase, for other relevant publications; and in contacting researchers who authored primary sources who we believe could be working on the topic, to enquire whether they have other unpublished papers or technical reports (i.e. grey literature).

Specific tables (Data Extraction Forms and Aggregation Tables) must then be defined for documenting all the outcomes of the search process and for accurately collecting and recording the information of all reviewed papers. The entire process should be rigorously documented, for example in spreadsheets, database tables. As final part of the protocol, selection criteria and procedures have to be defined. They determine the criteria for including or excluding sources from the systematic review. In carrying out the review, an initial selection of primary sources is carried out after examining the title and abstract, in order to exclude primary sources that appear completely irrelevant to the information need of the research question.

A good rule of thumb is to assign at least two researchers to review the search list and keep a record of the selected papers. Selected papers are then to be reviewed against the inclusion/exclusion criteria using the same process used for the abstracts. Reasons for inclusion/exclusion must also be recorded on the spreadsheet.

The third activity of planning involves *validating the review protocol*. It is suggested that reviewers external to the study also be involved in this activity to avoid biases. Validation aims to make sure that all the information extracted relates to the research questions the review intends answering.

Once the protocol is finalized and validated, the next step is to **conduct** the review, i.e. apply all the steps as they have been formally defined. This points out the importance of the first phase in making the process replicable and adoptable by other researchers that may differ from those having defined the protocol itself. Documentation of the steps in this phase is also crucial to keep track of results. This phase includes: *identifying relevant research* through the search strings; *selecting primary studies* according to the inclusion/exclusion criteria; *synthesizing the data* in data extraction tables defined according to the information needs of each research question.

The final phase is to **document** the review. This phase is the conclusive part of the review, important for communicating the results of the study. It can be done in formats such as technical reports, journal or conference papers, as well as non technical articles or web pages.

3 The Hands-On Approach

In this section we will describe how university graduate students have been involved in a systematic review not only by teaching them the theory, but also by allowing them to apply the concepts. The graduate students involved in the study are students at their first year of a MSc degree, all with a BSc degree in informatics or engineering. They all attended the “Empirical SE Methods” course held by the authors of this work. The topic of the review, Statistical Process Control, was part of their course program, so students were all familiar with it. Given the aim of the paper and the space available, we will not go into detail on SPC. Further details on the topic, and on the review results can be found in [4, 9]. We scheduled our classes in order to

train students first, and then receive feedback before assigning them the papers. In this sense our major effort was to introduce them to systematic review. Students participated on a volunteer basis. We defined a schedule similar to the one adopted in occasion of the International Advanced School on Empirical Software Engineering (IASESE 2005), with the difference that we had more time available for training students and receiving feedback before assigning them the reviews. The schedule we followed is commented below and reported in Table 1.

Table 1. Schedule of the systematic review

Lesson 1	Systematic review guidelines & Seminar	We introduced the systematic review methodology according to the guidelines in [17]; B.Kitchenham held a Seminar.
Lesson 2	Experiences of undertaking a systematic review	Some examples of systematic review carried out in literature were presented to the students.
Lesson 3	Revise SPC concepts	A general overview of SPC
Lesson 4	Define search terms, inclusion/exclusion criteria, data extraction forms	Search terms, inclusion/exclusion criteria and data extraction forms were defined according to the research questions and the search motivation provided to students. The task was assigned as homework.
Lesson 5	Discussion	Discussion of proposals. Final validated version of the protocol. The protocol was handed out to students so they could familiarize with all the material for the assignment.
Lesson 6	Search the sources	Students were divided into groups, one for each search source. They selected papers according to the protocol criteria.
Lesson 7	Group work – guided exercise	Students were assigned a paper on SPC and were asked to extract data from the primary source, fill in extraction forms and aggregate data
Lesson 8	Group work – feedback on guided exercise	Correctly completed forms were handed out to students. Obtained results were discussed in groups of 2 and then with the class.
Lesson 9	Assignment of papers	Selected primary sources were assigned to students. They worked individually at home.
Lesson 10	Group work on assigned papers	Students that worked on the same paper individually confronted their data extraction tables and aggregation tables in groups with the other students that worked on the same paper.

First we introduced the students to systematic review and presented the guidelines illustrated in [17]. Barbara Kitchenham also held a seminar on the guidelines (Lesson 1). On the next day (Lesson 2), we illustrated some examples of reviews carried out in literature, supported by published papers and technical reports [8, 18]. Although SPC is part of the students' course program, we thought it was the case to "refresh" their minds on the topic, so we dedicated a lesson (Lesson 3) on the concepts that would appear in the papers to revise. Next, we defined the systematic review protocol on SPC as a class assignment. It is the case to point out that our research group, in collaboration with Barbara Kitchenham, had previously set up a preliminary version of the protocol on this topic. As so, search motivation, research goal and a sketch of the data extraction forms were clear to us. Consequently, we considered the "definition" part of the protocol as a

useful training exercise for the students and a manner for receiving feedback on our behalf. On Lesson 4 we gave students the search motivation and research questions and asked them to identify possible search terms (in class). We assigned the definition of inclusion/exclusion criteria and data extraction forms as homework. Individual work and proposals were then discussed in class during lesson 5. We finally came up with a definitive version of the protocol which was validated by the researchers working on the project. We also provided the list of search sources, i.e. 8 digital libraries for retrieving the papers on SPC.

At this point we presented the complete and validated version of the SPC systematic review protocol and outlined all the details: tables, extraction procedures, inclusion/exclusion and quality assessment criteria. In Lesson 6 students were randomly divided into 8 groups, one for each digital library source, and a person of our research group was assigned to each group as supervisor. Students searched for papers according to the search terms and search strings; read titles and abstracts and adopted inclusion/exclusion criteria to select relevant papers. All decisions were motivated and reported on a spreadsheet. Lists were handed in by each group and results were discussed in class. A total of 129 sources were identified. After excluding duplicates we had a set of 96 relevant titles. Given the number of students in the class (77) we selected a set of 24 papers to review according to the protocol. This initial set of papers was identified from the digital libraries that we had access to as University through an account. Also, we tried to balance the total workload for each student according to the length of papers and type of journals they were retrieved from and made sure that each paper was analyzed by at least 3 students. So, we had cases of students assigned to two papers (i.e. the case of short papers) and cases of papers read by more than 3 students (i.e. the case of long papers). The assignments are summarized in Table 2. Note that they are reported with a paperID. Full references can be found in [4].

Before actually assigning the papers of the review, we decided to carry out a guided exercise (Lesson 7 & Lesson 8). This was important for allowing students to familiarize with the documentation they used in their final assignment.

In the guided exercise students were asked to extract data from either of two primary sources [13, 21] (which we ourselves selected) according to the data extraction form and aggregation tables defined in the SPC protocol. Students were handed the following material: data extraction form, data aggregation tables, research questions of the systematic review, one of the two primary studies. So, for the exercise, half of the students were assigned to a paper [13] and half to another [21]. Students split into pairs and each pair carried out the following tasks: each pair member extracted the data from the paper independently; the pair members compared their data collection forms; any disagreements were solved or noted as disagreements to discuss with the rest of the class and with the researchers. Then, all pairs that had worked on the same paper joined together and completed the aggregation tables. Once the tasks were completed, we handed students the correct complete data extraction forms for the two papers. We had previously analyzed the papers and completed the forms. Results were discussed in class with other groups that had worked on the same paper, and with the researchers. Overall, feedback of the guided exercise was positive. Students became familiar with the data extraction tables. Most of the data they

extracted was correct. Some students found it difficult to understand the meaning of the cells in the tables. Further explanations of the data extraction forms were provided. Also, we decided to translate the cell content in Italian as well, although answers were to be reported in English. Due to language problems, two students decided to not continue with the assignment.

Table 2. Paper Assignment to Students

Paper ID	Nr.Students assigned	Comments
ACM 1	5	--
ACM 2	3	--
CROSSTALK 1	4	--
CROSSTALK 2	4	--
CROSSTALK 3	5	--
CROSSTALK 4	4	--
EMEROTECA1	3	Two papers per person
IEEE 5		
IEEE 10		
IEEE 14	3	Two papers per person
SPRINGER 7		
IEEE 4	3	Two papers per person
IEEE 3		
IEEE 6		
RIF TESI	4	--
SCIENCE DIR 9	3	--
SCIENCE DIRECT 10	4	--
SPRINGER 2	4	--
SPRINGER 3	3	--
SPRINGER 6	4	--
SPRINGER 8	3	--
SPRINGER 1	10	Students worked in couples due to the paper length
IEEE_1	0	Used for guided exercise
IEEE_2	0	Used for guided exercise

Given the positive results of the presentation part and guided exercise, we considered it feasible to continue with our schedule (Lesson 9 & Lesson 10). Students were individually assigned to one of the 24 papers. We ensured that data extraction from each paper was assigned to at least 3 students. Next, all students reviewing the same paper met as checkers and confronted their work, in groups, to obtain a unique version. At this stage a researcher was also assigned as checker, to guarantee that all pairs (and therefore papers) were controlled by an expert; also, conflicts were solved by an adjudicator. A PhD student was also recruited as adjudicator.

Given the focus of the paper, we will not discuss or illustrate the results of the analyses. The reader can refer to [4] for these details.

As it can be seen, the approach adopted has been scheduled in order to allow students to work individually and in groups, discuss and motivate all their decisions. Also, it has developed their ability to systematically search for information focused on the research questions to answer. In this sense, students read paper titles and abstracts critically addressing the questions and the need for evidence. Their opinions on the adopted approach were collected through a questionnaire. Details are reported in the next section.

4 Qualitative Assessment

Once we finished the lessons and carried out the systematic review, students filled in a questionnaire. The questionnaire is reported in the appendix of this paper. As it can be seen, its mere objective was to perceive students' opinions on this experience, given it was the first time we included systematic review as course topic and also the first time that we used a hands-on approach as the one described in the previous section. It is clear that from the students' answers we were able to carry out some type of qualitative assessment, which cannot allow us yet to generalize the collected information. In each case, we consider it an important experience.

The answers followed a general trend of positive impressions. In particular, 95% of our students found the theoretic lessons and examples provided significant for understanding the tasks carried out; 5% considered it significant although requested further details. No one considered them useless.

As for the topic chosen (SPC), 98% of the class agreed that the lessons were useful for understanding and interpreting the concepts of the analyzed papers. In some cases the papers faced issues that had not been discussed or presented in class. This made data extraction more difficult for the students having to review those papers. These comments suggested as improvement (to keep in mind in future courses) the need for us researchers to briefly read through the papers before assigning them to the class.

The data extraction forms were easier to understand as more examples were illustrated. We present some of the most interesting comments:

- "I found the forms easier to understand after a few examples";
- "I understood the data extraction forms after we were given the assignment to define them for our systematic review. The discussion in class with classmates and professors also helped a lot";
- "the data aggregation forms were a demanding task as it requested to combine individual work of different students";
- "knowing that I had to discuss and support my decisions in groups motivated the individual task of data extraction"

As so, students paid more attention to their individual work knowing they were asked to discuss it with other classmates in groups and present the results to the entire class and professors.

Our general impressions on the success of the approach were also confirmed in question 6, i.e. in most cases the individuals confronting their work agreed on

everything. This points out that in their individual tasks they interpreted the information request analogously. In few cases, discussions on minor aspects were necessary.

As it arises from the above opinions and the schedule described in

Table 1, the hands-on approach has given students the chance to apply theoretical concepts through the assignments on a real systematic review and has enforced the importance for searching evidence. A student commented: “This experience taught me that searching for evidence focused at answering a specific research question isn’t as easy as it seems.”

Students’ impressions are that the approach is rigorous and allows to retrieve the necessary information and only focus on the evidence from the research question perspective, i.e. another research question may have classified the same papers as not relevant to the search although always on SPC. Finally, many of them expressed their interest in carrying out another systematic review but on another topic like software product lines or a specific tool.

5 Conclusions

Software engineers as future practitioners are constantly asked to identify the most appropriate technologies, methods and tools to adopt in projects. This paper has focused on the importance of developing skills to allow software engineers to make decisions and evaluations according to the empirical evidence they are able to retrieve in literature. In order to do so, we have introduced a hands-on approach based on systematic review as manner for rigorously searching for evidence, as part of the EBSE process.

In this sense, we have proposed the approach within our university course on “Empirical SE Methods”. As far as we know it is the first attempt to include systematic review as part of a university course on empirical software engineering. We have collected qualitative data on this experience by submitting a questionnaire to students. The considerations presented in the previous section point out the positive attitude of our students towards the assigned tasks.

Overall, we consider the approach as a useful means for making students perceive the importance of searching for evidence in a rigorous and systematic manner such as systematic review. Also, the approach we adopted allowed them to actively participate to the review both individually and in groups confronting their opinions and discussing with pairs.

As researchers we consider systematic review as an essential part of EBSE. Indeed it is only after collecting evidence on a specific issue that decisions can be made. In this sense, in our opinion systematic review represents such rigor. The qualitative data that we have collected from the questionnaires submitted to our graduate course students has in some way confirmed our opinion. Given the results, we have decided to adopt the same strategy in our next course. As future work we are planning to extend our experience and collect evidence on the efficacy and effectiveness of the approach by assigning students to search for evidence with and without systematic review as part of the EBSE process.

References

1. Australian National Health and Medical Research Council.: How to review the evidence: systematic identification and review of the scientific literature (2000) ISBN 186-4960329
2. Australian National Health and Medical Research Council.: How to use the evidence: assessment and application of scientific evidence (February 2000) ISBN 0642432952
3. Bagert, D.J., Jilburn, T.B., Jislop, G., Lutz, M., McCracken, M., Mengel, S.: Guidelines for Software Engineering Education Version 1.0. Technical report, CMU/SEI CMU/SEI-99-TR-032, (1999)
4. Baldassarre, M.T., Caivano, D., Visaggio, G.: Systematic Review of Statistical Process Control: An Experience Report. In: 11th Evaluation and Assessment in Software Engineering Conference, BCS UK, pp.94-102 (April 2007) ISBN:978-1-902505-86-2
5. Bloom, B.S.: Taxonomy of educational objectives: The classification of educational goals: Handbook I, cognitive domain. Longmans Green, New York (1956)
6. Brereton, P., Kitchenham, B., Budgen, D., Turner, M., Khalil, M.: Employing Systematic Literature Review: An Experience Report. Technical Report TR 05/01, School of Computing & Mathematics, Keele University (2005)
7. Cochrane-Collaboration, Cochrane reviews' handbook. Version 4.2.1 (2003)
8. Dyba, T., Kampenes, V.B., Sjøberg, D.: A systematic review of statistical power in software engineering experiments. *Information and Software Technology* 48, 745–755 (2006)
9. Florac, W.A., Carleton, A.D.: Measuring the Software Process: Statistical Process Control for Software Process Improvement. Addison-Wesley, Reading (1999)
10. Glass, R., Vessey, I., Ramesh, V.: Research in software engineering: An analysis of the literature. *Information & Software Technology* 44, 491–506 (2002)
11. Glass, R., Vessey, I., Ramesh, V.: An Analysis of Research in Computing Disciplines. *Communications of the ACM* 47, 89–94 (2004)
12. Hart, C.: Doing a Literature Review: releasing the social science research imagination. SAGE Publications, London (1998)
13. Jacob, A., Pillai, S.K.: Statistical Process Control to Improve Coding and Code Review. *IEEE Software* 50–55 (May/June 2003)
14. Jorgensen, M., Dyba, T., Kitchenham, B.: Teaching Evidence-Based Software Engineering to University Students. In: 11th IEEE International Software Metrics Symposium. IEEE Computer Society Press, Los Alamitos (2005)
15. Jorgensen, M., Shepperd, M.: A Systematic Review of Software Development Cost Estimation Studies. *IEEE Transactions on Software Engineering* 33(1), 33–53 (2007)
16. Kahan, K.S., ter Riet, G., Glanville, J., Sowden, A.J., Kleijnen, J.: Undertaking Systematic Review of Research on Effectiveness. In: CRD's Guidance for those Carrying Out or Commissioning Reviews. CRD's Report Number 4 (2nd edn.), NHS Centre for Reviews and Dissemination, University of York. ISBN 1900640201 (March 2001)
17. Kitchenham, B.: Procedures for Performing Systematic Reviews. Technical Report TR/SE0401, Keele University, and Technical Report 0400011T.1, National ICT Australia (2004)
18. Kitchenham, B., Mendes, E., Travassos, G.: A systematic review of Cross vs. Within company cost estimation studies. In: 10th International Conference on Evaluation and Assessment in Software Engineering, Keele University Staffordshire, UK, April 2006, vol. 3, pp. 79–88 (2006) ISBN 1-902505-74-3
19. Meyer, B.: Software Engineering in the Academy. *IEEE Computer* 34(5), 28–35 (2001)

20. Staples, M., Mahmood, N.: Experiences Using Systematic Review Guidelines. In: 10th International Conference on Evaluation and Assessment in Software Engineering, Keele University, pp.79-88, BCS UK (2006) ISBN 1-902505-74-3

21. Weller, E.: Practical Applications of Statistical Process Control. IEEE Software, 48–55 (2000)

Appendix: Assessment Questionnaire

Paper_ID: _____

1. the theoretic lessons and the examples on systematic review were:
 - a. significant for understanding the tasks to carry out
 - b. of no use. The task we carried out consisted in data extraction that could have been done even without the concepts and examples on systematic review.
 - c. Significant but needed further investigation.
 - d. Other.(specify) _____
2. the training lessons on SPC were:
 - a. useful for understanding and interpreting the contents of the papers
 - b. pointless. The contents of the paper were different than those presented in class
 - c. useful, but more details would have been better.
 - d. Other.(specify): _____
3. Comments on the Data Extraction Forms: _____
4. Comments on the Data Aggregation Forms: _____
5. In the group work, how much was taken from your individual work:
 - a. 20%
 - b. 40%
 - c. 60%
 - d. more than 80%comments: _____
6. In the group work (more than one answer possible)
 - a. We always agreed on everything
 - b. We had to discuss about ____% of times
 - c. In ____ cases we were unable to agree on a decisioncomments: _____
7. would you repeat this experience? YES / NO
motivate your answer: _____
8. your impressions on individual work: _____
9. your impressions on group work: _____

An Empirical Study Identifying High Perceived Value Practices of CMMI Level 2

Mahmood Niazi¹, Muhammad Ali Babar², and Suhaimi Ibrahim³

¹ School of Computing and Mathematics, Keele University, ST5 5BG, UK
mkniazi@cs.keele.ac.uk

² Lero, University of Limerick, Ireland
muhammad.alibabar@ul.ie

³ Centre for Advanced Software Engineering, University Technology Malaysia,
Jalan Semarak, 54100 Kuala Lumpur, Malaysia
suhaimiibrahim@utm.my

Abstract. We have conducted face-to-face questionnaire based interview sessions with twenty-three Malaysian software practitioners in order to determine the perceived value associated with the specific practices of “requirements management”, “process and product quality assurance” and “configuration management” process areas of CMMI level 2 in the stage representation. The objective of this study is to identify the extent to which a CMMI practice is used in order to develop a finer-grained framework, which encompasses the notion of perceived value within specific practices. This will provide software process improvement (SPI) practitioners with some insight into designing appropriate SPI implementation strategies.

We asked practitioners to choose and rank “requirements management”, “process and product quality assurance” and “configuration management” practices against the five types of assessments (high, medium, low, zero or do not know). From this, we propose the notion of ‘perceived value’ associated with each practice. We have identified ‘high’ and ‘medium’ perceived values CMMI level 2 practices. We have also identified the viewpoints of developers and managers about these practices.

1 Introduction

Software Process Improvement (SPI) has been a long-standing approach promoted by software engineering researchers, intended to help organisations develop higher-quality software more efficiently. Process capability maturity models such as CMM, CMMI [1] and ISO/IEC 15504 (SPICE) are SPI frameworks for defining and measuring processes and practices that can be used by software developing organisations. However, the population of organisations that have adopted process capability maturity model SPI is only a part of the entire population of software-developing organisations. CMMI is the successor to CMM and is consistent with the international standard ISO/IEC 15504. The most well-known representation of CMMI is the “staged” representation, which has five “levels” of process maturity for organisations. However, a common concern about CMM, CMMI, and related approaches is their

relevance and applicability for small organisations [2]. Case studies reporting small organisations' experience with CMM [3] invariably discuss the peculiar difficulties that small organisations have of using and benefiting from CMM, and attempts have been made to provide guidance about using tailored CMM for small organisations [4].

There has been a call to understand business drivers for SPI "...to make SPI methods and technologies more ... widely used" [5]. Moreover, there has also been an increasing emphasis on identifying and understanding the relative "perceived value" of different SPI practices and factors [6]. A better understanding of the relative value of SPI practices perceived by organisations and practitioners is expected to enable SPI program managers to concentrate more on "high perceived value" practices. Previously, researchers have also reported the relative "perceived value" of CMMI practices with the aim of helping practitioners to pay more attention to the "high perceived value" practices [6].

Our research is aimed at extending the findings of Wilkie et al. [6] by conducting a similar study in a different culture. However, we decided to identify the relative "perceived value" of SPI practices based on practitioners' perception rather than based on process appraisal like reported by Wilkie et al. in [6]. We believe that software practitioners may associate different values to different SPI practices and the relative value of a practice may encourage or discourage them from fully supporting a particular practice. As part of a large project on SPI, we have been empirically studying different aspects of the SPI programs in the Asian region [7; 8] as this region has been attracting significant number of software outsourcing contracts from Western countries including Ireland, where Wilkie et al. carried out their study. The results of this project are expected to help software practitioners from vendor organisations (i.e., usually Asian software development houses) and client organisations (i.e., usually Western software development outsourcers) to understand the human related aspects of SPI in order to design better SPI implementation strategies.

This paper presents results of an empirical study aimed at identifying and understanding the relative "perceived value" of CMMI level 2 practices based on the perception of practitioners in a developing country, Malaysia, involved in outsourced software development. The findings of the reported study identify those CMMI level 2 practices (requirements management, configuration management and process and product quality assurance), which are perceived to have "high value" by Malaysian practitioners.

We have encountered several interesting findings which enabled us to identify and explain the relative "perceived value" of different practices required to achieve the CMMI level 2 process maturity. We have also identified a set of research questions that need to be explored in this line of research. Since a theory explaining the attitude and behaviour of practitioners toward different aspects of SPI programs does not exist, this study employs an inductive approach (i.e., using facts to develop general conclusions) as an attempt to move toward such a theory.

The paper makes the following contributions to the SPI discipline:

- It presents the design and results of a first of its kind study in a developing country to understand an important aspect of CMMI-based SPI practices.
- It provides information about what practitioners think about the value of different SPI practices for three of the CMMI level 2 process areas.

It identifies further research areas that need to be explored to support an effective and successful SPI program.

The following Section explains the concept of perceived value. Section 3 describes the study design and logistics. Sections 4 presents and discusses findings based on frequency analysis of the gathered data. Limitations of the study are described in Section 5. The paper finishes in Section 6 with summary and future work.

2 Perceived Value

In this study, we define ‘perceived value’ to mean the extent to which a SPI practice is perceived to add value to a project or an organisation based on the perceptions of practitioners who have been working in the area of SPI in their respective organisations. This may be considered to be a subjective view as it relies on the self-reported data. However, the respondents of this study are considered to be SPI experts within their organisations. Hence, we are confident that their opinion is grounded in significant experience of real world SPI initiatives.

In order to describe the notion of perceived value of SPI practices, it is important to decide on the “criticality” of a perceived value. For this purpose, we have used the following definition:

- If the majority of respondents ($\geq 50\%$) perceive that a SPI practice has high value then we treat that practice as critical.

A similar approach has been used in the literature [9; 10]. Rainer and Hall [9] identified important factors in SPI with the criterion that if the 50% or more participants perceive that a factor has a major role in software process improvement efforts then that factor should be treated as having a major impact on SPI. In the highly competitive industry of software development, it is becoming increasingly difficult for managers to make “value neutral” decisions. Like Wilkie et al. [6], we assert that “perceived value” of a particular practice can be used as a judgement criterion for determining activities that organisations need to pursue. We believe that where respondents from different organisations identify a practice as having a high-perceived value then that practice should be considered for its importance in a process improvement program. The information about relative “perceived value” can help practitioners and researchers to better understand various practices detailed within the CMMI for the purpose of developing more appropriate implementation strategies and appraisal procedures for small-to-medium sized organisations.

3 Study Design

We used face-to-face questionnaire based interview sessions as our main approach to collect data from twenty-three software development practitioners of twenty-three Malaysian software development organisations. Appendix A shows the demographics of participants’ organisations. The data was collected from practitioners who were involved in tackling real SPI implementation issues on a daily basis in their respective organisations. It is important to acknowledge that the practitioners sampled within

organisations are representative of practitioners in organisations as a whole. A truly representative sample is impossible to attain and the researcher should try to remove as much of the sample bias as possible [11]. In order to make the sample fairly representative of SPI practitioners in particular organisation, different groups of practitioners from each organisation were selected to participate in this research. The sample of practitioners involved in this research includes developers, quality analysts, Software Quality Analysis (SQA) team leaders, SQA managers, project managers, and senior management. Thus the sample is not random but a convenience sample, because we sought a response from a person with a specific role within a software development organisation. We consider that the practitioners who participated in this study fall into two main categories:

- “Developers” consisting of programmer/ analyst/ SQA coordinator.
- “Managers” consisting of team leader/ project manager, and senior managers.

We used a closed ended questionnaire as an instrument to collect self-reported data. In order to describe the importance of a SPI practice, the respondents were supposed to mention each identified practice’s relative value (i.e., High value, Medium value, Low value, Zero value, or Not sure).

In order to analyse the perceived value of each identified SPI practice, the occurrence of a perceived value (high, medium, low, zero) in each questionnaire was counted. By comparing the occurrences of one SPI practice’s perceived values obtained against the occurrences of other SPI practices’ perceived values, the relative importance of each SPI practice has been identified.

The responses to the questionnaire were gathered during July 2006. Though all the participants were well-versed in English and the questionnaire was in English, the research team had a Malaysian speaking researcher, who provided necessary interpretation and explanation whenever required.

4 Findings

The questionnaire was designed to gather data about the “perceived value” of practices of six of the seven CMMI maturity level 2 process areas: Requirements Management, Configuration Management, Project planning, Project Monitoring & Control, Measurement & Analysis and Process & Product Quality Assurance. However, because of space limitations, this paper reports the results about the practices of three process areas: Requirements Management, Process & Product Quality Assurance, and Configuration Management.

There were 23 participants representing 23 small-to-medium sized Malaysian software development companies. Fourteen of the companies were subsidiaries of the Multi-National organisation, while remaining 10 companies were national. Majority of the companies were developing software for more than five years, while three companies were less than five year old. Thirteen of the participants were working for small companies (0 to => 19 employees) and other ten worked for medium sized companies (20 to => 199 employees). We used this categorization based on the organisation size definition provided by the Australian Bureau of Statistics [12].

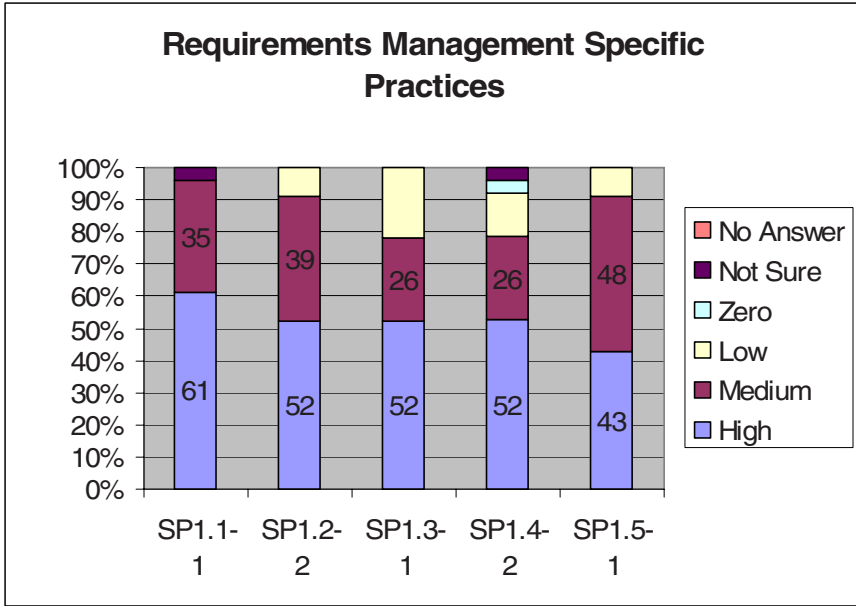
4.1 Requirements Management Practices

Figure 1 presents the CMMI level 2 specific practices for requirements management. The results show that the most common 'high' value practice (61%) is 'obtain an understanding of requirements'. Research shows that if a system's requirements are not fully understood, it can have a huge impact on the effectiveness of the software development process for that system [13-15]. When requirements are poorly defined, the end result is usually a poorer product or a cancelled project [14]. An industry survey in the UK reported that only 16% of software projects could be considered truly successful: "Projects are often poorly defined, codes of practice are frequently ignored and there is a woeful inability to learn from past experience" [16]. The evidence is clear: problems in the requirements phase can have a large impact on the success of software development projects [15; 17] and this is a lesson that continues to be usually ignored despite the evidence and the low amount of effort needed to have a reasonable requirements process. Majority of the participants of this study perceive this practice of "high value", which shows that they appear to be well aware of the critical importance of this practice in a SPI initiative

The requirements management practice 'obtain commitment to requirements' is also a frequently reported 'high value' practice in Malaysia. That means majority of the participants of our study are aware of the importance of obtaining commitments from the project participants. One can obtain commitment to the requirements from the project participants by involving the stakeholders in the systems development process [18; 19]. Involving stakeholders in the development process can reduce their fear, for example, that development of a software system will result in loss of jobs. It is also possible that if a new system is installed in an organisation without consulting the stakeholders, who would be affected by the system, then they may feel that a new system is unnecessary and therefore they should not co-operate in its specification.

The other frequently cited 'high value' practices are 'manage requirements changes' and 'maintain bidirectional traceability requirements'. Our results have confirmed the previous findings of several accounts that describe the importance of these requirements practices [15; 20-23]. It is widely reported that requirements often change during the software/system development process. These changes are inevitable and driven by several factors including errors in original requirements, evolving customer needs, constant changes in software and system requirements, business goals, work environment and government regulation [24]. Volatile requirements are regarded as a factor that cause major difficulties during system development for most organisations in the software industry [25]. Volatile requirements contribute to the problems of software project schedule overruns and may ultimately contribute to software project failure [25-27]. Furthermore, ad hoc change management can lead to escalating requirements, uncontrolled scope creep and potentially uncontrolled system development [13; 28]. Requirements traceability is also cited as a 'high value' practice by the Malaysian practitioners. Traceability of requirements is, in our opinion, one of the most important parts of requirements management process. It will simply be impossible to manage requirements relationships except for perhaps in the most simple of projects without any traceability mechanisms agreed and implemented.

Using the criterion described in Section 2, we have identified four, frequently cited (>50%), ‘requirements management’ practices (SP1.1-1, SP1.2-2, SP1.3-1, SP1.4-2) as having a ‘high’ perceived value.



CMMI Practice Number	CMMI Practice Description
SP1.1-1	Obtain an understanding of requirements
SP1.2-2	Obtain commitment to requirements
SP1.3-1	Manage requirements changes
SP1.4-2	Maintain bidirectional traceability requirements
SP1.5-1	Identify inconsistencies between project work and requirements

Fig. 1. Requirements Management Practices

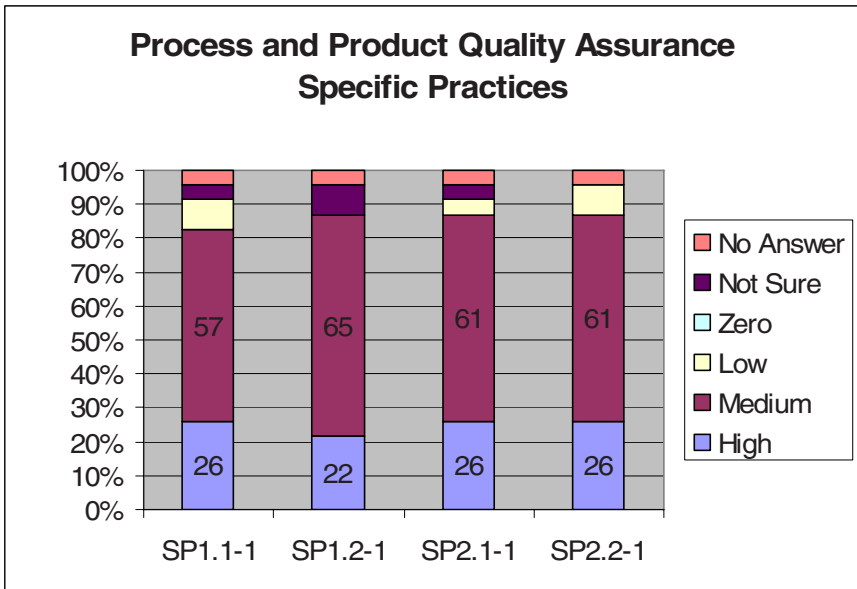
4.2 Process and Product Quality Assurance Practices

There are four specific practices in ‘process and product quality assurance’ process area as shown in Figure 2. It is interesting to find that none of the practice of this process area was perceived as “high value” practice. Less than 30% of the participants cited all practices of ‘process and product quality assurance’ as ‘high value’ practices. These findings may lead someone to conclude that very limited attention is being paid to process and quality assurance activities in Malaysia. Given that being able to demonstrate process and product quality is one of the main factors for software vendor companies to achieve certain CMMI maturity level, it is quite interesting revelation that majority of our study’s participants do not place high value on the practices designed for ensuring process and product quality. If this situation is prevalent in

Malaysian software development industry, it is likely to have negative impact on software development economy of the country as many firms from Western countries (such as USA and UK) are outsourcing software development projects to Asian software development houses [29]. Process and product quality assurance is very important for companies that either develop software or buy software. This is because the process and product quality assurance is the activity which provides evidence that the methods and techniques used are integrated, consistent and correctly applied [30].

However, the bar chart in Figure 2 shows that more than half of the Malaysian practitioners perceived all practices of ‘process and product quality assurance’ process area as having ‘medium value’. This finding is quite encouraging as it shows that a large majority of the study’s participants is aware of the importance of all the practices in this process area, albeit a majority of them did not place those practices on the top of their priority list. It will be interesting to find out the detailed reasons for this situation. This finding may also be considered as an indicator of an increasing realization of the critical role of the practices of this process area in achieving CMMI maturity level 2.

However, based on these findings and using the criterion described in Section 2, this study has not identified any frequently cited (>50%), ‘process and product quality assurance’ practice which has a ‘high’ perceived value.

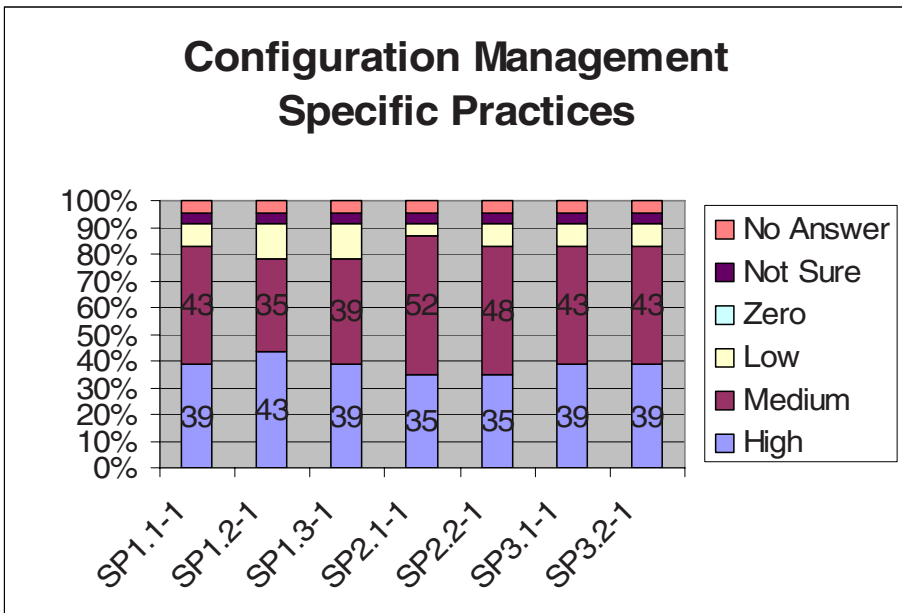


CMMI Practice Number	CMMI Practice Description
SP1.1-1	Objectively evaluate processes
SP1.2-1	Objectively evaluate work products and services
SP2.1-1	Communicate and ensure resolution of non-compliance issues
SP2.2-1	Establish records

Fig. 2. Process and Product Quality Assurance Practices

4.3 Configuration Management Practices

Figure 3 shows that there are total 7 specific practices in the Configuration management process area. Figure 3 also presents the percentages of the participants’ responses about the relative ‘value’ of each of the specific practices of this process area. It is clear that like all specific practices of the process and product quality assurance process area, none of the specific practices of this process area has been singled out as having ‘high value’ by majority of the participants of this study. However, a large majority of the participants perceived that each of the practices of this process area has either ‘high’ or ‘medium’ value in their process improvement program. That means there were only a small number of participants who perceived that the practices of this process area were of ‘Low value’



CMMI Practice Number	CMMI Practice Description
SP1.1-1	Identify the configuration items
SP1.2-1	Establish a configuration management system
SP1.3-1	Create or release baselines
SP2.1-1	Track change requests
SP2.2-1	Control configuration items
SP3.1-1	Establish configuration management records
SP3.2-1	Perform configuration audits

Fig. 3. Configuration Management Practices

We do not believe that such findings can be considered as a problem specific to Malaysia as many companies in Western countries with relatively longer history of software development have been found not being able to adequately control their configuration items [6]. Hence, we contend that with the passage of time, the importance of the configuration management practices will also grow among Malaysian practitioners as they will learn from their Westerns colleagues or such practices will be mandated by their clients. Another reason for not having any frequently cited 'high value' configuration management practice could be the background and organisational roles of the participants of this study. There were 49% developers and 51% managers in this study. Since process related configuration management practices are usually considered a manager's responsibility that may be the reasons that none of the practices was perceived of 'high value' by more than 50% of the participants. This explanation of the finding get more support when we take into account the fact that 50% of the managers participants (see section 4.4) perceived four 'configuration management' practices (SP1.2-1, SP1.3-1, SP3.1-1, SP3.2-1) as having 'high value'.

However, more than half of the Malaysian practitioners have cited only one practice - 'track change requests for the configuration items' - as a 'medium' value practice. All other practices are also low cited 'medium' value practices. Hence, based on the criterion described in Section 2, we have not identified any practice in this process area that has been perceived of 'high value' by more than 50% of the participants of our study.

4.4 Comparing Developers' and Managers' Perceptions

We have also decided to analyse the findings based on the participants' roles in their organisations, i.e., developer and manager. We contend that an understanding of the similarities and differences found among practitioners' perceptions about the relative value of the CMMI practices can help managers to design better SPI strategies as they can place more focus on the practices that are considered of 'high value' by practitioners across different roles. We believe that when respondents from different groups of practitioners consider a practice of 'high value', that practice tends to have significant impact on the success of a SPI program. Tables 1-3 show the relative 'perceived value' of the CMMI practices reported by developers and managers.

Table 1. Requirements Management practices identified by developers and managers

Requirements Management Practices	Developers (n=11)					Managers (n=12)				
	H	M	L	Z	NS/ NR	H	M	L	Z	NS/ NR
SP1.1-1	6	4	0	0	1	8	4	0	0	0
SP1.2-2	4	6	1	0	0	8	3	1	0	0
SP1.3-1	4	5	2	0	0	8	1	3	0	0
SP1.4-2	5	4	0	1	1	7	2	3	0	0
SP1.5-1	4	7	0	0	0	6	4	2	0	0

H=High, M=Medium, L=Low, Z= Zero, NS/ NR=Not sure/ No response

Table 1 shows the list of requirements management practices along with their respective relative 'value' cited by developers and managers. It is evident from Table 1 that only one practice 'Obtain an understanding of requirements' considered as having

‘high value’ by majority of the developers (6 out of 11) and managers (8 out of 12). ‘Obtain commitment to requirements’ is the ‘high’ value practice frequently cited by the Malaysian managers while the Malaysian developers consider this practice as a ‘medium’ value practice. This shows that the Malaysian managers give importance to stakeholders’ participation in the systems development process as this is one of the motivations to obtain commitment to requirements from project participants [18; 19]. More than 50% of the Malaysian managers consider ‘manage requirements changes’, ‘maintain bidirectional traceability requirements’ and ‘identify inconsistencies between project work and requirements’ as ‘high’ value practices. This results show that more than 50% of the Malaysian managers have cited all specific practices of the requirements management process area as having ‘high value’. However, only one practice, (SP1.1-1) of this process area has been reported as having ‘high value’ by more than 50% of the Malaysian developers.

Table 2. Process and Quality Assurance practices identified by developers and managers

Process and Product Quality Assurance practices	Developers (n=11)					Managers (n=12)				
	H	M	L	Z	NS/ NR	H	M	L	Z	NS/ NR
SP1.1-1	2	5	2	0	2	4	8	0	0	0
SP1.2-1	1	7	0	0	3	4	8	0	0	0
SP2.1-1	1	7	1	0	2	5	7	3	0	0
SP2.2-1	1	7	2	0	1	5	7	0	0	0

H=High, M=Medium, L=Low, Z= Zero, NS/ NR=Not sure/ No response

Table 2 presents the relative ‘value’ of each of the practices of the process and quality assurance process area reported by developers and managers. It is interesting to note that no practice in this process area has been frequently cited ($\geq 50\%$) as a ‘high’ perceived value. However, majority of the Malaysian developers and managers consider all practices as a ‘medium’ value practices except for a practice ‘objectively evaluate processes’ which is only considered by Malaysian managers as a ‘medium’ perceived value practice. These results again reveal that none of the practice of this process area has been perceived as having ‘high value’ by more than 50% of the participants from either group. We have already provided some of the possible explanations for such situation in Section 4.2.

Table 3. Configuration Management practices identified by developers and managers

Configuration Management practices	Developers (n=11)					Managers (n=12)				
	H	M	L	Z	NS/ NR	H	M	L	Z	NS/ NR
SP1.1-1	4	5	0	0	2	5	5	2	0	0
SP1.2-1	4	4	1	0	2	6	4	2	0	0
SP1.3-1	3	6	0	0	2	6	3	3	0	0
SP2.1-1	3	6	0	0	2	5	6	1	0	0
SP2.2-1	3	5	12	0	2	5	6	1	0	0
SP3.1-1	3	6	0	0	2	6	4	2	0	0
SP3.2-1	3	6	0	0	2	6	4	2	0	0

H=High, M=Medium, L=Low, Z= Zero, NS/ NR=Not sure/ No response

Table 3 shows the specific practices of the configuration management process area along with their respective perceived 'value' as cited by the Malaysian developers and managers. Our results show that no specific practice of this process area has been frequently ($\geq 50\%$) cited by Malaysian developer as being 'high' value practice. We argue that this is due to the fact that 'configuration management' is often considered the responsibility of managers in any organisation. However, 50% of the Malaysian managers consider 'establish a configuration management system', 'create or release baselines', 'establish configuration management records' and 'perform configuration audits' as 'high value' practice of the configuration management process area. Using the criterion described in Section 2, we can say that four 'configuration management' practices (SP1.2-1, SP1.3-1, SP3.1-1, SP3.2-1) can be considered as having 'high perceived value' for Malaysian managers. However, none of the practice can be considered as having 'high perceived value' for the Malaysian developers based on their responses.

5 Limitations

Construct validity is concerned with whether or not the measurement scales represent the attributes being measured. Our interview instrument was based on the specific practices of the six process areas of CMMI maturity level 2 [1]. During the interviews, the researchers observed that the participants were able to recognize each of the practice without any difficulty. Hence, their responses provided us with the confidence that all the practices included in the interview instrument were relevant to the participants' workspace. External validity is concerned with the generalisation of the results to other environments than the one in which the initial study was conducted. External validity was examined by conducting survey with 23 practitioners from 23 different organisations.

Another issue is that the interview surveys are usually based on self-reported data that reflects what people say they believe or do, not necessarily what they actually believe or practice. Hence, our results are limited to the respondents' knowledge, attitudes, and beliefs regarding the relative 'value' of each of the specific practices of different process areas of CMMI maturity level 2 through semi-structured interviews. However, the interviewees' perceptions have not been directly verified through another mechanism such as appraising their organisational practices like Wilkie et al. [6]. This situation can cause at least one problem – practitioners' perceptions may not be fully accurate as the relative 'values' assigned to different practices may actually be different. However, like the researchers of many studies based on opinion data (e.g. [31; 32]), we also have full confidence in our findings because we have collected data from practitioners who were directly involved in SPI efforts in their respective organisations and their perceptions were explored without any direction from the researchers.

Sample size may be another issue as we could interview only 23 practitioners from 23 Malaysian companies. In this respect, our research was limited by available resources and the number of companies that could be convinced to participate in the reported study. However, to gain a broader representation of Malaysian practitioners' views on this topic, more practitioners and companies need to be included in a study.

Another limitation of the study is the non-existence of a proven theory about human and organisational aspects of SPI efforts to guide a research similar to ours. That is why we considered our research as an exploratory approach, aimed at gathering facts in the hopes of drawing some general conclusions. We expect that the findings from this research can help us and SPI community to identify research direction to develop and validate a theory of mechanics of SPI based on a certain maturity model.

6 Summary and Conclusion

Our research has been motivated by the challenges faced by companies, especially SMEs, in implementing CMMI based process improvement programs. Moreover, there have also been calls to understand business drivers and relative ‘perceived value’ of different practices of a process improvement model (Such as CMMI) in order to make SPI methods and technologies widely used [5]. We believe that a better understanding of the relative value of SPI practices perceived by practitioners should be taken into consideration while designing and implementing a SPI program.

Other researchers have also conducted a study to identify the relative value of the each of the practices of CMMI level 2 process areas with the aim of helping practitioners to pay more attention to the “high perceived value” practices [6]. The relative “perceived value” of CMMI practices can act as a guide for SPI program managers when designing and implementing SPI initiatives. The assertion behind this argument is that it will be easier to encourage the use of those SPI practices that are commonly used elsewhere and known to be perceived of high value by practitioners.

This paper reports an empirical study aimed at identifying and understanding the relative ‘value’ of different practices of CMMI level 2 process areas. Our results are based on the analysis of self-reported data gathered to explore the experiences, opinions, and views of Malaysian software development practitioners. In order to describe the notion of perceived value of SPI practices, it is important to decide on the “criticality” of a perceived value. For this purpose, we used the following definition:

- If the majority of respondents ($\geq 50\%$) perceive that a SPI practice has high value then we treat that practice as critical.

Based on this criterion and perceptions of all Malaysian practitioners:

- We have identified four ‘requirements management’ practices (SP1.1-1, SP1.2-2, SP1.3-1, and SP1.4-2) as having a ‘high’ perceived value.
- Our study has not identified any frequently cited ‘process and product quality assurance’ practice which has a ‘high’ perceived value.
- We have not identified any practice in ‘configuration management’ process area which has a ‘high’ perceived value.

Moreover, we have also found that:

- All specific practices of the ‘requirements management’ process area have been reported as ‘high value’ practices by Malaysian managers. However, only one practice, (SP1.1-1) of this process area has been reported as ‘high value’ by Malaysian developers.
- None of the practice of ‘process and product quality assurance’ process area has been perceived as ‘high value’ by Malaysian developers and Malaysian managers.

- Four ‘configuration management’ practices (SP1.2-1, SP1.3-1, SP3.1-1, and SP3.2-1) can be considered as having ‘high perceived value’ for Malaysian managers. However, none of the practice can be considered as having ‘high perceived value’ for the Malaysian developers.

Our long-term research goal is to build an empirically tested body of knowledge of different aspects of SPI initiatives and assessment. We are approaching this by firstly focusing on complementing and/or extending the current understanding about practitioners’ attitudes toward and opinions of different aspects of software process improvement models and programs. We plan to develop appropriate support mechanisms and tools to facilitate the design and implementation of suitable SPI strategies. In this study, we have gained important insights into the relative “perceived value” of each practice of the three process areas of the CMMI level 2 maturity. We found that practitioners view certain practices are of “high value” and should be paid more attention in any SPI program initiative. From the findings of this study, we have identified following goals that we plan to follow in future:

- Collect additional data on the perceived value of different SPI practices by exploring the basis of practitioners’ perceptions through more in-depth interviews and case studies.
- Conduct empirical studies to determine the relationship between the relative “perceived value” of each practice and how its implementation is justified by return on investment.
- It is also important to determine the mechanics of encouraging practitioners to support those practices which have been perceived of “low value” but are required to achieve a certain level of process maturity.

Acknowledgements

We are grateful to the participants and their companies for participating in this study. Lero is funded by Science Foundation Ireland under grant number 03/CE2/I303-1.

This research is also funded by Science Foundation, vot no. 79276 (UTM-RMC) under the Malaysian Ministry of Science, Technology and Innovation (MOSTI).

References

1. Chrissis, M., Konrad, M., Shrum, S.: CMMI Guidelines for Process Integration and Product Improvement. Addison-Wesley, Reading (2003)
2. Brodman, J.G., Johnson, D.L.: What Small Businesses and Small Organizations Say About the CMMI. In: Proceedings of 16th International Conference on Software Engineering (ICSE 1994). IEEE Computer Society Press, Los Alamitos (1994)
3. Batista, J., Dias, d.F.: Software Process Improvement in a Very Small Team: a Case with CMM. *Software Process-Improvement and Practice* (5), 243–250 (2000)
4. Paulk, M.: Using the Software CMM in small organizations. In: The Joint 1998 Proceedings of the Pacific Northwest Software Quality Conference and the Eighth International Conference on Software Quality, Portland, pp. 350–361 (1998)

5. Conradi, R., Fuggetta, A.: Improving Software Process Improvement, July/August (2002), pp. 92–99. *IEEE Software* (2002)
6. Wilkie, F.G., McFall, D., McCaffery, F.: An Evaluation of CMMI Process Areas for Small to Medium-sized Software Development Organisations. *SOFTWARE PROCESS IMPROVEMENT AND PRACTICE* 10, 189–201 (2005)
7. Niazi, M., Babar, M.: Ali: De-motivators for software process improvement: An Analysis of Vietnamese Practitioners' Views. In: Münch, J., Abrahamsson, P. (eds.) *PROFES 2007*. LNCS, vol. 4589, pp. 118–131. Springer, Heidelberg (2007)
8. Niazi, M., Babar, M.: Ali: Motivators of Software Process Improvement: An Analysis of Vietnamese Practitioners' Views. In: *International Conference on Evaluation and Assessment in Software Engineering (EASE 2007)*, pp. 79–88 (2007)
9. Rainer, A., Hall, T.: Key success factors for implementing software process improvement: a maturity-based analysis. *Journal of Systems & Software* 62(2), 71–84 (2002)
10. Niazi, M., Wilson, D., Zowghi, D.: A Maturity Model for the Implementation of Software Process Improvement: An empirical study. *Journal of Systems and Software* 74(2), 155–172 (2005)
11. Coolican, H.: *Research Methods and Statistics in Psychology*. Hodder and Stoughton, London (1999)
12. Trewin and D: *Small Business in Australia: 2001*. Australian Bureau of Statistics report 1321.0 (2002)
13. El Emam, K., Madhavji, H.N.: A Field Study of Requirements Engineering Practices in Information Systems Development. In: *Second International Symposium on Requirements Engineering*, pp. 68–80 (1995)
14. Standish-Group: *Chaos: A Recipe for Success*. Standish Group International (1999)
15. Hall, T., Beecham, S., Rainer, A.: Requirements Problems in Twelve Software Companies: An Empirical Analysis. In: *IEE Proceedings - Software*, August 2002, pp. 153–160 (2002)
16. *Jobserve.com: UK Wasting Billions on IT Projects (21/4/2004)*,
<http://www.jobserve.com/news/NewsStory.asp?e=e&SID=SID2598>
17. Sommerville, I.: *Software Engineering*, 5th edn. Addison-Wesley, Reading (1996)
18. Rauterberg, M., Strohm, O.: About the Benefits of User-Oriented Requirements Engineering. In: *Proceedings of the First International Workshop on Requirements Engineering: Foundation of Software Quality (REFSQ 1994)* (1994)
19. DeBillis, M., Haapala, C.: User-Centric Software Engineering. *IEEE Expert* 10(1), 34–41 (1995)
20. Niazi, M.: An empirical study for the improvement of requirements engineering process. In: *The 17th International Conference on Software Engineering and Knowledge Engineering*, Taiwan, Republic of China, July 14-16, 2005, pp. 396–399 (2005)
21. Niazi, M., Shastry, S.: Role of Requirements Engineering in Software development Process: An empirical study. In: *IEEE International Multi-Topic Conference (INMIC 2003)*, pp. 402–407 (2003)
22. Niazi, M., Cox, K., Verner, J.: An empirical study identifying high perceived value requirements engineering practices. In: *Fourteenth International Conference on Information Systems Development (ISD 2005)*, Karlstad University, Sweden, August 15-17 (2005)
23. Sommerville, I., Ransom, J.: An empirical study of industrial requirements engineering process assessment and improvement. *ACM Transactions on Software Engineering and Methodology* 14(1), 85–117 (2005)
24. Barry, E.J., Mukhopadhyay, T., Slaughter, S.A.: Software Project Duration and Effort: An Empirical Study. *Information Technology and Management* 3(1-2), 113–136 (2002)

25. Zowghi, D., Nurmuliani, N.: A study of the impact of requirements volatility on software project performance. In: Ninth Asia-Pacific Software Engineering Conference, pp. 3–11 (2002)
26. Stark, G., Skillicorn, A., Ameen, R.: An Examination of the Effects of Requirements Changes on Software Maintenance Releases. *Journal of Software Maintenance: Research and Practice* 11, 293–309 (1999)
27. Zowghi, D., Nurmuliani, N., Powell, S.: The Impact of Requirements Volatility on Software Development Lifecycle. In: Proceedings of Software Engineering Conference, Australian, pp. 28–37 (2004)
28. Verner, J., Evanco, W.M.: In-house Software Development: What Software Project Management Practices Lead to Success? *IEEE Software* 22(1), 86–93 (2005)
29. Kobitzsch, W., Rombach, D., Feldmann, R.L.: Outsourcing in India. *IEEE Software*, 78–86 (2001)
30. Jarvis, A., Crandall, V.: *INROADS to software quality*. Prentice-Hall, Inc., Englewood Cliffs (1997)
31. Beecham, S., Hall, T., Rainer, A.: Software Process Problems in Twelve Software Companies: An Empirical Analysis. *Empirical software engineering* 8, 7–42 (2003)
32. Niazi, M., Wilson, D., Zowghi, D.: A Framework for Assisting the Design of Effective Software Process Improvement Implementation Strategies. *Journal of Systems and Software* 78(2), 204–222 (2005)

Appendix A: Demographics

ID	Company Age (yrs)	Size	Primary function
1	>5	20-199	In-house development
2	>5	20-199	In-house development
3	>5	20-199	In-house and outsourced development
4	3-5	20-199	In-house and outsourced development
5	>5	20-199	IT service provider
6	>5	20-199	In-house development
7	>5	20-199	In-house and outsourced development
8	3-5	20-199	In-house and outsourced development
9	>5	20-199	In-house development
10	>5	20-199	In-house development
11	>5	20-199	In-house development
12	>5	20-199	In-house development
13	>5	<20	In-house development
14	3-5	<20	In-house and outsourced development
15	>5	20-199	Outsourced development
16	>5	20-199	In-house development
17	>5	20-199	In-house and outsourced development
18	>5	20-199	Other
19	>5	20-199	In-house development
20	>5	20-199	In-house development
21	>5	20-199	In-house development
22	>5	20-199	In-house development
23	>5	20-199	In-house development

2nd International Workshop on Measurement-Based Cockpits for Distributed Software and Systems Engineering Projects (SOFTPIT 2008)

Marcus Ciolkowski¹, Jens Heidrich¹, Marco Kuhrmann², and Jürgen Münch¹

¹ Fraunhofer IESE, Fraunhofer Platz 1, 67663 Kaiserslautern, Germany
{marcus.ciolkowski, jens.heidrich,
juergen.muench}@iese.fraunhofer.de

² Technische Universität München, Arcisstrasse 21, 80333 München, Germany
kuhrmann@in.tum.de

In order to successfully conduct global development projects, one crucial success factor is the existence of well-specified and coordinated distributed development processes. Therefore, it is necessary to have efficient management and controlling mechanisms in place. Many companies are currently establishing so-called software cockpits for systematic quality assurance and management support. A software cockpit is comparable to an aircraft cockpit, which centrally integrates all relevant information for monitoring, controlling, and management purposes. In practice, a variety of simple dashboards approaches for project control exists. However, approaches supporting advanced management techniques and allowing for organization-wide data collection, interpretation, and visualization are rarely implemented. The goal of this workshop is to discuss techniques, methods, and tools for the measurement-based management of globally distributed software development projects and to share experiences among researchers and practitioners. Efficiently managing distributed development projects (multi-sited and multi-organizational) implies many challenges that need to be addressed by software engineering research. Typical research questions address the problems of how to cope with different cultures or heterogeneous organizational structures, platforms, measurement systems, and process maturities. Other research areas include the establishment of data security, transparency, confidence in integrated data and interpretation models, and well-coordinated development processes and interfaces.

The workshop will discuss techniques, methods, and tools that support the effective management of distributed development projects. Topics of interest include, but are not limited to: data-driven management of distributed development projects, strategies for distributed management and controlling, key performance indicators and success factors for managing distributed projects, introduction and application of distributed control mechanisms, cultural, technical, social, and organizational issues for distributed project management, efficient mechanisms for aggregation and drill-down of distributed data, innovative visualization mechanisms and stakeholder-specific views for distributed control centers, goal-oriented measurement, and process-/tool integration and harmonization in distributed environments.



10th International Workshop on: Learning Software Organizations –Methods, Tools, and Experiences–

Raimund L. Feldmann¹ and Martin Wessner²

(Workshop Chairs)

¹ Fraunhofer Center for Experimental Software Engineering, Maryland (FC-MD)
4321 Hartwick Road, Suite 500, College Park, MD 20742-3290, USA

r.feldmann@fc-md.umd.edu

² Fraunhofer Institute for Experimental Software Engineering, (IESE)
Fraunhofer-Platz 1, 67663 Kaiserslautern, Germany

martin.wessner@iese.fraunhofer.de

Abstract. Ten years ago, the LSO workshop series was designed as a communication forum that addresses the questions of organizational learning from a software point of view. Today this topic is still current and in high demand. The workshop focuses on existing work on knowledge management and organizational learning as well as on newer developments such as Web 2.0 or Enterprise 2.0. It brings together practitioners and researchers for an open exchange of experience with successes and failures in organizational learning and provides an opportunity to learn about new ideas. Fostering interdisciplinary approaches is one key concern of this workshop series.

Keywords: Continuous Process Improvement, Knowledge Management.

Scope

More and more, software is not only a highly innovative and economically important sector on its own but software is also an important driver of innovation for most other industry sectors. In this context, knowledge – the experience, insights, understandings, and practical know-how of highly educated, skilled and experienced employees – is one of a company's most important assets, enabling individual and organizational intelligent behavior and, thus, the development and delivery of high-quality products and services. Therefore, the ability to share and leverage knowledge on individual, team and organization levels becomes critical to its competitive advantage. To maximize productivity and quality gains methods and tool support, organizational, team and individual perspectives need to be fostered and balanced.

The goal of LSO is to discuss all aspects of learning organizations and knowledge management solutions in the Software Engineering domain. LSO aims at the discussion of existing applications in practice, the examination of problems and limitations experienced in practice, and the search for innovative approaches to overcome problems, improve existing approaches and enable the wide-spread establishment of learning organizations in practice.

Implementing Product Line Engineering in Industry: Feedback from the Field to Research

Davide Falessi¹ and Dirk Muthig²

¹ University of Roma Tor Vergata, DISP, Rome, Italy

² Fraunhofer Institute for Experimental Software Engineering, Kaiserslautern, Germany
falessi@ing.uniroma2.it, dirk.muthig@iese.fraunhofer.de

Software product line engineering refers to methods (tools and techniques) for creating sets of similar software systems by taking advantage of their commonalities and predicted variabilities. After successfully installing product line engineering in practice, organizations typically experience a great improvement with respect to productivity or quality. Product line engineering is a proactive and strategic approach towards software reuse that requires the involvement of whole organizations to be successful. Implementing product line engineering in practice thus corresponds to identifying a strategy that transforms an existing organization into an organization that is fully centered around their product line(s). Although this transformation impacts nearly everything in an organization, it is often realized in an incremental way to manage risks and to ensure that production continues during the change process.

This workshop aims at a great collection of experience reports in implementing product line engineering in practice. The reports should expose best practices and typical challenges of transferring product line technology into practice. During the workshop we will also try to identify proven first steps that make a first, successful step towards the ultimate vision of product line engineering.

Therefore, we are particularly interested in discussing industrial experiences gained in the following areas:

- Initiating product line adoption.
- Estimating or monitoring the ROI of product line technology
- Managing organizational changes and issues while adopting product line technology

What to Learn from Different Standards and Measurement Approaches? Is a Pragmatic Integrative Approach Possible?

Fabio Bella and Horst Degen-Hientz

KUGLER MAAG CIE GmbH, Leibnizstr. 11, 70806 Kornwestheim, Germany
{fabio.bella, horst.degen-hientz}@kuglermaag.com

Several measurement approaches such as Balanced Score Cards (BSC), Six Sigma, or Goal Question Metrics (GQM), can be applied to set up and embed measurement programs within the scope of software projects. Different approaches may highlight different aspects of the measurement process or address different organizational levels. They may also show complementary aspects and provide a maximum of benefit when jointly applied. Often different approaches are applied by different groups within one same organization and effort has to be spent to integrate different measurement initiatives. To maximize the information gain and minimize the effort to be spent in measurement-related activities, informed decisions have to be made and the right approaches must be chosen when setting up a measurement program. The aim of this workshop is to present single measurement approaches currently available, share experience gathered from their application, and sketch a map to assist preliminary decision making when defining measurement programs.

Author Index

- Ackermann, Christopher 158
Ahmad, Rashid 143
Ahonen, Jarmo J. 258
Alexandre, Simon 189
Ali Babar, Muhammad 143, 400, 427
Ardimento, Pasquale 289
Ast, Stefan 304
- Baldassarre, Maria Teresa 415
Bella, Fabio 133, 445
Bertolino, Antonia 1
Birkhölzer, Thomas 304
Boffoli, Nicola 415
Brito e Abreu, Fernando 330
Buglione, Luigi 75
- Caivano, Danilo 415
Caldeira, João 330
Catal, Cagatay 244
Cimitile, Marta 289
Ciolkowski, Marcus 442
Concas, Giulio 386
Conradi, Reidar 158, 400
Cruzes, Daniela 158
- Daneva, Maya 90
Degen-Hientz, Horst 2, 445
Demirors, Onur 105
Deprez, Jean-Christophe 189
Di Francesco, Marco 386
Díaz-Ley, María 19
Dickmann, Christoph 304
Diri, Banu 244
- Eteläperä, Matti 174
- Falessi, Davide 444
Feldmann, Raimund L. 443
Fukazawa, Yoshiaki 45
- García, Félix 19
Gencel, Cigdem 75, 105
Giachetti, Giovanni 215
Gupta, Anita 158
- Hanssen, Geir Kjetil 371
Harjuma, Lasse 230
Heidrich, Jens 4, 442
Hickman, Charles 143
Hiraguchi, Hiroki 45
Holz, Wolfgang 34
Hörmann, Klaus 133
- Ibrahim, Suhaimi 427
- Jäntti, Marko 317
- Kaur, Arvinder 204
Klein, Harald 304
Kuhmann, Marco 442
- Land, Rikard 117
Landre, Einar 158
Li, Jingyue 400
Lindgren, Markus 117
Liukkunen, Kari 174
- Magazinovic, Ana 61
Malhotra, Ruchika 204
Marchesi, Michele 386
Marín, Beatriz 215
Markkula, Jouni 230
Meyer, Ludger 304
Mishra, Alok 273
Mishra, Deepti 273
Moe, Nils Brede 345
Moser, Raimund 360
Münch, Jürgen 4, 442
Muthig, Dirk 444
- Niazi, Mahmood 143, 427
Norström, Christer 117
- Öhman, Peter 61
Oivo, Markku 174, 230
Ozcan Top, Ozden 105
Ozkan, Baris 105
- Pastor, Oscar 215
Pedrycz, Witold 360
Pellikka, Mika 174

- Pernstål, Joakim 61
Piattini, Mario 19
Pinna, Sandro 386
Premraj, Rahul 34
Pylkkänen, Niko 317
- Quaresima, Roberta 386
- Rønneberg, Harald 158
- Schneider, Kurt 3
Shull, Forrest 158
Sillitti, Alberto 360
Singh, Yogesh 204
Slyngstad, Odd Petter Nord 400
Šmite, Darja 345
Soininen, Juha-Pekka 174
- Stålhane, Tor 371
Succi, Giancarlo 360
- Torkar, Richard 345
Turetken, Oktay 105
- Valtanan, Anu 258
Vanamali, Bhaskar 133
Vaupel, Jürgen 304
Visaggio, Giuseppe 415
- Wall, Anders 117
Washizaki, Hironori 45
Wessner, Martin 443
- Zeller, Andreas 34
Zimmermann, Thomas 34