Qing Wang
Vahid Garousi
Raymond Madachy
Dietmar Pfahl (Eds.)

LNCS 5543

Trustworthy Software Development Processes

International Conference on Software Process, ICSP 2009
Vancouver, Canada, May 2009
Proceedings

Springer

Lecture Notes in Computer Science 5543

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Qing Wang Vahid Garousi
Raymond Madachy Dietmar Pfahl (Eds.)

Trustworthy Software Development Processes

International Conference on Software Process, ICSP 2009
Vancouver, Canada, May 16-17, 2009
Proceedings

Springer

Volume Editors

Qing Wang
Institute of Software
Chinese Academy of Sciences
4 South Fourth Street, Zhong Guan Cun, Beijing 100190, China
E-mail: wq@itechs.iscas.ac.cn

Vahid Garousi
University of Calgary
Schulich School of Engineering
Department of Electrical and Computer Engineering
2500 University Drive N.W., Calgary, AB T2N 1N4, Canada
E-mail: vgarousi@ucalgary.ca

Raymond Madachy
Naval Postgraduate School
Department of Systems Engineering
Bullard Hall, Room 201J, 777 Dyer Road, Monterey, CA 93943, USA
E-mail: rjmadach@nps.edu

Dietmar Pfahl
Simula Research Laboratory
P.O.Box 134, 1325 Lysaker, Norway
E-mail: dietmarp@simula.no

Library of Congress Control Number: Applied for

CR Subject Classification (1998): D.2, I.6, D.2.1, D.1, D.3.3

LNCS Sublibrary: SL 2 – Programming and Software Engineering

ISSN 0302-9743
ISBN-10 3-642-01679-0 Springer Berlin Heidelberg New York
ISBN-13 978-3-642-01679-0 Springer Berlin Heidelberg New York

springer.com

Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12677566 06/3180 5 4 3 2 1 0

Preface

This volume contains papers presented at the International Conference on Software Process (ICSP 2009) held in Vancouver, Canada, during May 16-17, 2009. ICSP 2009 was the third conference of the ICSP series, continuing the software process workshops from 25 years ago. The theme of ICSP 2009 was “Processes to Develop Trustworthy Software.”

Software development takes place in a dynamic context of frequently changing technologies and limited resources. Teams worldwide are under increasing pressure to deliver trustworthy software products more quickly and with higher levels of quality. At the same time, global competition is forcing software development organizations to cut costs by rationalizing processes, outsourcing part or all of their activities, reusing existing software in new or modified applications and evolving existing systems to meet new needs, while still minimizing the risk of projects failing to deliver. To address these difficulties, new or modified processes are emerging including lean and agile methods, plan-based product line development, and increased integration with systems engineering processes.

Papers present research and real-world experience in many areas of software and systems processes impacting trustworthy software including: new software development approaches; software quality; integrating software and business processes; CMMI and other process improvement initiatives; simulation and modeling of software processes; techniques for software process representation and analysis; and process tools and metrics.

In response to the call for papers, 96 submissions were received from 26 different countries and regions including: Australia, Austria, Brazil, Canada, Chile, China, Colombia, Finland, France, Germany, Greece, India, Ireland, Italy, Japan, Mexico, New Zealand, Pakistan, Russia, Serbia, Singapore, Spain, Sweden, Turkey, UK, and USA. Each paper was rigorously reviewed and held to very high quality standards, and finally 33 papers from 12 countries and regions were accepted as regular papers for presentations at the conference.

The papers were clustered around topics and presented in seven regular sessions organized in two parallel threads. Topics included process management, process tools, process modeling and representation, process analysis, process simulation modeling, experience report, process metrics.

Highlights of the ICSP2009 program were three keynote speeches, delivered by Günther Ruhe (University of Calgary, Canada), Rick Selby (Northrop Grumman Space Technology, USA), and Lionel C. Briand (Simula Research Laboratory and University of Oslo, Norway).

On this 25th anniversary of workshops in the field, it was gratifying to see increasing maturity in the work with the continued high rate of submissions from all over the world. Although this was only the third ICSP conference, it continued a long tradition of important workshops and conferences in the field starting with the International Software Process Workshop (ISPW, from 1984 to 1996), the International

Conference on the Software Process (ICSP, from 1991 until 1996), the International Workshop on Software Process Simulation and Modeling (ProSim, from 1998 until 2006), and the Software Process Workshop (SPW, in 2005 and 2006). ProSim and SPW were held together in 2006 and merged in 2007 to form the new International Conference on Software Process. This year we were able to tighten the review process with the help of our reviewers to keep up the tradition.

This conference would not have been possible without the dedication and professional work of many colleagues. We wish to express our gratitude to all contributors for submitting papers. Their work forms the basis for the success of the conference. We also would like to thank the Program Committee members and reviewers because their work guarantees the high quality of the workshop. Special thanks go to the keynote speakers for giving their excellent presentations at the conference. Finally, we also would like to thank the members of the Steering Committee, Barry Boehm, Mingshu Li, Leon Osterweil, David Raffo, and Wilhelm Schäfer for their advice, encouragement, and support.

We wish to express our thanks to the organizers for their hard work. The conference was sponsored by the Chinese Academy of Sciences (ISCAS) and the ISCAS Laboratory for Internet Software Technologies (iTechs). We also wish to thank the 31st International Conference on Software Engineering (ICSE 2009) for sponsoring this meeting as an ICSE co-located event. Finally, we acknowledge the editorial support from Springer for the publication of this proceeding.

For further information, please visit our website at
http://www.icsp-conferences.org/icsp2009.

March 2009

Dietmar Pfahl
Raymond Madachy
Qing Wang

International Conference on Software Process 2009

Vancouver, Canada
May 16–17, 2009

Steering Committee

Barry Boehm	University of Southern California, USA
Mingshu Li	Institute of Software, Chinese Academy of Sciences, China
Leon J. Osterweil	University of Massachusetts, USA
David M. Raffo	Portland State University, USA
Wihelm Schäfer	University of Paderborn, Germany

General Chair

Dietmar Pfahl	Simula Research Laboratory and University of Oslo, Norway

Program Co-chairs

Raymond Madachy	Naval Postgraduate School, USA
Qing Wang	Institute of Software, Chinese Academy of Sciences, China

Publicity Co-chairs

Vahid Garousi	University of Calgary, Canada

Secretary

Juan Li	Institute of Software, Chinese Academy of Sciences, China
Lizi Xie	Institute of Software, Chinese Academy of Sciences, China

Program Committee

Muhammad Ali Babar	University of Limerick, Ireland
Stefan Biffl	Technische Universität Wien, Austria
Thomas Birkhölzer	University of Applied Science, Konstanz, Germany
Danilo Caivano	University of Bari, Italy

Keith Chan	Hong Kong Polytechnic University, Hong Kong
Sorana Cimpan	University of Savoie at Annecy, France
Oscar Dieste	Universidad Politecnica de Madrid, Spain
Jacky Estublier	French National Research Center in Grenoble, France
Anthony Finkelstein	University College London, UK
Vahid Garousi	University of Calgary, Canada
Dennis Goldenson	Carnegie Mellon University, USA
Volker Gruhn	University of Leipzig, Germany
Paul Grünbacher	Johannes Kepler University Linz, Austria
Kequing He	Wuhan University, China
Dan Houston	The Aerospace Corporation, USA
LiGuo Huang	Southern Methodist University, USA
Hajimu Iida	Nara Institute of Science and Technology, Japan
Katsuro Inoue	Osaka University, Japan
Ross Jeffery	University of New South Wales, Australia
Raymond Madachy	Naval Postgraduate School, USA
Frank Maurer	University of Calgary, Canada
James Miller	University of Alberta, Canada
Jürgen Münch	Fraunhofer Institute for Experimental Software Engineering, Germany
Flavio Oquendo	University of South Brittany, France
Dewayne E. Perry	University of Texas at Austin, USA
Dietmar Pfahl	Simula Research Laboratory, Norway
Dan Port	University of Hawaii, USA
Juan F. Ramil	The Open University, UK
Andreas Rausch	Technische Universität Kaiserslautern, Germany
Daniel Rodriguez	University of Alcalá , Spain
Günther Ruhe	University of Calgary, Canada
Mercedes Ruiz	University of Cádiz, Spain
Ioana Rus	University of Maryland, USA
Walt Scacchi	University of California, Irvine, USA
Barbara Staudt Lerner	Mount Holyoke College, USA
Stan Sutton	IBM T. J. Watson Research Center, USA
Guilherme H Travassos	Federal University of Rio de Janeiro/COPPE, Brazil
Qing Wang	Institute of Software, Chinese Academy of Sciences, China
Yasha Wang	Peking University, China
Brian Warboys	University of Manchester, UK
Paul Wernick	University of Hertfordshire, UK

Ye Yang	Institute of Software, Chinese Academy of Sciences, China
Yun Yang	Swinburne University of Technology, Australia
Li Zhang	Beihang University, China

External Reviewers

Fengdi Shu	Institute of Software, Chinese Academy of Sciences, China
Junchao Xiao	Institute of Software, Chinese Academy of Sciences, China
Jian Zhai	Institute of Software, Chinese Academy of Sciences, China
Lizi Xie	Institute of Software, Chinese Academy of Sciences, China
Dapeng Liu	Institute of Software, Chinese Academy of Sciences, China

Table of Contents

Invited Talks

Process Management

Process Tools

Process Analysis

Process Simulation Modeling

Experience Report

Process Metrics

Process Modeling and Representation

System Engineering in the Energy and Maritime Sectors: Towards a Solution Based on Model-Centric Processes

Lionel Briand

Simula Research Laboratory
University of Oslo

Abstract. The Maritime and Energy industry is facing rapid change with an increasing reliance on software embedded systems and integrated control and monitoring systems. From a practical stand point, challenges are related to increased system complexity, increasingly integrated sub-systems relying on Commercial-Of-The-Shelf software, longer supply chains for equipment and components delivered by different suppliers, and short duration for construction and commissioning of ships and offshore platforms. As a result, there is a lack of visibility into the architecture of systems, their design rationale, how subsystems/components were verified and integrated, and finally how systems were validated and certified with a particular focus on safety. In turn, this has hindered effective collaboration among stakeholders, including suppliers and system integrators.

This talk will present a recent initiative, led by Simula Research Laboratory and Det Norske Veritas (DNV), Norway, to address the above problems. The general approach relies on model-centric processes, where models of the system specifications, architecture, and design properties, are used to support the documentation of architecture and design rationale, traceability among development artifacts, and guide safety analysis and testing, among other things. The project is focused on devising novel but scalable approaches to the long-standing model-driven development challenges.

Q. Wang et al. (Eds.): ICSP 2009, LNCS 5543, p. 1, 2009.

Decision Processes for Trustworthy Software

Guenther Ruhe

University of Calgary

Abstract. A recent survey done across different engineering domains has revealed that support of decision-making is one of the key motivations for performing process modeling. Decisions are focal points at all stages of the software development lifecycle. The appropriateness of decisions made about methods, tools, techniques, activities and resources is of pivotal importance for the success (or failure) of a project. The discipline of software engineering decision support studies the processes of facilitating "good decisions". Besides effectiveness and efficiency, transparency is a key characteristic of the "goodness" of decisions.

This talk analyzes software engineering decision-making processes with emphasis on its transparency. From empirical studies it was shown that transparency of processes improves trustworthiness into solutions offered. This is especially true for solutions offered from decision support systems with comprehensive and complex computations done to generate candidate decision alternatives. More specifically, we discuss the transparency of decisions for the class of product release decision process. We provide the state-of-the art practice in this area and summarize existing research to address current deficits. Experience from a series of real-world projects in this area is analyzed from the perspective of how transparent processes have contributed to trustworthy software.

Q. Wang et al. (Eds.): ICSP 2009, LNCS 5543, p. 2, 2009.

Synthesis, Analysis, and Modeling of Large-Scale Mission-Critical Embedded Software Systems

Richard W. Selby

Northrop Grumman Space Technology
Computer Science Department, University of Southern California

Abstract. Mission-critical embedded software performs the core processing logic for pervasive systems that affect people and enterprises everyday, ranging from aerospace systems to financial markets to automotive systems. In order to function properly, these embedded software systems rely on and are highly interdependent with other hardware and software systems. This research identifies design principles for large-scale mission-critical embedded software and investigates their application in development strategies, architectures, and techniques. We have examined actual embedded software systems from two different problem domains, advanced robotic spacecraft and financial market systems, and these analyses established the foundations for these design principles. Both system types embody solutions that respond to detailed specifications defined and modeled with heavy user involvement. Both system types possess mission-critical logic represented using state machines and other structured techniques. They both use a layered architecture approach with a foundation that provides infrastructure services, a layer with a simple set of foreground and background tasks, a layer with deterministic synchronous processing steps, and a layer with event-driven monitoring, commanding, and sequencing capabilities. The architectural approach supports a domain-specific command sequencing macro language that defines table-driven executable specifications and enables developers to work at higher abstraction levels throughout the lifecycle. The architectural approach also facilitates extensibility, reuse, and portability across multi-processor execution environments. The systems rely on extensive built-in self-tests, invariants, and redundant calculations that assess states and detect faults. From a development standpoint, both systems use risk-driven incremental lifecycles, system modeling, end-to-end prototyping, and statistical analysis of development processes. Based on insights gained from embedded software design principles and their application on these as well as other systems, improvement opportunities and research directions will be identified.

1 Introduction

Software products for large-scale mission-critical embedded systems require systematic principles for development strategies and architectures as well as for development processes and improvement methods. This paper summarizes software product development principles established and validated across aerospace and financial market systems. The illustrations describe examples that emphasize aerospace software. The development

Q. Wang et al. (Eds.): ICSP 2009, LNCS 5543, pp. 3–10, 2009.

environment examined has an organizational charter that focuses on embedded software products for advanced robotic spacecraft platforms, high-bandwidth satellite payloads, and high-power laser systems (Figure 1). The environment emphasizes both system management and payload software and has well-defined reusable, reconfigurable software architectures and components. The development languages range from object-oriented to C to assembler. The organization was appraised at CMMI Level 5 for software and also supports other process improvement initiatives such as ISO/AS9100 and Six Sigma [Hum88] [Wom90]. The developers are skilled in producing high-reliability, long-life, real-time embedded software systems.

2 Example Embedded Software System

This research examines one example spacecraft called Prometheus intended for a mission to Jupiter that enables data-intensive exploratory deep-space science. The spacecraft configuration has 58m length, 36,000kg launch mass, five on-board processors (excluding redundancy), 250mbps internal data transfer rate, 500gbit on-board storage, and 10mbps external data downlink rate (Figure 2). The spacecraft has gas-cooled power with 200kW Brayton output, and it stows in a 5m diameter fairing. The embedded software implements functions for commands and telemetry, subsystem algorithms, instrument support, data management, and fault protection. Over the last several years, the size of on-board software has continued to grow to accelerate data

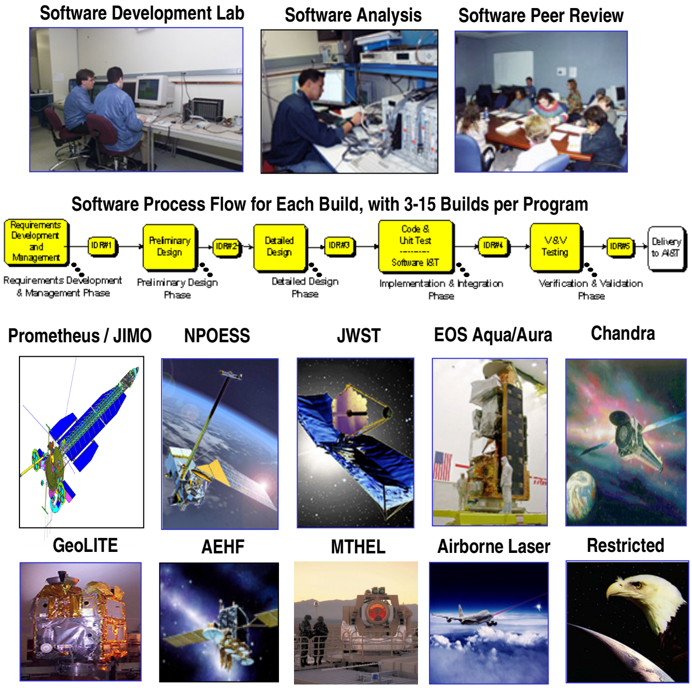


Fig. 1. Aerospace software development environment uses incremental software deliveries and extensive simulations and reviews

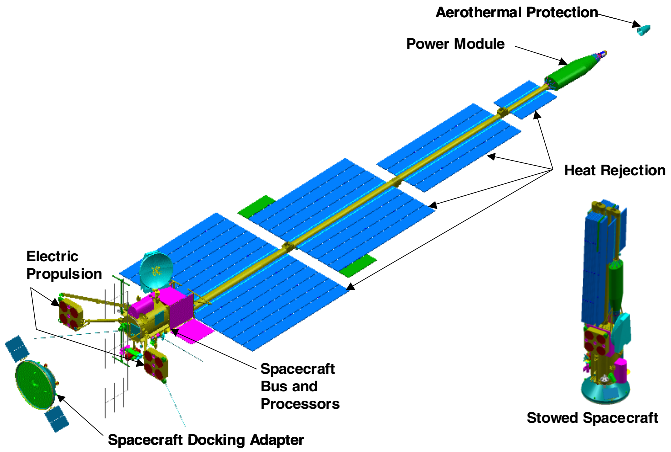


Fig. 2. Example spacecraft with embedded software system

processing and increase science yield. Functionality implemented in software, as opposed to hardware such as digital ASICs, adds value to missions by enabling post-delivery changes to expand capabilities and overcome hardware failures.

3 Embedded Software Design Principles

3.1 Embedded Software Design Principles for Strategies

The investigations establish the following example embedded software design principles for system development strategies:

- Share best-of-class software assets across organizations to leverage ideas and experience.
- Define common software processes and tools to improve communication and reduce risk.
- Adopt risk-driven incremental lifecycle with multiple software builds to improve visibility, accelerate delivery, and facilitate integration and test.
- Conduct early tradeoff studies and end-to-end prototyping to validate key requirements, algorithms, and interfaces.
- Design simple deterministic systems and analyze them with worst-case mindset to improve predictability.
- Analyze system safety to minimize risk.
- Conduct extensive reviews (intra-build peer reviews, five build-level review gates, higher level project reviews) and modeling, analysis, and execution (testbed conceptual, development, engineering, and flight models called CMs, DMs, EMs, and FMs, respectively) to enable thorough understanding, verification, and validation.

An example incremental software development strategy appears in Figure 3.

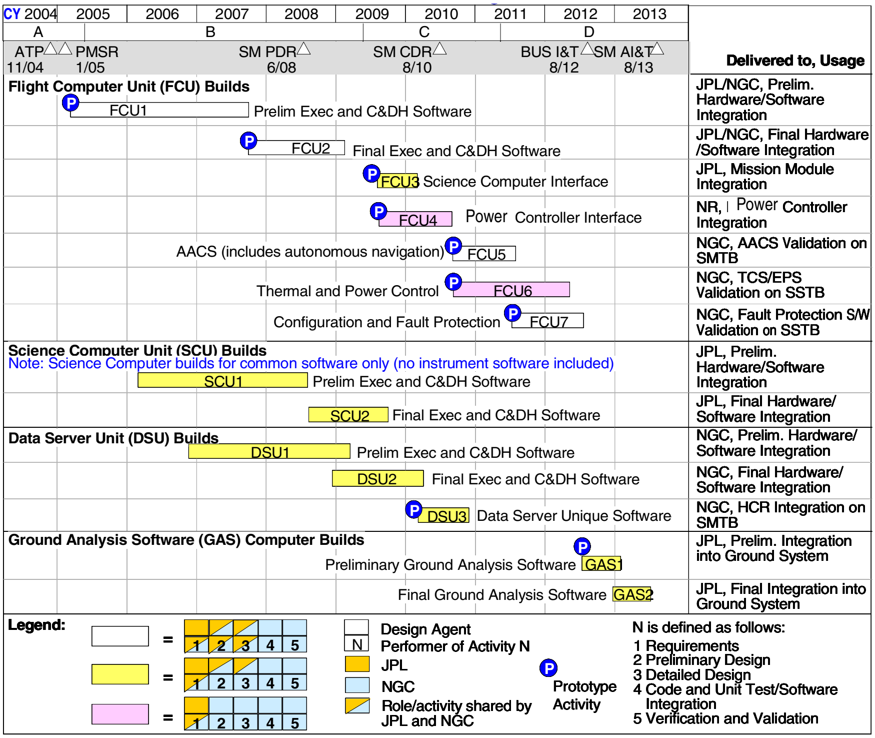


Fig. 3. Incremental software builds deliver early capabilities and accelerate integration and test

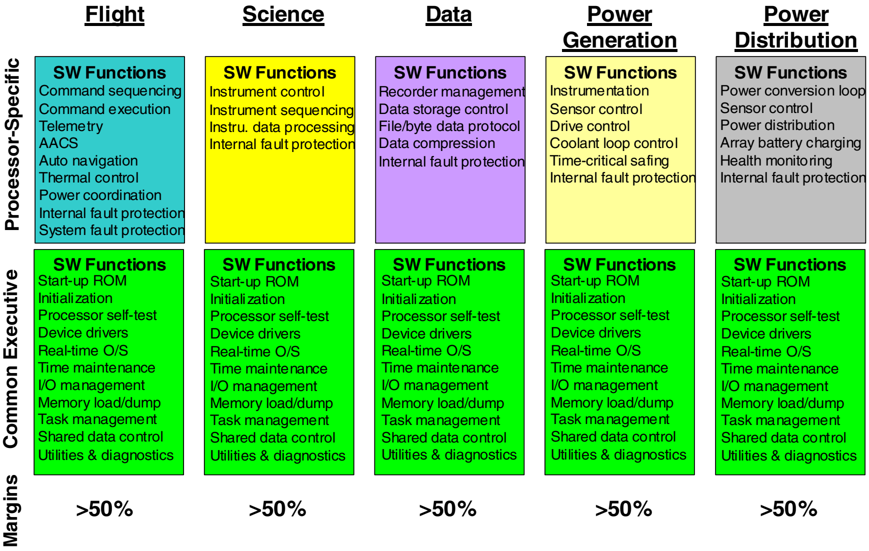
3.2 Embedded Software Design Principles for Architectures

The investigations establish the following example embedded software design principles for system architectures:

- Partition functions across multi-processor architecture (such as flight, science, data, power generation, power distribution) to distribute performance, allocate margins, and improve fault protection.
- Define software "executive" foundational layer that is common across multi-processor architecture to enable reuse and flexibility.
- Develop software using architectural simplicity, table-driven designs, deterministic behavior, and common interfaces to improve verifiability and predictability.
- Adopt high-level command sequencing "macro language" for non-software personnel, such as system engineers, to use, typically structured as table-driven templates of commands and parameters, to improve specifiability and verifiability.
- Define simple deterministic synchronous control-loop designs with well-defined task structures (typically 3-4 levels), static resource allocation (such as no dynamic memory allocation), and predictable timing (such as minimizing interrupts) to improve understandability and verifiability.

- Centralize system level autonomy and fault protection and distribute lower level autonomy and protection to appropriate control points to orchestrate system configurations, ensure timely isolations and responses, and support overall safety.
- Dedicate pathways for high-speed data (such as from payload instruments to high capacity storage) to separate specialized processing and faults from core functionality (such as payload versus spacecraft).
- Adopt large resource margins (processor, memory, storage, bus) to accommodate contingencies and post-delivery changes.

An example usage of a common software foundational layer coupled with partitioning of software functionality across processors appears in Figure 4.



	Flight	Science	Data	Power Generation	Power Distribution
Processor-Specific	SW Functions Command sequencing Command execution Telemetry AACS Auto navigation Thermal control Power coordination Internal fault protection System fault protection	SW Functions Instrument control Instrument sequencing Instru. data processing Internal fault protection	SW Functions Recorder management Data storage control File/byte data protocol Data compression Internal fault protection	SW Functions Instrumentation Sensor control Drive control Coolant loop control Time-critical safing Internal fault protection	SW Functions Power conversion loop Sensor control Power distribution Array battery charging Health monitoring Internal fault protection
Common Executive	SW Functions Start-up ROM Initialization Processor self-test Device drivers Real-time O/S Time maintenance I/O management Memory load/dump Task management Shared data control Utilities & diagnostics	SW Functions Start-up ROM Initialization Processor self-test Device drivers Real-time O/S Time maintenance I/O management Memory load/dump Task management Shared data control Utilities & diagnostics	SW Functions Start-up ROM Initialization Processor self-test Device drivers Real-time O/S Time maintenance I/O management Memory load/dump Task management Shared data control Utilities & diagnostics	SW Functions Start-up ROM Initialization Processor self-test Device drivers Real-time O/S Time maintenance I/O management Memory load/dump Task management Shared data control Utilities & diagnostics	SW Functions Start-up ROM Initialization Processor self-test Device drivers Real-time O/S Time maintenance I/O management Memory load/dump Task management Shared data control Utilities & diagnostics
Margins	>50%	>50%	>50%	>50%	>50%

Fig. 4. Five-processor architecture provides partitioned functions, common executive layer, and growth margins. Partition of software functions across processors improves performance, margins, and fault protection.

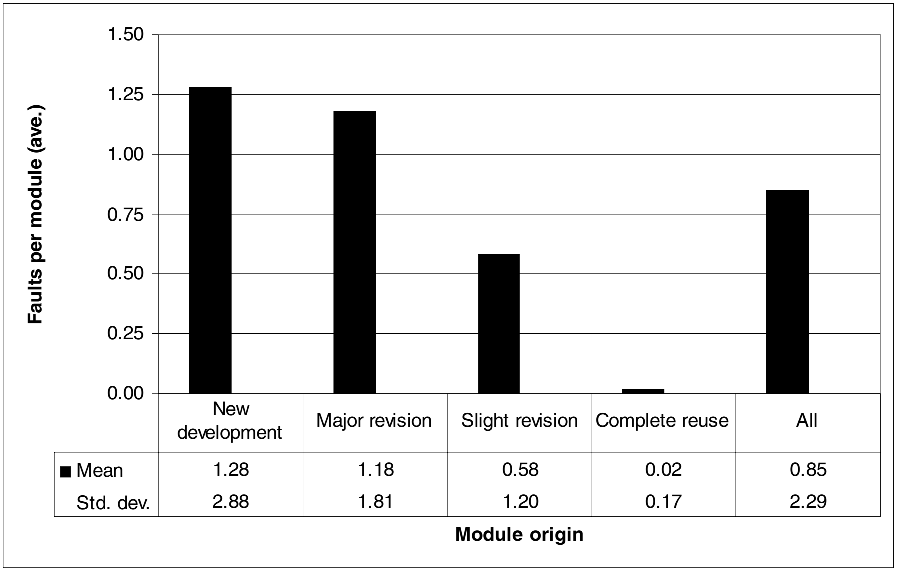
3.3 Embedded Software Design Principles for Techniques

The investigations establish the following example embedded software design principles for system development techniques:

- Flowdown requirements systematically from project, system (space, ground, launch, etc.), module (spacecraft, mission, etc.), segment (bus, software, etc.), subsystem/build, assembly, etc. to clarify functionality and accountability.
- Identify a manageable number of “key driving requirements”, where key is top-down mission-success and driving is bottom-up design-limiting, to prioritize attention and analysis.

- Define user-perspective “mission threads” to focus modeling, end-to-end prototyping, and validation.
- Specify “command abstractions” that define standalone command primitives with pre-conditions, atomic processing, resource constraints (such as timing), and post-conditions (such as data modified) to enable analysis and predictability.
- Define and enforce “control points”, such as centralized sequential command queue and explicit data dependency graphs for read/write of data shared across commands and sequences, to facilitate analysis and isolate faults.
- Include built-in self-tests, invariants, and redundant calculations in implementations to help ensure accurate processing and isolate faults.
- Compare executions of system models and software implementations automatically using toolsets to improve verification.
- Apply workflow tools, checklists, statistical analyses, root cause analyses, and metric dashboards to improve repeatability, visibility, and preventability.

An example analysis of fault-proneness across newly developed, modified, and reused software components appears in Figure 5 [Sel05]. An example comparative study of software fault detection techniques appears in Figure 6 [Bas99] [Myr05].



	New development	Major revision	Slight revision	Complete reuse	All
■ Mean	1.28	1.18	0.58	0.02	0.85
Std. dev.	2.88	1.81	1.20	0.17	2.29

Fig. 5. Analyses of component-based software reuse shows favorable trends for decreasing faults, based on data from 25 NASA systems. Overall difference is statistically significant (a < .0001). Number of components (or modules) in each category is: 1629, 205, 300, 820, and 2954, respectively.

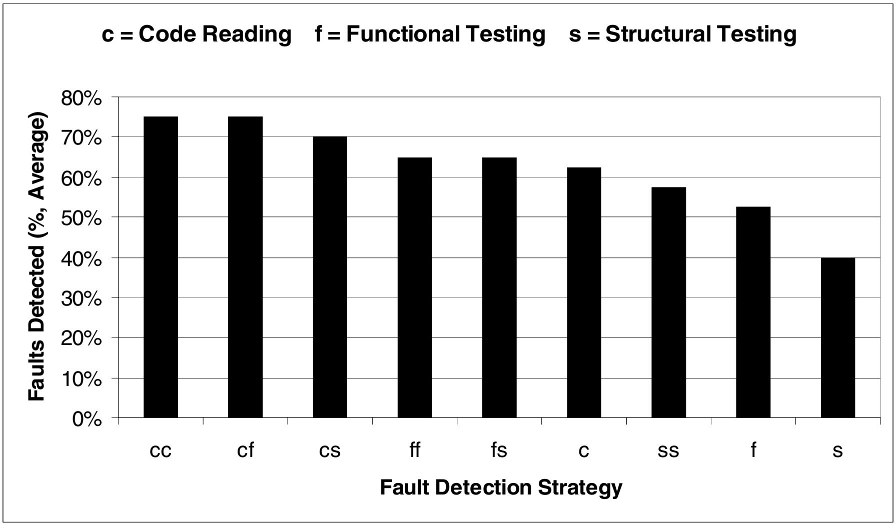


Fig. 6. Analyses of fault detection strategies characterize fault types and effectiveness of teaming. Comparisons use component-level fault detection strategies applied by 32 NASA developers and two-person developer teams. The combinations include single and paired developers using: (c) code reading by stepwise abstraction, (f) functional testing using equivalence partitioning and boundary value analysis, and (s) structural testing using 100% statement coverage criteria.

4 Future Research

The continuous improvement of software development methods requires effective and efficient identification of improvement opportunities and systematic steps actualizing changes [Boe81] [Sel91] [Gra00]. Our future research further explores the parallels among embedded software for aerospace, financial, and related mission-critical problem domains. Systems in these domains typically utilize a distributed network of embedded processors and rely on embedded software to implement major instrumentation, processing, and control functions. Identification of common and domain-specific software design principles enables sharing of lessons learned and opportunities for adapting specialized techniques to increase system capabilities, enhance extensibility, and improve fault prevention and detection.

References

1. [Bas99] Basili, V.R., Shull, F., Lanubile, F.: Building Knowledge through Families of Experiments. IEEE Transactions on Software Engineering SE-25(4), 456–473 (1999)
2. [Boe81] Boehm, B.W.: Software Engineering Economics. Prentice-Hall, Englewood Cliffs (1981)

3. [Gra00] Graves, T.L., Karr, A.F., Marron, J.S., Siy, H.: Predicting Fault Incidence Using Software Change History. IEEE Transactions on Software Engineering SE-26(7), 653–661 (2000)
4. [Hum88] Humphrey, W.S.: Characterizing the Software Process: A Maturity Framework. IEEE Software 5(2), 73–79 (1988)
5. [Myr05] Myrtveit, I., Stensrud, E., Shepperd, M.: Reliability and Validity in Comparative Studies of Software Prediction Models. IEEE Transactions on Software Engineering SE-31(5), 380–391 (2005)
6. [Sel91] 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, Austin, TX (May 1991)
7. [Sel05] Selby, R.W.: Enabling Reuse-Based Software Development of Large-Scale Systems. IEEE Transactions on Software Engineering SE-31(6), 495–510 (2005)
8. [Wom90] Womack, J.P., Jones, D.T., Roos, D.: The Machine that Changed the World: The Triumph of Lean Production. Rawson Associates, New York (1990)

Statistically Based Process Monitoring: Lessons from the Trench

Maria Teresa Baldassarre, Nicola Boffoli, Giovanni Bruno, and Danilo Caivano

Department of Informatics, University of Bari – SER&Practices SPIN OFF
Via E. Orabona 4 - 70126 - Bari - Italy
{baldassarre,boffoli,bruno,caivano}@di.uniba.it

Abstract. Monitoring software processes is a non trivial task. Recently many authors have suggested the use of Statistical Process Control (SPC) for monitoring software processes, while others have pointed out its potential pitfalls. Indeed, the main problem is that SPC is often used "as is" without the appropriate customizations or extensions needed for making it applicable to software contexts. This work points out and discusses four main issues related to software process monitoring and highlights how SPC can be used as solution to address each problem. The solutions arise from experience collected by the authors during empirical investigations in industrial contexts. As so, this work is intended as a first step in clarifying how SPC can contribute to practically solve some monitoring issues and guide practitioners towards a more disciplined and correct use of the approach in controlling software processes.

Keywords: Statistical Process Control, Software Process Monitoring.

1 Introduction

A software process can be considered as a synergic blend of man, machine and methods in working activities whose execution leads to the production of desired outputs, starting from the available inputs [1, 2]. Product quality is tightly related to the quality of the processes used to produce them and therefore "improve quality" means "improve software processes". This implies the need for monitoring process execution, highlighting process anomalies and being able to quickly react to them. Since software processes are mainly human intensive and dominated by cognitive activities, each process execution is a creative and unique activity where the predominant human factor implies differences in process performances and thus multiple outputs. The phenomena known as "Process Diversity" [3, 4], makes process prediction, monitoring, and improvement activities a non trivial task. "Monitoring" is commonly intended as measurement of process performances over time, using process metrics (such as execution time, productivity, fault detected, costs etc.) able to point out anomalies through some sort of alert mechanism and quickly react to them We can therefore state that process monitoring, implies the need to: (i) define process performance thresholds, i.e. typically upper and lower control limits for identifying admissible process performance variations, and use them for controlling the process behavior by comparing actual performances with admissible ones (i.e. process limits);

Q. Wang et al. (Eds.): ICSP 2009, LNCS 5543, pp. 11–23, 2009.

(ii) define the actual process anomalies so the monitor is able to identify them; (iii) investigate the anomalies, the exceptional performance variations, in order to understand the causes and react to them; (iv) dynamically tune monitoring sensibility and adapt it to process behavior and admissible process variations over time.

Given the considerations above, the following issues arise: *Problem1*: Baseline Definition. Given the heterogeneity of the operative contexts, the different maturity levels of the processes in use, and (often) the lack of past knowledge about the monitored process, it is difficult to define reliable and useful baselines for characterizing and evaluating performances. *Problem2*: Anomalies Detection. "Process anomalies" is a foggy concept. As so, conceptual and operational tools for identifying anomalies are needed. *Problem3*: Causes Investigation. The detected anomalies are difficult to interpret and it is challenging to discover the causes and undertake the appropriate actions. *Problem4*: Tuning Sensibility. Monitoring sensibility should be tuned according to process performance relevant changes.

In this paper we propose a couple: "monitoring problem - SPC based solution" (from here on referred to as "pattern"), for each of the four problems listed above, according to our experiences collected in previous empirical studies facing monitoring issues, and the use of Statistical Process Control (SPC) [2, 5, 6] in industrial projects [7, 8, 9, 10, 11, 12, 13], along with a systematic review on SPC [14].

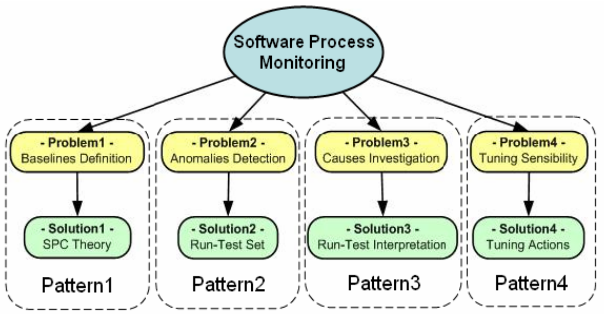


Fig. 1. Research Approach Schema

Furthermore, throughout the paper we use the empirical results as explanatory cases for showing how each solution works in practice. This paper isn't an answer to monitoring problems, nor is a silver bullet on the use of SPC. On the contrary, we are well aware that SPC applied to software presents various limitations of different nature, from theoretical to operational [15, 16, 17, 18, 19]. As so, this work is intended as a first step for clarifying how SPC can contribute to solve monitoring issues and guide practitioners towards a more disciplined use of the approach.

The paper is organized as follows: in section 2, the 4 patterns that describe the 4 "monitoring problem–SPC solution" couples are presented (Fig. 1); section 3 discusses the main outcomes of section 2 and organizes them in a disciplined process for guiding monitoring activity; finally, in section 4 conclusions are drawn.

2 Software Process Monitoring and Statistical Process Control: Identified Patterns

Each pattern (monitoring problem-SPC based solution couple) was obtained from the generalization of our experience collected during empirical investigations on the use of SPC in an industrial software project [7]. In such study SPC was applied to a legacy data set collected during the execution of a renewal project made up of two sub-projects: reverse engineering and restoration, in which the process improvements made were known. The project data was used to validate, through a simulation on legacy data, whether SPC would have been able to point out the process performance changes during project execution. The renewed legacy software system was an aged banking application, made of 638 Cobol programs. A total of 289 programs were subject to reverse engineering, while 349 to restoration. For each program, the metrics collected and analyzed during project execution were: *PERFORMANCE=NLOC*[1]*/EFFORT*[2]. The programs were renewed sequentially. In the explanatory case of this paper, we will only present the data related to the reverse engineering sub-project, from here on referred to as RE. However, similar considerations can be made for the restoration sub-project as well. Table 1 reports the *PERFORMANCE* value for the first 30 RE data points.

Table 1. Reverse Engineering Data Points

1	2	3	4	5	6	7	8	9	10
213.875	243.600	237.176	230.700	209.826	226.375	167.765	242.333	233.250	183.400
11	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
201.882	182.133	235.000	216.800	134.545	363.241	411.392	409.298	406.861	406.989
21	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**
404.147	425.801	293.073	423.644	353.160	416.469	496.097	358.352	511.646	396.718

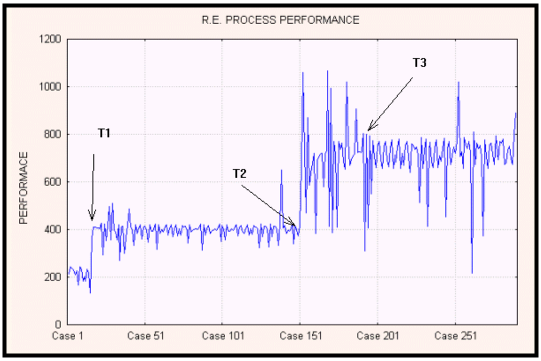


Fig. 2. Reverse engineering performance expressed in NLOC/Hour

[1] NLOC: number of lines of code in a program, excluding lines of comment and blank spaces. If a statement extends over several lines of listing it is considered as a single line of code.

[2] EFFORT: man-hours spent for the reverse engineering/restoration of a cobol program.

The trend of the performances is shown in figure 2 where three major improvements made during RE, identified as T1, T2, and T3 are recognizable. Just to provide some details, in T1 a more structured, formalized and transferable reverse engineering process was provided to the developers, in T2, a tool for automating the RE process was introduced, in T3 reading procedures for checking reengineered programs were introduced. As it can be seen in the graph, process performances vary deeply during project execution. In the next paragraphs we present and discuss each of the 4 patterns and relate them to the explanatory case on RE.

2.1 Pattern 1: Baselines Definition ➔ SPC Theory

Problem: Monitoring involves measuring a quantifiable process characteristic over time and pointing out anomalies[3]. A process must be characterized before being monitored, for example by using a couple of reasonable threshold values, one for the upper and one for the lower process performance limits. When the observed performance falls outside these limits, someone can argue that there is something wrong in the process. Moreover, process characterization requires past knowledge which is often lacking. This is especially true in innovative projects where a production process is used for the first time and no historical data is related to it.

Solution: SPC, developed by Shewart in the 1920s, has shown to be effective in manufacturing [20] and has recently been used in software contexts [21, 22, 23, 24, 25, 26]. It uses several "control charts" together with their indicators [27, 28] to establish operational limits for acceptable process variation. By using few data points, it is able to dynamically determine an upper and lower control limit of acceptable process performance variability. Such peculiarity makes SPC a suitable instrument to face problem 1. A control chart usually adopts an indicator of the process performances central tendency (CL) and upper and lower control limits (UCLs and LCLs). Process performances are tracked overtime on a control chart, and if one or more of the values fall outside these limits, or exhibit a "non random" behavior an anomaly (technically speaking, the effect of an assignable cause) is assumed to be present. Many control charts exist in literature, but in software processes, due to the scarceness of data and since measurements often occur only as individual values, the most used ones are the XmR i.e. individual (X) and moving range (mR) charts (Fig. 3) [29, 30]. We will briefly introduce them. In an X chart: each point represents a single value of the measurable process characteristic under observation; CL_X calculated as the average of the all available values; UCL_X and LCL_X are set at $3sigma_X$ around the CL_X; $sigma_X$ is the estimated standard deviation of the observed sample of values [6, 31] and it is based on statistical reasoning, simulations carried out and upon the heuristic experience that: "it works"[4]. In a mR chart: each point represents a moving range (i.e. the absolute difference between a successive pair of observations); CL_{mR}, is the average

[3] Examples of anomalies are: reduction in productivity, an exceptional defect density or an u execution time (too high or too slow).

[4] Simulations carried out have shown that the following rules of thumb work:

- Rule1: from 60% to 75% of the observations fall in the CL ± sigma.
- Rule2: from 90% to 98% of the observations fall in the CL ± 2sigma.
- Rule3: from 99% to 100% of the observations fall in the CL ± 3sigma.

of the moving ranges; $UCL_{mR} = CL_{mR}+3sigma_{mR}$ and $LCL_{mR}=0$; $sigma_{mR}$ is the estimated standard deviation of the moving ranges sample. With reference to the explanatory case, if we consider the first 15 RE data points (reported in table 1), we obtain:

$$\overline{mR} = \frac{1}{m-1} \times \sum_{i=1....m-1} |x_{i-1} - x_i| = 33.11$$

$$3sigma_X = 2{,}660 * \overline{mR} = 88.07$$

$$CL_X = \overline{X} = 210.58$$

$$UCL_X = \overline{X} + 2{,}660 * \overline{mR} = 298.64;$$

$$LCL_X = \overline{X} - 2{,}660 * \overline{mR} = 122.52$$

$$CL_{mR} = \overline{mR} = 33{,}11;$$

$$UCL_{mR} = 3{,}268 * \overline{mR} = 108{,}2;$$

$$LCL_{mR} = 0$$

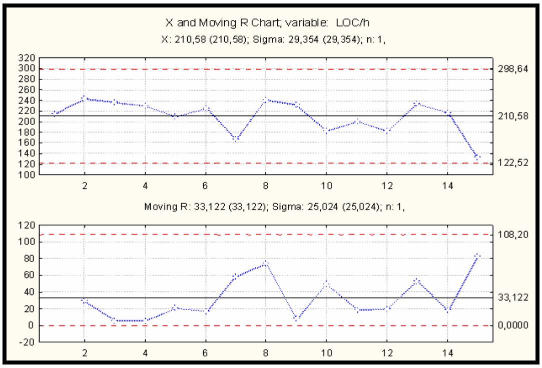


Fig. 3. XmR charts on the first 15 RE data points

The control charts are presented in figure 3. Note that the control limits carried out using SPC are based on a process observation and they are expression of it. They are not the result of expert judgment and, therefore obtained deterministically.

2.2 Pattern 2: Anomalies Detection ➔ Run-Test Set

Problem: monitoring activity points out process anomalies. Anomalies are intended as some kind of noise in the process performances, the results of an unknown cause in action that implies unattended variation (in better or worse) and thus lower process predictability[5]. During process monitoring, can we say that an anomaly is detected: when a set of observations rather than a single point fall(s) outside the specified threshold? or when there is an increasing/decreasing trend in the observed performances? Usually, only manager experience can answer these questions.

Solution: in software processes, one should look for systematic patterns of points instead of single point exceptions, because such patterns emphasize that the process performance has shifted or is shifting. This surely leads to more insightful remarks and suggestions. There is a set of tests for such patterns referred to as “run rules” or “run tests” [32, 33] that aren’t well known (or used) in the software engineering community, as also pointed out by our systematic review [14]. As the sigma concept, the run rules are based on "statistical" reasoning. The solution to this problem is based on previous research results of the authors who have proposed a set of indicators [7, 9, 10, 12] consisting of a selection among those present in SPC literature of run tests resulting most appropriate for software (Table2).

[5] In the software context, relevant examples in this sense are: the introduction of a new case tool that speeds up development; the use of a new testing or inspection technique that reduces post release defects etc.

Table 2. Run-Test Set Details

Run-Test	Description
RT1: Three Sigma	1 point beyond a control limit (±3sigma)
RT2: Two Sigma	2 out of 3 points in a row beyond (±2sigma)
RT3: One Sigma	4 out of 5 points in a row beyond (±1sigma)
RT4: Run above/below CL	7 consecutive points above or below the centreline
RT5: Mixing/Overcontrol	8 points in a row on both sides of the centreline avoiding ±1sigma area
RT6: Stratification	15 points in a row within ±1sigma area
RT7: Oscillatory Trend	14 alternating up and down points in a row
RT8: Linear Trend	6 points in a row steadily increasing or decreasing

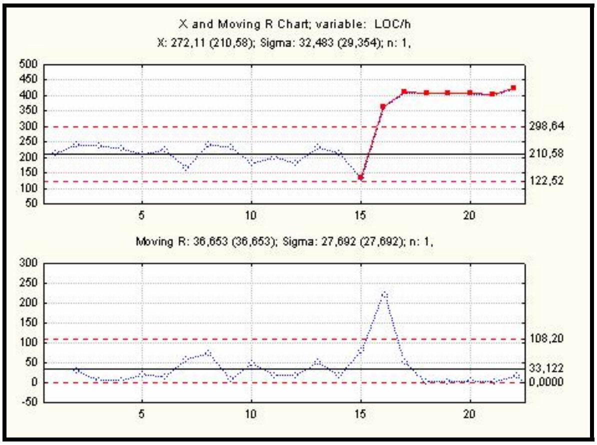


Fig. 4. Run Test Failures in XmR Charts

With reference to the explanatory case, if we continue plotting the data points, from 16 to 22, on the previous control charts (section 2.1), keeping the calculated control limits fixed, and checking the run tests for each added data point, we can observe the following anomalies (figure 4): RT1 (for all new points), RT2 (on sets of points from 15-17, 16-18, 17-19, 18-20, 19-21), RT3 (on sets of points from 15-19, 16-20,17-21), RT4 (on sets of points from 16-22), and RT5 (on sets of points from 15-22). Obviously this set of anomalies occurs over time as new data points are collected on the observed process. S, the monitor can face the problem and intervene appropriately.

2.3 Pattern 3: Causes Investigation ➔ Run-Test Interpretation

Problem: if the aim of monitoring activity is to point out process anomalies, Software Process Improvement aims to find the causes, eliminate them if detrimental or, otherwise, make them part of the process [34]. The possible causes of variation can be various such as their effects on process performances and, consequently, the observable anomalies. As so, a standard mechanism is needed to characterize the pointed out anomalies.

Solution: SPC is only able to detect whether the process performance is "out of control" and if an anomaly exists. It doesn't support the manager during the causes investigation and the selection of the appropriate corrective actions. This solution extends the SPC-theory by providing a specific interpretation (Table 3) of the anomaly for each run test failure (section 2.2) from the software process point of view, and suggesting possible causes that make the process "Out of Control" [7]. We have arranged and interpreted the selected SPC indicators (table 2) in logical classes: sigma (RT1, RT2, RT3), limit (RT4, RT5, RT6) and trend (RT7, RT8):

Sigma Tests. provide an "early" alarm indicator that must stimulate searching possible assignable causes and, if the case, their identification and further elimination. One, Two and Three sigma tests point out a potential anomalous "trend" that "may" undertake assignable causes. Due to the high variance in software processes the faults highlighted by these tests could be numerous but less meaningful than in manufacturing contexts and might refer to "Occasional Change" (passing phenomena). The signals that Sigma tests detect may express a general behaviour determined by an assignable cause or passing phenomena.

Limit Tests. This class of tests point out an "Occurred Change" in process performance. They highlight the need to recalculate the control limits when the actual ones are inadequate, because they are too tiny or larger than required. In software process monitoring we represent a measurable characteristic that expresses a human related activity outcome (time spent, productivity, defect found during inspection etc.) on a control chart. Thus a "sequence" of points that Limit Tests detects means something has changed within the process (i.e. performance mean or variability). The process changes may refer to "New Mean" or "Decreased/Increased Variability".

Trend Tests. These tests highlight an "Ongoing Change": an ongoing phenomenon that reflects an ongoing shift that needs to be investigated. Typically a failure in this test class can be the result of both spontaneous and induced process improvement initiatives. Finally, according to the interpretations given, we are able to define the following function:

φ: *{Run-Test Failures}* ➔ *{Process Changes}*
"detected anomalies" "what happens"

Table 3. Run-Test Interpretation Details

SPC Theory		Process Changes	
Run-Test Failure	**Process Performance**	**Type**	**What Happens**
None	In Control	None	Nothing
RT1	Out of Control	Occasional	Early Alarm
RT2	Out of Control	Occasional	Early Alarm
RT3	Out of Control	Occasional	Early Alarm
RT4	Out of Control	Occurred	New Mean
RT5	Out of Control	Occurred	Increased Variability
RT6	Out of Control	Occurred	Decreased Variability
RT7	Out of Control	Occurred	New Sources of Variability
RT8	Out of Control	Ongoing	Ongoing Phenomena

For each run-test failure, φ is able to relate the "detected anomalies" to "what happens" within the process and suggest their cause. With reference to figure 4 of the explanatory case, we can make the following interpretations: all the anomalies related to RT1, RT2, RT3 are early alarms that announce a shift in process performance mean, as pointed out in the RT4 and an increased variability (RT5) with respect to the process performance threshold in use. These interpretations are coherent with what really happen in the RE project when (from data point 16 on) a more structured, formalized and transferable reverse engineering process was provided to the developers. This improvement (T1 in figure 2), initially, determined a process instability until the new reverse engineering process was internalized by developers.

2.4 Pattern 4: Sensibility ➔ Tuning Actions

Problem: Process Diversity means that a process performance varies between different organizations, different projects and also during the execution of a project [3]. A typical example is the so called "maturity effect", i.e. an improvement of human performances due to the experience acquired during process execution: better knowledge on the techniques in use, better confidence with the development tool etc. Hence, even though a good initial estimation of the process performance is done, it will not prevent estimation errors during project execution [10, 32, 33].

Solution: as emerges above it is difficult to correctly characterize process behaviour from the start to the end in terms of adopted baselines. As so, the SPC control limits need to be recalibrated according to relevant process performance changes and thus the sensibility of the monitoring activity has to be tuned continuously. The risk of not tuning sensibility is to miss anomalies as the result of using larger limits than necessary or having several false alarms due to narrow limits. In both cases it is necessary to: 1) identify when a relevant process performance change occurs; 2) tune the control model (i.e. recalibrate control limits) according to performance changes.

The solution follows from our experience and has been generalized in Table 4. More precisely, we have formalized the ψ that relates "what happens" in the process with "what to do" in terms of Tuning Actions needed to tune the sensibility of the monitoring activity.

$$\psi\text{: } \underbrace{\textit{\{Process Changes\}}}_{\text{"what happens"}} \rightarrow \underbrace{\textit{\{Tuning Actions\}}}_{\text{"what to do"}}$$

ψ is defined so that it determines the appropriate tuning actions needed to adapt the monitoring sensibility. At a glance with respect to the type of observed process changes, ψ can be explained as follows:

- if the process change is "Occasional", the process performance: (i) should be the same as in the past if assignable causes have been detected and removed or, if this is not the case, further observations are needed to exhibit the new process performance; (ii) is probably changing due to the fact that assignable causes were made part of the process. In this case further observations have to be collected. (iii) In both cases the control limits and the observed process characteristics remain the same.
- if the process change is "Occurred": (i) if process mean or variability are changed then the control limits should always be recalculated. A new set of data points that represents the new process performance have to be identified. The

candidate data points are those responsible for the test failure. (ii) if there is a new source of variability then the different sources must be identified, separated and tracked on different charts.

- if the process change is "Ongoing" additional observations are needed to determine reliable limits for the process because the actual observations express an ongoing change and thus, they cannot be used for determining new control limits. In this case "no action" is advisable.

Let us now apply these concepts to the explanatory case. Given the pattern we can see that RT1, RT2, and RT3 are classified as "occasional" process changes. They detect an early alarm, and according to ψ do not require any tuning action. On the other hand, RT4 and RT5 are classified as "occurred" process changes because the process mean has changed (RT4) and the process variability, considering the limits in use, has also increased (RT5) as can clearly be seen in figure 5. Indeed, the observed data points, from 16 on, no longer fall within the fixed limits. Consequently, in accordance to ψ and to the guidelines in table 4, new control limits must be calculated. Figure 6 shows the result of the tuning action, i.e. the new control limits calculated from data points 16-30. These interpretations also reflect what actually happened in the RE project where, following the improvement T1, described in section 2.3, developer performances sped up. As so, the limits in use (figure.5) were no longer appropriate and therefore recalibrated (figure 6) to reflect the new developer performances.

Table 4. Relationship between Process Changes and Tuning Actions

Process Changes		Tuning Actions
Type	What Happens	
None	Nothing	No Action
Occasional	Early Alarm	No Action
Occurred	New Mean	Identify new control limits
Occurred	Increased Variability	Identify new control limits
Occurred	Decreased Variability	Identify new control limits
Occurred	New Sources of Variability	Identify a new measurement object
Ongoing	Ongoing Phenomena	No Action

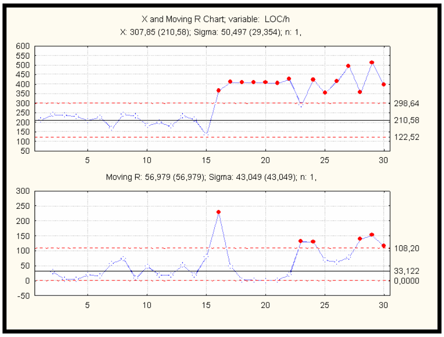


Fig. 5. RT4 and RT5 suggesting a shift in process performances

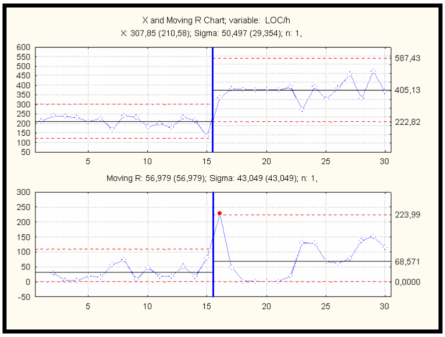


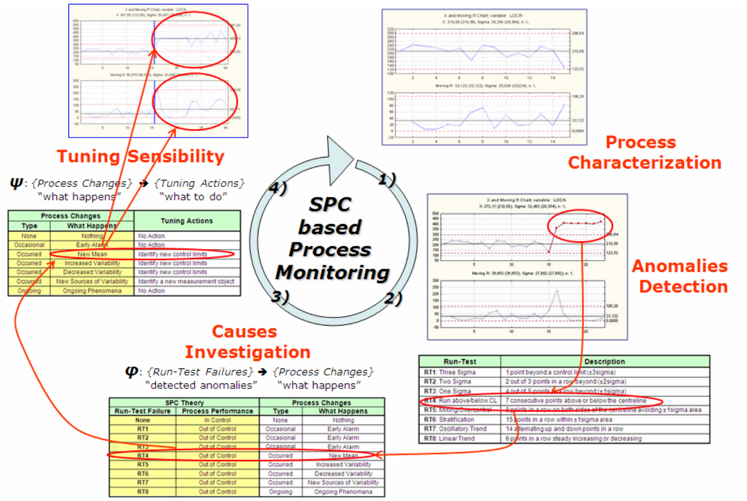
Fig. 6. New control limits calculated from data points 16-30

3 Discussion

The four patterns presented in section 2 represent a perspective for overcoming some non trivial problems related to software process monitoring. They can be seen as a set of lessons learned to use for guiding software process monitoring activities. Figure 7 summarizes the steps for applying the guidelines: first, process characterization is carried out, i.e. a process characteristic to monitor is observed over time, and data points are collected and used to determine upper and lower control limits on a control chart (Step 1); secondly anomaly detection occurs, i.e. each new data point observed is plotted on the chart, keeping control limits and central line the same, and the set of run tests (RT1…RT8) is executed and anomalies are detected each time a test fails (Step 2); then, causes investigation is carried out, i.e. the cause of the anomaly pointed out is investigated to provide an interpretation (Step 3). Finally, according to the process changes occurred and identified in the previous step, appropriate tuning actions are applied to tune the sensibility of the monitoring activity and adapt it to the new process performances (Step 4).

Adopting the guidelines during monitoring activities assures various benefits. First, it is possible to characterize process performances, even without having any previous knowledge, by deterministically determining a clear set of reference points. Note that lack of previous knowledge usually occurs for innovative processes, or for processes that are used in different contexts with different maturity levels, or refer to various application domains (technical rather than business). Moreover, in the SPC-based monitoring process, limits are not an expert-based estimation, but an actual expression of the process itself. Second, they provide a conceptual manner for defining process anomalies and, at the same time, an operational means for identifying them. Without such instruments (conceptual and operational) the interpretation of a trend rather than a single observation would completely rely on the project manager, who may not necessarily have the previous knowledge needed and thus, may neglect important events or focus on irrelevant ones resulting in ineffective monitoring. Third, they allow to adapt monitoring sensibility, not provided by the SPC-theory, to the actual process performances. Consequently, the guidelines can be used both for identifying spontaneous process changes (ex. learning effect) not induced by the project manager, and as means for verifying the validity of improvements accomplished by project manager. Finally, these guidelines represent a clear reference point, follow from explicit reasoning and are based on a solid theoretic model (SPC).

In the RE project, the monitored process characteristic was "*PERFORMANCE=NLOC/EFFORT*". By applying the guidelines during monitoring of the RE project we were able to: (i) characterize the process in use. Since the RE process had never been used in the industrial context, no past experience or baselines were available. The control limits represented a clear reference model and a way for building knowledge on the process in use; (ii) tune monitoring sensibility continuously, i.e. recalibrate the control limits according to process performance changes. (iii) identify all the known improvements (T1, T2, T3). Given the study was a simulation on legacy data, we were able to verify that the actual improvements occurred during the project were also identified during the simulation when adopting the monitoring guidelines. Similar results were obtained for the Restoration sub-project, not reported for space reasons.



ψ: {Process Changes} ➔ {Tuning Actions}
"what happens" "what to do"

Process Changes		Tuning Actions
Type	What Happens	
None	Nothing	No Action
Occasional	Early Alarm	No Action
Occurred	New Mean	Identify new control limits
Occurred	Increased Variability	Identify new control limits
Occurred	Decreased Variability	Identify new control limits
Occurred	New Sources of Variability	Identify a new measurement object
Ongoing	Ongoing Phenomena	No Action

Run-Test	Description
RT1: Three Sigma	1 point beyond a control limit (±3sigma)
RT2: Two Sigma	2 out of 3 points in a row beyond (±2sigma)
RT3: One Sigma	4 out of 5 points in a row beyond (±1sigma)
RT4: Run above/below CL	7 consecutive points above or below the centreline
RT5: Mixing/Overcontrol	8 points in a row on both sides of the centreline avoiding ±1sigma area
RT6: Stratification	15 points in a row within ±1sigma area
RT7: Oscillatory Trend	14 alternating up and down points in a row
RT8: Linear Trend	6 points in a row steady increasing or decreasing

φ: {Run-Test Failures} ➔ {Process Changes}
"detected anomalies" "what happens"

SPC Theory		Process Changes	
Run-Test Failure	Process Performance	Type	What Happens
None	In Control	None	Nothing
RT1	Out of Control	Occasional	Early Alarm
RT2	Out of Control	Occasional	Early Alarm
RT3	Out of Control	Occasional	Early Alarm
RT4	Out of Control	Occurred	New Mean
RT5	Out of Control	Occurred	Increased Variability
RT6	Out of Control	Occurred	Decreased Variability
RT7	Out of Control	Occurred	New Sources of Variability
RT8	Out of Control	Ongoing	Ongoing Phenomena

Fig. 7. SPC based Process Monitoring guidelines

4 Conclusion

In this paper we have synthesized four main issues to software process monitoring through a pattern based approach (where pattern is intended as a couple problem-solution) and suggest an SPC-based solution as result of our experience collected over the past years on industrial project data. Nevertheless, the application of SPC to process monitoring still represents an open issue with some limitations to keep in mind. As discussed in [14], there are many aspects related to software process measurement such as the difficulty of collecting metrics, their reliability and the selection of monitored process characteristics [16]; the violation of assumptions underlying SPC [19]; predominance of human factors that can impact on the SPC-theory and monitoring effectiveness [17]. All these aspects leave much space for subjective management decisions that can influence the success/failure of monitoring activities. Given these limitations, this paper is not intended as the solution to monitoring problems, nor as a silver bullet for applying SPC to software processes. Rather, it should be considered as a perspective on how SPC can contribute to practically solve some monitoring issues according to our experience from the trench . It can be seen as a first contribution for guiding practitioners towards a more disciplined use of SPC starting from understanding how it can really address software process monitoring. In this way operational, practical issues and pitfalls of SPC can be faced more systematically.

References

1. Florac, W.A., Carleton, A.D.: Measuring the Software Process: Statistical Process Control for Software Process Improvement. Addison-Wesley, Reading (1999)
2. Shewhart, W.A.: Statistical Method from the Viewpoint of Quality Control. Dover Publications, Mineola (1939) (republished 1986)

3. IEEE Software: Process Diversity. IEEE Software 17, 4 (July-August 2000), entire issue
4. IEEE Software: The Global View. IEEE Software (March-April 2001), entire issue
5. Grant, E.L., Leavenworth, R.S.: Statistical quality control. McGraw-Hill, New York (1980)
6. Wheeler, D.J., Chambers, D.S.: Understanding Statistical Process Control. SPC Press (1992)
7. Baldassarre, M.T., Boffoli, N., Caivano, D., Visaggio, G.: Managing SPI through SPC. In: Bomarius, F., Iida, H. (eds.) PROFES 2004. LNCS, vol. 3009, pp. 30–46. Springer, Heidelberg (2004)
8. Baldassarre, M.T., Caivano, D., Visaggio, G.: Software Renewal Projects Estimation Using Dynamic Calibration. In: 19th ICSM, pp. 105–115. IEEE Press, Amsterdam (2003)
9. Caivano, D.: Continuous Software Process Improvement through Statistical Process Control. In: 9th European CSMR, pp. 288–293. IEEE Press, Manchester (2005)
10. Baldassarre, M.T., Boffoli, N., Caivano, D., Visaggio, G.: Improving Dynamic Calibration through Statistical Process Control. In: 21st ICSM, pp. 273–282. IEEE Press, Budapest (2005)
11. Caivano, D., Lanubile, F., Visaggio, G.: Software Renewal Process Comprehension using Dynamic Effort Estimation. In: 17th ICSM, pp. 209–218. IEEE Press, Florence (2001)
12. Boffoli, N.: Non-Intrusive Monitoring of Software Quality. In: 10th European conference on Software Maintenance and Reengineering, pp. 319–322. IEEE Press, Bari (2006)
13. Baldassarre, M.T., Boffoli, N., Caivano, D., Visaggio, G.: SPEED: Software Project Effort Evaluator based on Dynamic-calibration. In: 22nd ICSM, pp. 272–273. IEEE Press, Philadelphia (2006)
14. Baldassarre, M.T., Caivano, D., Kitchenham, B., Visaggio, G.: Systematic Review of Statistical Process Control: an Experience Report. In: 11th EASE, pp. 119–129. BCS, Keele (2007)
15. Card, D.: Statistical Process Control for Software. IEEE Software, 95–97 (1994)
16. Sargut, K.U., Demirors, O.: Utilization of statistical process control in emergent software organizations: pitfalls and suggestions. Software Quality Journal 14, 135–157 (2006)
17. Eickelmann, N., Anant, A.: Statistical Process Control: What You Don't Measure Can Hurt You! IEEE Software, 49–51 (March/April 2003)
18. Weller, E., Card, D.: Applying SPC to Software Development: Where and Why. IEEE Software, 48–51 (May/June 2008)
19. Raczynski, B., Curtis, B.: Software Data Violate SPC's Underlying Assumptions. IEEE Software, 49–51 (May/June 2008)
20. Shewhart, W.A.: The Economic Control of Quality of Manufactured Product. D. Van Nostrand Company, New York (1931) (reprinted by ASQC Quality Press) (1980)
21. Paulk, M.C.: Applying SPC to the Personal Software Process. In: Proc. 10th ICSQ (2000)
22. Florac, W.A., Carleton, A.D., Bernard, J.R.: Statistical Process Control: Analyzing a Space Shuttle Onboard Software Process. IEEE Software, 97–106 (July/August 2000)
23. Jalote, P.: CMM in Practice: Processes for Executing Software Projects at Infosys. Addison-Wesley, Reading (1999)
24. Weller, E.: Applying Quantitative Methods to Software Maintenance. ASQ Software Quality Professional 3(1) (2000)
25. Jacob, A.L., Pillai, S.K.: Statistical Process Control to Improve Coding and Code Review. IEEE Software, 50–55 (May 2003)
26. Jalote, P.: Optimum Control Limits for Employing Statistical Process Control in Software Process. IEEE TSE 28(12), 1126–1134 (2002)

27. Nelson, L.: The Shewhart control chart - tests for special causes. Journal of Quality Technology 15, 237–239 (1984)
28. Nelson, L.: Interpreting Shewart X-bar Control Charts. J.of Quality Technology 17, 114–116 (1985)
29. Weller, E.F.: Practical Applications of SPC. IEEE Software, 48–55 (May/June 2000)
30. Gardiner, J.S., Montgomery, D.C.: Using Statistical Control Chart for Software Quality Control. In: Quality and Reliability Eng. Int'l., vol. 3, pp. 40–43. Wiley, Chichester (1987)
31. Park, Y., Choi, H., Baik, J.: A Framework for the Use of Six Sigma Tools in PSP/TSP. In: 5th International Conference SERA, Busan, Korea, pp. 807–814. Springer, Heidelberg (2007)
32. IEEE Software. Estimation 17(6) (November–December 2000)
33. Bohem, B.W.: Software Cost Estimation with COCOMO II. Prentice-Hall, Englewood Cliffs (2000)
34. Florac, W.A., Park, R.E., Carleton, A.D.: Practical Software Measurement: Measuring for Process Management and Improvement. Carnagie Mellon University (1997)

The How? When? and What? for the Process of Re-planning for Product Releases

Anas Jadallah[1], Ahmed Al-Emran[1], Mahmoud Moussavi[1], and Guenther Ruhe[1,2,3]

[1] Department of Electrical & Computer Engineering
Schulich School of Engineering, University of Calgary
[2] Department of Computer Science, University of Calgary
[3] Expert Decisions Inc, Calgary
{agjadall,aalemran,moussam,ruhe}@ucalgary.ca

Abstract. Volatility of features and dynamic change in stakeholders' needs often requires re-planning of an existing release plan to accommodate changes. H2W is a re-planning method that answers the questions of how, when, and what to re-plan of an existing product release strategy. For HOW, a greedy heuristic based on prioritization of candidate features is applied. A value-based re-planning approach is proposed for the WHEN question. For WHAT, a trade-off analysis between the degree of change related to the originally announced release plan and the improvement achievable by replacing existing features with more attractive ones is suggested. At each of the re-planning iterations, H2W either provides a new improved plan or states that an improvement does not exist. As a proof-of-concept, a case study is conducted.

Keywords: Release planning, Re-planning process, Change request, Volatile requirements, Decision support, Trade-off analysis.

1 Introduction and Motivation

When developing large scale software systems with hundreds or thousands of features, software engineers usually use iterative or incremental development where they adopt a scheduling and staging strategy, in which various parts of the system are developed at different times or rates, and integrated as they are completed. In market driven software development, features or requirements tend to change frequently due to many factors [8]:

- Requirements errors, conflicts, and inconsistency.
- Evolving customer/end-user knowledge of the system.
- Technical, schedule or cost problems.
- Changing customer priorities.
- Environmental changes.
- Organizational Changes.

Changing features or requirements is considered as one of the major causes for software product failure. A survey over 8000 projects undertaken by 350 US companies revealed that one third of the projects were never completed and one half

Q. Wang et al. (Eds.): ICSP 2009, LNCS 5543, pp. 24–37, 2009.

succeeded only partially. The third major source of the failure (11%) was changing requirements [20].

Release planning addresses the problem of assigning features (defined as sets of bundled requirements) to a sequence of releases such that the technical, resources, risk, and budget constraints are met [15]. Any need for having a well established change management process and handling re-planning of release plans is articulated in [1],[9],[16].

We proposed and evaluate a decision support method called H2W to address the "when?", "how?" and "what?" of re-planning. The structure of the paper is as follows. Section 2 describes related research on handling change requests in release planning. A formalization of the problem is contained in Section 3. Section 4 provides the necessary concepts for the H2W method proposed in Section 5. The method is illustrated in Section 6 by case study data. Overall validity of the method and discussion of the results is studied in Section 7. A summary and an outlook on future research are given in Section 8.

2 Related Work

A number of known methods and techniques have been proposed for product release planning (see for example [3],[7],[15],[19]). None of these methods are designed to address the issue of dynamically changing feature requests. Changing features can involve the arrival of new features or the change of existing features.

In the context of change request handling, the simplest strategy is to freeze change requests [21]. It mandates that an organization stop accepting new change requests after performing the initial activities in the release iteration. These changes are then introduced at the start of the next release. This approach ignores the fact that some of the changes may be very urgent and need to be implemented immediately.

Stark et al. [16] explored the relationship between requirements changes and software releases by looking at maintenance requirements volatility in a large software system with more than eight million lines of code spanning multiple languages, environments, and client organizations. The research tried to answer the following questions:

- How much volatility do the product releases experience?
- What kind of change is most commonly requested?
- When in the release cycle do requirements changes occur?
- Who requests requirements changes?
- How much effort is associated with implementing each requirement type?
- What schedule impact will a requirements change have?

For each question, data was measured, collected, and analyzed. Finally, a regression analysis was performed to find a relation between different factors affecting these change requests and the quality of the release planning problem. This regression model was used to facilitate communication among the project stakeholders when discussing requirements changes. The process of re-planning was not studied, though.

In the context of re-planning, AlBourae et al. [1] proposed a lightweight re-planning process for software product releases where change requests are accumulated and categorized during release iterations until we reach a specified level where resources are estimated for these new features and stakeholders are asked to vote for them. However, both the "when?" and the "what?" question were not addressed in this research.

3 Formalization of the Problem

The re-planning problem is based on different types of information related to the existing plan, the change requests, and the capacities and time interval of the release under investigation.

3.1 Release Information

The re-planning process happens in a given release period denoted by [T1,T2]. The implementation process of the planned release starts at t = T1. The planned release date is t = T2. Each feature has some estimated effort for its implementation and each release has a limited effort capacity CAP. The capacity limits the number of features that can be implemented in the proposed plan. The overall capacity CAP equals the product of (T2-T1)* AvgNDev, where AvgNDev denotes the average number of developers available over the planning period.

3.2 Existing Plan

At the beginning of the re-planning process, we assume a set F = {f(1)... f(N)} of features. The actual plan is described by the set of features selected from F for implementation in the given release. We describe the plan by a Boolean vector x with

$$x(n) = 1 \text{ iff feature } f(n) \text{ is selected from F for implementation} \quad (1)$$

This plan can be determined by any of the existing methods mentioned above. We will later see that our greedy re-planning method can be used as well to determine an initial plan.

Each feature f(n) for n = 1..N is characterized by a set of attributes. If attribute effort(n) describes the (estimated) effort for implementation of feature f(n), then we have

$$\Sigma_{n=1..N}\ x(n)\ \text{effort}(n) \leq \text{CAP} \quad (2)$$

In addition to effort, other attributes that we consider are risk(n) and value(n) that represents a risk and a benefit value of the feature, respectively.

3.3 Change Requests

In the process of implementation of a once agreed upon plan, change request might occur. A change request relates to changing an already accepted feature (some of the data of its related attributes) or requesting a new feature. The set of change request is dynamically changing, so we need to describe them in dependence of time.

We define by CR(t) the set of all change requests from the beginning of the release period (t = T1) until point in time t $\in$ (T1, T2]. The change requests themselves are denoted by f(N+1), f(N+2), ... f(N_t) where N_t denotes the number of change requests until point in time t. Each change request f(n), n = N+1 … N_t is characterized by four attributes called risk(n), value(n), time(n) and effort(n).

Both risk and the feature's value (from a stakeholder viewpoint) are classified on a nine-point scale with values as specified in Table 1. A change request f(n) is arriving at point in time t denoted by t = time(n) and is estimated to consume effort in the amount of effort(n). Both the risk and the value estimation are assumed to be the result of stakeholder (and expert) evaluation as described in [10]. Experts can estimate the level of risk for features from previous project history. For estimation of effort, a variety of known methods and techniques can potentially be applied. A method for estimation in consideration of change was studied in [12].

Table 1. Nine point scale for evaluating effort, risk and value of features

risk(n), value(n)	Interpretation
1	Extremely low
3	Low
5	Average
7	High
9	Extremely high

4 Key Solution Concepts

4.1 Distance-to-Ideal-Point Minimization

We borrow the concept of "Distance-to-ideal-point minimization" known from multi-criteria optimization [17] for determining the most attractive features for replacement of less attractive ones. The ideal solution is an artificial solution defined in the criterion space, each of whose elements is the optimum of a criterion's value. As this artificial solution can not be achieved in most of the cases, the goal is to come as close as possible to the ideal solution. The distance can be measured, for example, by the Euclidean distance. The solution being closest to the ideal one is considered to be the best compromise and the most promising one to pursue.

In our case, we apply the concept of "Distance-to-ideal-point minimization" to the three-dimensional space defined by the feature attributes risk, effort and value. For simplicity reasons, the three attributes are treated as being defined on an interval scale. The solutions minimizing the distance to the ideal case of having extremely low effort, extremely low risk and extremely high value are considered to be the top candidates for inclusion into the current release plan.

4.2 Greedy Method for Release Planning

Greedy algorithms are widely applied to iterative solution technique aimed at finding a good global solution by determining the local optimum choice at each stage [4]. The key principle of greedy release planning is to elaborate the features top down according to their overall priority. In our approach, the priority is higher the lower the distance is to the ideal point, as defined above.

The optimization process starts with the feature with the highest overall priority. This process is continued with the next best feature. Each new feature is assigned to the current release as long as the resource capacities are not exceeded. If the addition of any of the new features would violate any of the capacity constraints, then the feature is dismissed and the next best feature is considered for inclusion. The process terminates when no further feature can be added to the existing set.

5 H2W Re-planning Method

The method applies the above concepts in an iterative manner and considers the most recently created plan as a baseline for the next re-planning iteration. The workflow of the overall method is shown in Figure 1.

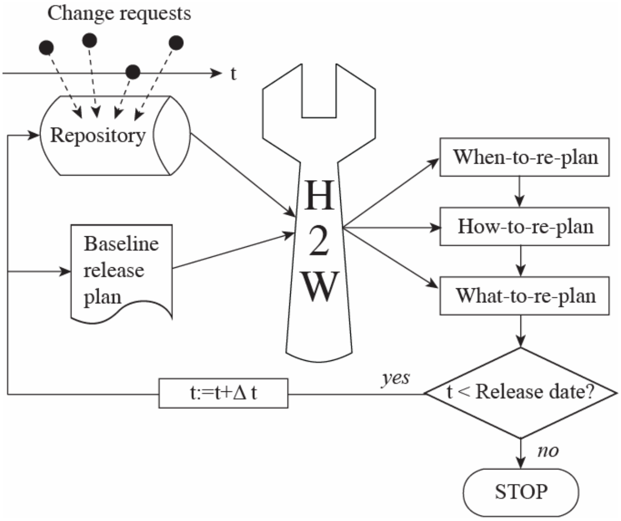


Fig. 1. Workflow of the H2W method

The decision about when to re-plan is based on some threshold related to the accumulated value of all the arriving features. We consider a value based threshold here which is called V-THRESHOLD. The actual value is project and context specific and is defined by the product manager, taking into account his or her former experience, as well as the current business and market conditions.

During the "how to re-plan?" part, candidate features are ranked based on their distance to the ideal point. Greedy optimization is applied adding the most attractive features to the capacities remaining. Finally, "what to re-plan?" determines the best compromise between creating additional value and changing the existing plan. The relative increase in value is compared to the degree of change of the release plan for a sequence of exchanges of varying cardinality of the set of removed (from the baseline plan) features. The point of intersection between these curves determines the number of features to be exchanged from the given plan. This process is further illustrated in the case study example of Section 6. The three key steps of the method are described in more detail below in a pseudo-code representation.

Step 1: When to re-plan?

```
Initialize TotalNewValue to 0 // accumulative added values of new change requests.
Initialize DoReplanning to false // determine when we can do re-planning

For     n = N+1 to $N_t$ // for each new change request
          TotalNewValue = TotalNewValue + value(n)
Endfor
If      ( TotalNewValue >= V-THRESHOLD )
Then    Set DoReplanning = true  // start re-planning
Else    Set DoReplanning = false // don't re-plan
Endif
```

Step 2: How to re-plan?

```
Initialize CandidateFeatures to empty // list containing the best candidate features
Initialize AvgNDev // Average no. of developers available over the planning period
Initialize Features to empty // containing unimplemented features & change requests

Set     RemCAP = (T2-t) * AvgNDev
For     n = 1 to $N_t$
          Normalize effort(n) in a scale of {1,..,9} and store it in normEff(n)
          If      ((f(n) is not implemented ) OR (f(n) is change request))
          Then    Set Dis(n) = Sqrt$((value(n)-9)^2+(normEff(n)-1)^2+(risk(n)-1)^2)$
                  Features.add(f(n))
          Endif
Endfor
Features.sortBy(Dis)
While   ( RemCAP > 0 )
          Set     featureFound = false
          For     n = 1 to Features.Size()
          If      (effort(Features[n]) < RemCAP )
          Then    CandidateFeatures.add(f(n))
                  RemCAP=RemCAP – effort(n)
                  Features.remove(f(n))
                  Set featureFound = true
                  Break // stop the closest for loop
          Endif
          Endfor
          If      (featureFound = false )
          Then    Break // stop the while loop
          Endif
Endwhile
```

Step 3: What to re-plan?

Assumption 1: The current baseline plan includes m features.
Assumption 2: List of features RF = {rf(1) …rf(P)} to be removed from baseline plan as a result of Step 2.

Assumption 3: The features of RF are arranged in decreasing order in terms of their distance to the ideal point.

```
For p = 1 to P
    Determine the best replacement for the set of existing features rf(1)... rf(p) with new features → features nf(1) ... nf(q)
    Determine added value
    $AdValue(p) = \Sigma_{n=1..q}$ value(nf(n)) - $\Sigma_{n=1..p}$ value(rf(n))
    $Stability (p) = 1 - p/m$
Endfor
```

Determine p* to be the index such that the two points (p*, AdValue(p*)) and (p*, Stability(p*)) are closest to each other.// This represents the best compromise.

H2W applies the three steps in an iterative manner and considers the most recently created plan as a baseline for the next re-planning iteration. The process of arriving features with their increasing cumulated value is illustrated in Figure 2. Each step of the curve corresponds to the arrival of a new feature adding some new value. However, only a portion of these new features is actually incorporated into the existing plan. In the illustration, we have assumed two re-planning steps. Their timing within the release interval [T1,T2] is highlighted by the two black arrows at the x-axis.

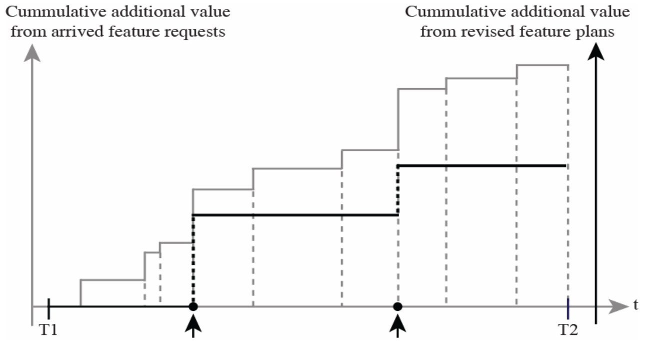


Fig. 2. Illustration of the impact of arriving features on additional value

6 Illustration of H2W by a Case Study Example

In this section, we provide a case study based on real-world data coming from planning a project in the decision support tool ReleasePlanner® [14] and illustrate the usefulness of this method.

6.1 Release Information and Baseline Plan

The case study includes:

- A set of fifty features, F = {f(1), f(2), ..., f(50)} with their corresponding attributes shown in Table 3 of the Appendix;
- Effort capacity, CAP = 1500;
- Average number of developers, AvgNDev = 5;

- Release start time, T1 = 0;
- Release end time, T2 = 300 (i.e., CAP/AvgNDev).

Table 3 also provides step-by-step data for baseline planning. We first prioritized all features (c.f., Table 3, Column 2) after normalization of effort(n) in a 9 point scale (denoted by normEff(n)) and application of "Distance-to-Ideal-Point minimization" as discussed in Section 4.1. Then we apply the greedy planning method (c.f., Section 4.2) that results in 39 features to be accommodated in the next release within given capacity CAP. A total of 11 features are rejected from the baseline plan: f(3), f(10), f(12), f(14), f(18), f(25), f(29), f(30), f(34), f(35), and f(39). A release value (i.e., stakeholders' satisfaction) of 236 can be achieved that can be computed by adding up value(n) for all 39 features that be accommodated in the release.

6.2 Change Requests

We simulate twenty change requests (50% of baseline features): f(51), f(52), ..., f(70) are arriving at different points in time within the release duration as mentioned in Section 3.3. The corresponding attributes are also randomly generated within the same range (e.g., in a 9 point scale for risk(n) attribute) in the set of features. The arrival time for the change requests, time(n), is equally distributed over the time period T1 to T2. Table 4 (see Appendix) provides the information about the simulated data.

6.3 STEP 1: When to Re-plan?

As expressed in Section 4, we consider a value based threshold called V-THESHOLD as a trigger for re-planning. Once the accumulated value(n) of the incoming change requests (or new features) reaches or exceeds V-THRESHOLD, the re-planning process is initiated.

For our case study, we decided to set this threshold value to 25% of the baseline release value. In Section 6.1, we have seen that the release value of the baseline plan is 236. Thus, 25% of this value becomes 59. Thus, whenever the accumulated value(n) for incoming change requests reaches or exceeds 59, the re-planning event will occur.

The next step is to continually check the time period of 0 to 300 to see if re-planning is triggered by the threshold. In our case, as soon as change request f(62) arrives at time 194 (c.f., Table 4; highlighted) the accumulated value(n) becomes 62, i.e., exceeds V-THRESHOLD and triggers the first re-planning iteration.

6.4 STEP 2: How to Re-plan?

At the 2nd stage, we know "when" to re-plan but the question now becomes how to do that. The answer lies in applying the same greedy optimization as performed in case of baseline planning. However, this time we consider a new "candidate list" that includes the change requests (or new features), features that are planned but have not been implemented in the release, and the rejected features, while we perform our baseline planning. Since some time has passed, 29 features have already been implemented as shown in Table 4. They are: f(42), f(33), f(32), f(7), f(23), f(9), f(5), f(17), f(8), f(16), f(24), f(31), f(44), f(37), f(4), f(48), f(41), f(13), f(21), f(22), f(47), f(50), f(19), f(49), f(15), f(38), f(2), f(20), and f(26). Since, f(26) is under construction at time 194, we let it to be finished before selecting features or change requests from the candidate list for the rest of the release.

At this re-planning point, the effort capacity (CAP) already consumed by the implemented features is 990 and we have (1500 – 990) = 510 effort capacity remaining for the rest of the release development. We prepare our candidate list by including the rest of the 10 features that were planned in the baseline release but not yet implemented, 12 change requests that have arrived so far and 11 features that have been rejected at the time of baseline planning. We normalize the effort(n) attribute of all items in the candidate list in a 9 point scale to normEff(n) and apply the concept of "Distance-to-Ideal-Point" to prioritize each of the candidates. Table 4 shows the prioritized candidate list with each item's distance Dis(n) from the ideal point.

Then the method takes the most attractive features, one-by-one, from the top of the list until it is out of available capacity. In our case, the features that are selected from the list for the rest of the release development are: f(62), f(51), f(59), f(46), f(27), f(45), f(55), f(36), f(53), and f(57) (see Table 4). The release value of the revised plan becomes 249. If we compare the revised plan with the baseline then we can see features f(28), f(1), f(11), f(6), f(43), and f(40) of the baseline plan are exchanged with change requests f(55), f(57), f(51), f(62), f(59), and f(53) for an overall gain of (249 – 236) = 13, in terms of release value.

6.5 STEP 3: What to Re-plan?

In Step 3, we determine the best compromise between gaining additional value from exchanging features and accepting some instability from the same effort. The two extremes in this respect would be no changes at all (highest degree of stability) and replacing features f(1), f(6), f(11), f(28), f(40), and f(43) by five new features: f(51), f(53), f(55), f(57), f(59), and f(62). The results of a more detailed trade-off analysis are summarized in Table 2.

Table 2. Evolution of feature replacements for the five features determined in Step 2

Number of features eliminated from baseline	Set of eliminated features	Set of replacement features	Added value
1	{28}	{55}	3
2	{28,1}	{55,57}	3+3
3	{28,1,11}	{55,57,51}	3+3+3
4	{28,1,11,6}	{55,57,51,62}	3+3+3+2
5	{28,1,11,6,43}	{55,57,51,62,59}	3+3+3+2+1
6	{28,1,11,6,43,40}	{55,57,51,62,59,53}	3+3+3+2+1+1

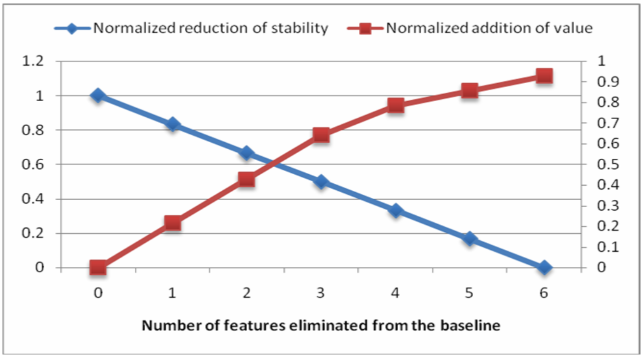


Fig. 3. Trade-off analysis between added value and stability of the plan

Based on analysis of the trade-off situation as shown in Figure 3, it was recommended that the best compromise would be the elimination of three features from the baseline. The features to be eliminated are f(1), f(11) and f(28). Their recommended replacement are features f(51), f(55) and f(57) and the added value of this replacement is 9.

7 Discussion of the Results

The re-planning process accommodates a number of essential parameters which have not been considered yet in the existing literature. We reconsider not only the new change requests, but also model changes in the features already in consideration for releases. The reason for this consideration is that all the planning information is time-dependent, and priorities or effort estimates may need to be corrected at a later point in time.

The illustrative example, partially based on real-world data, served as a "proof-of-concept" for the proposed re-planning approach. We have performed a more comprehensive empirical analysis analyzing a set of 1000 randomly generated scenarios for re-planning. The detailed results of that under the additional aspect of using predictive effort and defect models have been studied in [2]. From running the case study, we have made a number of observations:

- Re-planning is important, as otherwise significant information is not taken into account and possible gain of value is ignored. For the case study data, an improvement (measured as the overall value of the plan) of about 4% would have been ignored from one re-planning effort.
- In dependence of the frequency of re-planning, substantial changes in the structure and the overall value of the plans occur. For the case study, just three features have been exchanged to achieve the 4% gain in customer satisfaction.
- The method allows easy and quick re-planning for problems of at least 50 + 20 features.
- Decisions about re-planning of an existing solution can not just be made by looking at the added value. There is a trade-off situation relating the added value to the loss of stability.

We do not claim external validity for the observations above. More comprehensive empirical evaluation is needed for this purpose. There are also limitations with the internal validity of the results. This is caused by simplifications made on the underlying model. Firstly, we just consider effort as the attribute characterizing the amount of implementation needed to implement the feature. However, in reality, effort is composed of different types of resources. In a more fine-grained model, different types of resources would need to be considered.

A second assumption is related to the same issue. In case of any re-planning, we need information about which features are already finished, which ones are in progress, and which ones are not yet started. Consequently, some form of operational planning needs to be in place to track this. We have made the simplifying assumption that the features are pursued in the order of priority (highest rated feature comes first) with all effort spent on the feature in progress. In other words, we have the model of serial implementation of all the features. This again is a simplification which allows re-planning in a reasonable amount of time and with a reasonable amount of computational effort.

Finally, to keep the method reasonably light-weight, we applied greedy optimization known to deliver good, but not necessarily optimal results. As part of the distance-from-ideal-point minimization, effort estimates were mapped to a nine-point scale when used as part of the distance computation.

8 Summary and Future Research

Software release planning is a problem with significant impact on the success or failure of software product development. A systematic approach is likely to generate plans achieving higher business value. Planning of product releases provides information to the stakeholders, customers and the development team about which features are planned to be delivered at which release. As there is dynamic change in the data, priorities and even objectives of the planning, plans need to be adjusted dynamically to fit reality. The process of re-planning existing release plans is an important aspect of responding to change.

The main contribution of this research is the provision of a method called H2W for re-planning of existing product release plans. The emphasis was the efficiency allowing easy-to-use procedures to support the “how?”, “when?” and “what?” questions of decision-making. A more comprehensive empirical evaluation and an industrial evaluation is one of the topics of future research.

A number of other questions need to be addressed in this context. More and more, planning for product releases takes non-functional requirements into account (see [3]). Consequently, this aspect also needs to considered for the re-planning process. We also need to address the issue of balancing effort between creating additional functionality and stabilizing the existing code and fixing detected defects in the context of re-planning (see [2]).

The threshold value is one of the critical parameters of the method. We plan to investigate the sensitivity of solutions depending on this value and the use of additional or other thresholds (e.g., for the cumulative risk, the cumulative effort of the arriving features, and the defects slippage density).

We do not know exactly the quality of the proposed plan. This refers to the industrial evaluation, but also to a comparison with a more sophisticated planning (and re-planning) process looking at optimal solutions at each iteration. Even though this is not always necessary, the comparison would allow one to better judge the quality of the H2W re-plans. In this context, more fine-grained resource consumption models (as studied in [10]), including different types of resources, would be considered as well.

Acknowledgement

Part of the work presented was financially supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada under Discovery Grant no. 250343-07. Ahmed Al-Emran would like to thank iCORE, NSERC, and the Killam Trust for their postgraduate/pre-doctoral scholarships and to ExtendSim for supporting implementation of the research prototype. The authors would also like to thank Jim McElroy and Kornelia Streb for editorial input on the paper.

References

1. AlBourae, T., Ruhe, G., Moussavi, M.: Lightweight Re-planning of Software Product Releases. In: 14th IEEE International Requirements Engineering Conference Minneapolis/St. Paul, Minnesota, USA (2006)
2. Al-Emran, A., Jadallah, A., Moussavi, M., Paikari, E., Pfahl, D., Ruhe, G.: Functionality versus Quality: Application of Predictive Models for Re-planning of Product Releases. Submission for International Conference on Predictor Models PROMISE 2009, Vancouver (2009)
3. Bagnall, A.J., Rayward-Smith, V.J., Whittley, I.M.: The Next Release Problem. Information and Software Technology 43, 883–890 (2001)
4. Cormen, T.H., Leiserson, C.E., Rivest, R.L.: Introduction to Algorithms, Clifford Stein. MIT Press, Cambridge (2007)
5. Greer, D., Ruhe, G.: Software Release Planning: An Evolutionary and Iterative Approach. Information and Software Technology 46, 243–253 (2004)
6. Jadallah, A.: `http://www.ucalgary.ca/~agjadall/ICSP2009`
7. Jung, H.W.: Optimizing Value and Cost in Requirements Analysis. IEEE Software 15, 74–78 (1998)
8. Kontonya, G., Sommerville, I.: Requirements Engineering: Processes and Techniques. Wiley, Chichester (1998)
9. Loconsole, A.: Empirical Studies on Requirement Management Measures. In: Proceedings - 26th International Conference on Software Engineering, ICSE 2004, Edinburgh, United Kingdom (2004)
10. Ngo-The, A., Ruhe, G.: Optimized Resource Allocation for Software Release Planning. IEEE Transactions on Software Engineering 35, 109–123 (2009)
11. Nurmuliani, N., Zowghi, D., Fowell, S.: Analysis of Requirements Volatility During Software Development Life Cycle, Melbourne, Australia (2004)
12. Ramil, J.F.: Continual Resource Estimation for Evolving Software. In: Proceedings Conference on Software Maintenance, pp. 289–292 (2003)
13. Regnell, B., Svensson, R.B., Olsson, T.: Supporting Road-mapping of Quality Requirements. IEEE Software 25, 42–47 (2008)
14. ReleasePlanner, Expert Decisions Inc., `http://www.releaseplanner.com`
15. Ruhe, G., Saliu, M.O.: The Art and Science of Software Release Planning. IEEE Software 22, 47–53 (2005)
16. Stark, G., Skillicorn, A., Ameele, R.: An Examination of the Effects of Requirements Changes on Software Maintenance Releases. Journal of Software Maintenance: Research and Practice (1999)
17. Steuer, R.E.: Multiple Criteria Optimization: Theory, Computation, and Application. John Wiley, New York (1986)
18. Strens, M.R., Sugden, R.C.: Change Analysis: A Step towards Meeting the Challenge of Changing Requirements. In: Proceedings of the IEEE Symposium and Workshop on Engineering of Computer Based Systems, pp. 278–283 (1996)
19. Van den Akker, M., Brinkkemper, S., Diepen, G., Versendaal, J.: Software Product Release Planning through Optimization and What-If Analysis. Information and Software Technology 50, 101–111 (2008)
20. Van Lamsweerde, A.: Requirements Engineering in the Year 2000: a research perspective. In: Proceedings of the 22nd International Conference on Software Engineering (ICSE 2000), Limerick, Ireland (2000)
21. Wiegers, K.E.: Software Requirements: Practical Techniques for Gathering and Managing Requirements Throughout the Product Development Cycle, 2nd edn. Microsoft Press (2003)

Appendix

Table 3. Performing baseline planning

Initial Feature Set				Prioritized Features based on "Distance-to-Ideal-Point"					Baseline Release Plan			
f (n)	effort (n)	value (n)	risk (n)	f (n)	normEff (n)	value (n)	risk (n)	Distance (n)	f (n)	start (n)	end (n)	Occupied CAP
1	16	5	9	42	1	9	1	0.00	42	0	2	10
2	16	2	3	33	1	9	3	2.00	33	2	3	15
3	17	2	8	32	1	9	4	3.00	32	3	5	25
4	30	6	6	7	1	7	4	3.61	7	5	6	30
5	29	6	3	23	4	7	1	3.61	23	6	14	70
6	23	3	8	9	1	9	5	4.00	9	14	16	80
7	4	7	4	5	3	6	3	4.12	5	16	22	110
8	65	9	2	17	4	6	2	4.36	17	22	30	150
9	9	9	5	8	6	9	2	5.10	8	30	43	215
10	31	3	9	16	2	7	6	5.48	16	43	47	235
11	75	6	7	24	2	4	3	5.48	24	47	51	255
12	45	1	4	31	4	9	6	5.83	31	51	60	300
13	60	6	4	44	4	9	6	5.83	44	60	67	335
14	45	1	5	37	5	7	5	6.00	37	67	77	385
15	45	3	2	4	3	6	6	6.16	4	77	83	415
16	17	7	6	48	7	7	1	6.32	48	83	98	490
17	40	6	2	41	1	5	6	6.40	41	98	100	500
18	85	1	4	13	6	6	4	6.56	13	100	112	560
19	22	2	1	21	3	6	7	7.00	21	112	119	595
20	83	9	4	22	3	3	4	7.00	22	119	125	625
21	31	6	7	47	4	3	3	7.00	47	125	133	665
22	28	3	4	50	1	9	8	7.00	50	133	135	675
23	38	7	1	19	2	2	1	7.07	19	135	140	700
24	20	4	3	49	5	9	7	7.21	49	140	151	755
25	80	1	4	15	5	3	2	7.28	15	151	160	800
26	75	9	6	38	1	2	3	7.28	38	160	162	810
27	27	4	7	2	2	2	3	7.35	2	162	166	830
28	95	6	1	20	8	9	4	7.62	20	166	183	915
29	70	2	9	26	7	9	6	7.81	26	183	198	990
30	55	1	7	46	3	2	4	7.87	46	198	204	1020
31	41	9	6	27	3	4	7	8.06	27	204	210	1050
32	10	9	4	45	5	9	8	8.06	45	210	219	1095
33	5	9	3	36	5	5	7	8.25	36	219	228	1140
34	80	1	2	28	9	6	1	8.54	28	228	247	1235
35	60	9	9	43	4	1	2	8.60	43	247	255	1275
36	45	5	7	40	9	7	4	8.78	40	255	273	1365
37	50	7	5	1	2	5	9	9.00	1	273	277	1385
38	10	2	3	11	7	6	7	9.00	11	277	292	1460
39	100	3	7	6	3	3	8	9.43	6	292	297	1485
40	90	7	4	12	5	1	4	9.43				
41	10	5	6	35	6	9	9	9.43				
42	10	9	1	14	5	1	5	9.80				
43	36	1	2	3	2	2	8	9.95				
44	34	9	6	10	3	3	9	10.20				
45	45	9	8	34	8	1	2	10.68				
46	29	2	4	30	5	1	7	10.77				
47	40	3	3	18	8	1	4	11.05				
48	75	7	1	25	8	1	4	11.05				
49	55	9	7	39	9	3	7	11.66				
50	6	9	8	29	7	2	9	12.21				

Table 4. Handling a re-planning scenario

Change Requests						Completed Baseline Plan at the Re-planning Point				The rest of the release plan after re-planning iteration		
f (n)	effort (n)	value (n)	risk (n)	time (n)	Cumulative value (n)	f (n)	start (n)	end (n)	Occupied CAP	f (n)	start (n)	end (n)
51	83	9	3	10	9	42	0	2	10	62	198	203
52	65	2	4	11	11	33	2	3	15	51	203	220
53	96	8	3	42	19	32	3	5	25	59	220	227
54	96	5	8	43	24	7	5	6	30	46	227	233
55	45	9	8	68	33	23	6	14	70	27	233	239
56	67	2	8	77	35	9	14	16	80	45	239	248
57	70	8	7	105	43	5	16	22	110	55	248	257
58	67	3	7	118	46	17	22	30	150	36	257	266
59	31	2	2	142	48	8	30	43	215	53	266	286
60	71	4	9	150	52	16	43	47	235	57	286	300
61	47	5	8	165	57	24	47	51	255			
62	22	5	5	194	62	31	51	60	300			
63	73	8	4	204	70	44	60	67	335			
64	67	3	2	210	73	37	67	77	385			
65	97	4	6	239	77	4	77	83	415			
66	77	4	6	248	81	48	83	98	490			
67	90	9	6	275	90	41	98	100	500			
68	19	4	8	281	94	13	100	112	560			
69	83	3	9	288	97	21	112	119	595			
70	23	4	6	292	101	22	119	125	625			
						47	125	133	665			
						50	133	135	675			
						19	135	140	700			
						49	140	151	755			
						15	151	160	800			
						38	160	162	810			
						2	162	166	830			
						20	166	183	915			
						26	183	198	990			

Prioritized Candidates at Re-planning Point Based on "Distance-to-Ideal-Point"					Prioritized Candidates at Re-planning Point Based on "Distance-to-Ideal-Point"				
f (n)	normEff (n)	value (n)	risk (n)	Distance (n)	f (n)	normEff (n)	value (n)	risk (n)	Distance (n)
62	2	5	5	5.74	6	3	3	8	9.43
51	8	9	3	7.28	12	5	1	4	9.43
59	3	2	2	7.35	35	6	9	9	9.43
46	3	2	4	7.87	14	5	1	5	9.80
27	3	4	7	8.06	3	2	2	8	9.95
45	5	9	8	8.06	10	3	3	9	10.20
55	5	9	8	8.06	58	7	3	7	10.39
36	5	5	7	8.25	34	8	1	2	10.68
53	9	8	3	8.31	30	5	1	7	10.77
28	9	6	1	8.54	18	8	1	4	11.05
57	7	8	7	8.54	25	8	1	4	11.05
43	4	1	2	8.60	60	7	4	9	11.18
40	9	7	4	8.78	54	9	5	8	11.36
1	2	5	9	9.00	56	7	2	8	11.58
11	7	6	7	9.00	39	9	3	7	11.66
61	5	5	8	9.00	29	7	2	9	12.21
52	6	2	4	9.11					

Overcoming the First Hurdle: Why Organizations Do Not Adopt CMMI

Nazrina Khurshid, Paul L. Bannerman, and Mark Staples

NICTA, Australian Technology Park, Eveleigh, NSW, Australia
School of Computer Science and Engineering, University of NSW, Australia
{nazrina.khurshid,paul.bannerman,mark.staples}@nicta.com.au

Abstract. This paper further examines why some software development organizations decide not to adopt CMMI by replicating an earlier Australian study in another country. The study examines data collected from the efforts of three consulting firms to sell a CMMI Level 2 program subsidized by the Malaysian government. The most frequently cited reasons for not adopting CMMI were: the program was too costly; the companies were unsure of the benefits; the organization was too small; and/or the organization had other priorities. The Malaysian study extends and generally supports the Australian study (differences were found in the frequency ordering of reasons and two new reason categories had to be introduced). It also adds to our understanding of CMMI adoption decisions. Based on the results, we conclude that to achieve broader impact in practice, software process improvement (SPI) researchers need to develop a stronger cost-benefit analysis for SPI, recognising it as a business investment rather than just a product or process quality improvement technique, and provide flexible entry options to enable more companies of difference sizes to take the adoption leap.

Keywords: Software Process Improvement, SPI, CMMI, adoption.

1 Introduction

Software development processes have attracted much attention in research and practice due to quality and reliability concerns, outsourcing opportunities and expanding complexity resulting from marketplace demands [9]. Software process improvement offers potential benefits of standardized processes, higher product quality, earlier detection of defects, reduced time to market, and lower post-release defects [9]. However, slow adoption of SPI standards and methods is an issue.

Various standards and methods have been published to facilitate software process improvement and many software developing companies have adopted practices to improve their processes. Examples include process capability models such as CMMI [5] and ISO/IEC 15504 (SPICE) that have been developed to assist software developing companies build processes that subscribe to world's best practices [1]. There have been many studies on SPI, but most have examined the experiences of organizations that have already adopted and successfully implemented SPI models rather than those that did not overcome the first hurdle of taking the decision to adopt.

Q. Wang et al. (Eds.): ICSP 2009, LNCS 5543, pp. 38–49, 2009.

While there is strong interest in SPI, there is a lack of knowledge of the motivators and de-motivators of SPI adoption, especially at the organizational level. While it is essential to understand the benefits of SPI, the factors that contribute to the success and failure of SPI initiatives, and the sustainment of SPI practices, it is also important to understand why organizations decide to adopt or not adopt SPI in the first place. This will enable organizations to better understand considerations that might be relevant in taking the decision to adopt SPI and help researchers to improve SPI methods by making them more relevant and accessible to prospective users [7].

Only one prior study has examined the reasons why organizations decided not to adopt CMMI [22]. This study found that organization size and SPI cost were significant barriers to adoption. This paper reports a study that replicates the earlier study in a different country (Malaysia rather than Australia). Replication studies play a key role in the scientific method and in empirical software engineering [20], and help to build knowledge and establish the external validity of a study's findings. Our study also contributes further insight into the problem of SPI adoption and suggests directions for future progress.

The paper is organized as follows. Section 2 briefly backgrounds prior research on SPI. Section 3 outlines the research methodology. Section 4 details the data analysis and study results. Section 5 discusses the limitations of the study and its implications before conclusions are reached in the last section.

2 Prior Research

Software process improvement has attracted considerable research attention in the literature. While this research has been dominated by CMM-based methods, it has been criticized as narrowly prescriptive rather than descriptive or reflective [11].

One stream of this research has focused on developing an understanding of why SPI adoption has been slow, compared to the perceived potential benefits [12], especially in small to medium sized organizations (SMEs). In the latter case, some studies have focused exclusively on SPI adoption success factors in small organizations (for example, [10], [25]). Curiously, a recent systematic review found that organizations mostly adopt CMM-based SPI methods to improve project performance and product quality more than to improve processes [23]. Furthermore, the reasons were unrelated to size. Another recent study found that SPI methods are implemented reactively and reluctantly because of their associated costs [6].

Other studies within this stream have focused on the SPI de-motivators of software practitioners [3], [17]. In a recent retrospective, Humphrey suggests that acceptance has been slow due to a combination of factors including: people are too busy; they do not believe SPI will work; they believe that a better tool or method is likely to come along; management is unwilling to make the investment; and/or SPI is not supported unless it addresses a current critical issue or business crisis [14].

By contrast, our interest is in understanding why organizations (as opposed to individuals) do not adopt SPI methods at all (specifically focusing on CMMI adoption). That is, why do some organizations not get past the first decision hurdle to commit to a formal SPI program?

Only one prior study has investigated this question. The study examined the reasons given for not adopting CMMI to a company selling CMMI appraisal and improvement services to 40 Australian software-developing organizations [22]. The most frequent reasons were: the organization was small; the services were too costly; the organization had no time; and the organization was using another SPI method. Small organizations tended to say that adopting CMMI would be infeasible (termed 'could not' reasons by the study), rather than that it would be unbeneficial (termed 'should not' reasons). The current study aims to empirically validate these findings by replicating the prior study in a different country (Malaysia).

Choosing Malaysia enables testing contrasting socio-cultural differences that may affect adoption of Western turnaround methods. Organizational behaviour varies between national cultures [21]. Alstrom and Bruton [2] argue that the traditional Asian characteristics of fear of failure and low risk-taking, together with different institutional environment and implementation strategies, may impact best practice adoption. Finding support for the previous study would suggest that these national differences may not be significant contingencies, as well as strengthen the external validity and generalizability of the findings.

The study investigates two research questions:

RQ1: Why do organizations decide not to adopt CMMI?

RQ2: How are these reasons related to the size of the organization?

3 Research Method

The study replicates the research questions and analysis method of a prior Australian study [22] using data from a different country. For each study, data collection was completed independently of knowledge of the other study. The occasion of the current study is that, in 2005, the Malaysian government introduced a funding assistance program to enable local SMEs to adopt and implement CMMI Level 2 to increase their export competitiveness. The study aims to investigate and understand the reasoning of the companies that decided not to participate in the program.

Based on the results of the previous study [22], we hypothesized that organizations would give organization size, cost and time as the main reasons for not adopting CMMI. However, we were prepared to find differences in the frequency order of the reasons provided, due to socio-cultural differences [8], [13]. We also hypothesized that small and medium sized organizations would differ in their response patterns compared to large organizations, due to the particular challenges smaller organizations face [19].

Three consulting firms were appointed to promote the CMMI program. The program offered a 50% subsidy of the CMMI Level 2 implementation cost (Class A). Companies were selected based on program criteria from government cross-agency databases, industry bodies, and consultants' personal contacts. During a two month recruitment period, consultants briefed the companies about the program and provided a formal overview of the CMMI framework. Consultants were required to note interest and reasons given for non-interest in the program. Data was collected from 85 companies and recorded in a spreadsheet, along with contact notes.

Based on the contact notes, each organization was categorised as “Interested”, “In Progress” or “Not Interested” at the end of the recruitment period. Of the 85 companies contacted, 60 were categorised as “Not interested”. Using only these “Not Interested” organizations, the text of the consultants’ contact notes was analysed to identify distinct reasons for each organization. Each organization could give multiple reasons. Three researchers (the authors) independently classified these reasons based on the reason classifications from the Australian study [22]. Initially, 23 of the 67 reasons coded (34%) were inconsistent between the researchers due to data and classification ambiguity. In a joint meeting, 16 of the differences were resolved by clarifying the intent of the reasons provided and the meaning of the categories [22]. The remaining 7 disagreements were resolved by introducing two new classification categories, namely, “Want certification for whole company” and “Priorities”. “Want certification for whole company” refers to an organization that did not want to use the program because CMMI assesses only the software development component and not the entire organization. When organizations did not want to use the program because they had other higher priorities, we classified their reason as “Priorities”. This was considered to be a different reason to the “No time” reason category from the Australian study. These reasons were then categorized under “could not” and “should not” reason groups, according to the previous study (see further following). Table 1 presents a summary of the categorised reasons and reason groups.

Table 1. Grouping of reasons from all “Not interested” organizations into “could not” and “should not” categories

Group	Group frequency (of 60 orgs)	Reason	Reason frequency
N/A	6	Want certification for whole company	1
		No software development	2
		Want higher rating	3
Insubstantive	9	No interest	7
		May consider SPI later	2
Could not	28	Small organization	8
		Too costly	15
		No time	4
		Not applicable to our projects	3
Should not	21	Priorities	6
		Using other SPI	3
		No clear benefit	9
		Potential benefits not wanted	2
		No customer demands	2
		Risk of poor certification damaging business	0

Since our research questions concern organizations that decided not to adopt CMMI, we excluded companies that were simply ‘not interested’ in pursuing CMMI or provided insubstantial reasons for not adopting (a total of 15 companies). The remaining 45 companies provided the data set for subsequent analysis.

The size of each organization was extracted from a government agency database that determined the organization's size category (SMALL, MEDIUM or LARGE). The number of employees recorded in the database was rounded rather than the exact number of employees. Each organization under analysis was categorized as SMALL, MEDIUM or LARGE. Organization size was defined as in the previous study [22] to ensure comparable analysis. This definition is based on the Australian Bureau of Statistics definition of small and medium enterprises [24]. Organizations with less than 20 employees were considered small organizations, with 20 to 199 employees classified as medium, and 200 or more employees classified as large. As the study was based on the Malaysian software sector, we also analysed the data by applying the Malaysian definition of organization size, as developed by the Malaysian National SME Development Council (small – from 5-19 employees; medium – from 20 to 50 employees; or annual sales turnover up to MYR$5 million) [16]. We addressed the size-related validity threats by separate analyses, which are discussed in later sections. Table 2 provides a summary of the size of the organizations in the study by size category, based on the Australian standard.

Table 2. Summary statistics of organization size under analysis

Size category	N	Min	Median	Mean	Max	Std.Dev
Small	11	10	15	13.18	15	2.64
Medium	30	20	35	49.67	150	32.97
Large	4	200	200	200	200	0

4 Results and Data Analysis

This section presents the results of the data analysis with a focus on the reasons given by organizations and the relationship between these reasons and organization size.

4.1 Non-adoption Reasons

The frequency-ordered list of substantive reasons for not participating in the CMMI Level 2 program given by the 45 non-adopting organizations is shown in Table 3. For comparison purposes, the table also shows the frequencies of reasons given in the earlier Australian study [22]. For organizations that provided a reason of "Using other SPI", two claimed that ISO9001 was sufficient to maintain any process management while another was using Six Sigma as its SPI method. The number of organizations citing reasons that grouped under "could not", "should not" or both, is shown in Table 4, together with comparison figures from the earlier study.

We had hypothesized that size, cost, and time would be the most common reasons for not adopting CMMI, after the earlier study [22]. We found that the most common reasons were: being too costly; no clear benefits; being a small organization and CMMI not being an organizational priority. These reasons are generally consistent with our hypotheses and the results of the previous study, although the frequency ordering is different.

Table 3. Reason classification

Reason	This Study Frequency (of 45)	Prior Study Frequency (of 40)	This Study %	Prior Study %
Too costly	15	14	33	35
No clear benefit	9	4	20	10
Small organization	8	17	18	43
Priorities	6	-	13	-
No time	4	10	9	25
Not applicable to our projects	3	2	7	5
Using other SPI	3	8	7	20
No customer demands	2	2	4	5
Potential benefits not wanted	2	3	4	8
Already know gaps	-	2	-	5
Risk of poor certification damaging business	-	1	-	3

Table 4. Number of organizations under analysis giving “could not” reasons, “should not” reasons, or both types of reason

Reason Group	This Study Frequency (of 45)	Prior Study Frequency (of 40)	This Study %	Prior Study %
Only could not reason(s)	24	23	53	58
Only should not reason(s)	17	8	38	20
Both could and should not reason(s)	4	9	9	23

Overall, organizations tended not to adopt CMMI because of its perceived infeasibility (that is, “could not” reasons).

One common reason found in our study that did not fit with reasons found in the earlier study was conflicting priorities with the organization’s current operations. Organizations citing this reason felt that business as usual operations were a higher priority than adopting CMMI. This reason may share common antecedents with the “No time” reason that ranked in the top three reasons in the Australian study.

4.2 Relationships between Size and Reason

To analyse the relationships between organizational size and reasons, we applied Fisher’s exact test (using FEXACT [15] in the R package, version 2.1.1 [18]). A chart of the percentage of each organization size category giving each reason is shown in Fig. 1. We found no statistically significant relationship between size and reason. We decided to combine the current study data with data from the previous study. We excluded all organizations with a size that was UNKNOWN, as discussed in the previous study [22]. We tested the significance between size and each of the reasons using the combined data. There were no significant associations except for the reason of being a “small organization” (Fisher’s exact test, $p = 0.0003$) at the 0.05 level. This agrees with the findings of the previous study [23] (p-value of 0.022). A total of 22% of the SMALL organizations gave this reason, compared to 19% of MEDIUM+ LARGE organizations. A summary of significance tests using Fisher’s exact test using data from this study and combined with previous data is shown in Table 5.

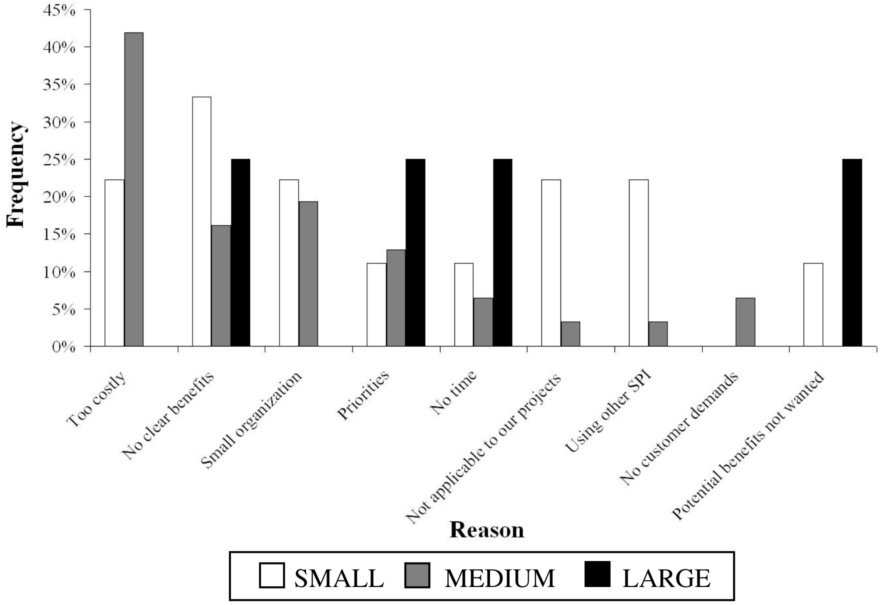


Fig. 1. Percent of each size group for organization under analysis giving each reason

Table 5. Summary of FEXACT test between organization size and reasons

Reasons	Current Study (p-value)	Previous Study (p-value)	Previous study + Current study (p-value)
Too costly	0.2723	0.6593	0.3794
No clear benefit	0.3960	0.2063	0.8870
Small organization	0.0717	0.0217	0.0003
Priorities	0.7834	-	-
No time	0.3378	1.0000	0.4741
Not applicable to our projects	0.2042	-	-
Using other SPI	0.2042	0.2318	0.3216
No customer demands	1.000	-	-

We then analysed the relationship between organization size and grouped reasons (could not, should not, or both reasons). The percentage of organizations of each size category for grouped reasons “should not”, “could not” or “both” is shown in Fig 2.

In the present data, there were no significant results shown for each of the groupings. However, when data from the current study was combined with data from the previous study [22], we found a significant relationship between size of organization and “should not” only, “could not” only, or “both” reasons (p = 0.0012, Fisher’s exact test) at the 0.05 level. However, we found no association between organization size and “should not” and “could not” only reasons in both groups of data. This may be due to the smaller representation of SMALL and LARGE organizations in the sample.

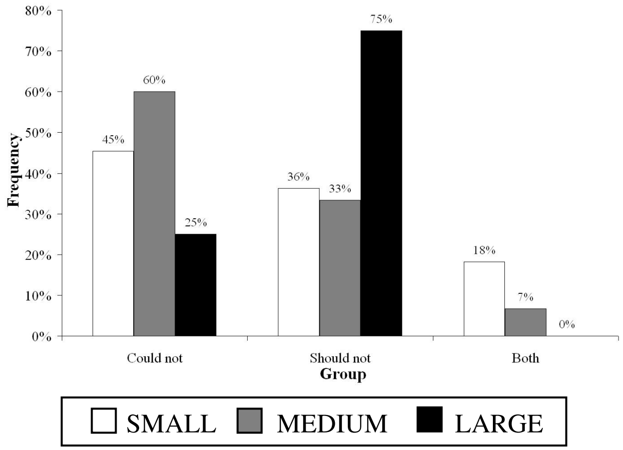


Fig. 2. Percentage of organizations of each size group giving reason(s) in either or both groups

We then re-organized the data using the Malaysian definition of organization size category within the present study [16]. Using the Fisher's exact significant test, there were no differences in results between organization size categories and reason(s) given. However, significance exists between organization size and grouping reasons (should not, could not, and both, p = 0.0389). For "could not" reasons, there was a significant overall relationship to size (p = 0.0012, Fisher's exact test). A summary of the Fisher's test between organization size and grouped reasons is shown in Table 6.

Table 6. Summary of significant test between organization size and grouped reasons

Groups	Current Study	Previous Study + Current Study	Current Study (revised org. category)
Could not	0.2876	0.0982	0.0012
Should not	0.2248	0.2901	0.0389
Both	0.4927	0.0432	0.0389

45% of SMALL organizations claimed they could not adopt the program offered (i.e. adopt CMMI) as opposed to 25% of LARGE organizations. By contrast, 75% of LARGE organizations cited they "should not" adopt CMMI compared to 36% of SMALL organizations.

We hypothesized that responses by SMEs may be different from those of large organizations because of the challenges faced by smaller organizations [19]. Smaller organizations are often challenged by cost overheads, limited funds, and stretched resources, that can significantly constrain their ability to innovate. SMALL and MEDIUM organizations tended to give "could not" reasons, while LARGE organizations tended to give "should not" reasons. This result is consistent with the prior Australian study [22] and a US study that reported that small organizations do not have the resources or funds to implement CMM and perceive that CMM practices are not directly applicable to them [4].

5 Discussion

To increase the adoption rate of SPI in organizations, it is necessary to understand why organizations do not overcome the first hurdle of deciding to initiate a software process improvement program. This is as important as understanding the factors that influence SPI implementation success and sustainment of SPI practices. This study has aimed to replicate a prior study [22] at the organizational (rather than practitioner) level, in a different country, to validate the earlier findings and add further insight to the problem. The study found that the most frequently cited reasons for not adopting CMMI were: the SPI program was too costly; the companies were unsure of the benefits; the organization was too small; and/or the organization had other priorities. The top two reasons, cost and benefit, suggest that regardless of size, organizations may face a challenge in justifying SPI from a business perspective. This may also reflect a perceived limited range of program entry options and/or a lack of understanding of the process improvement value chain.

Considering size, prior research (such as [4]) and experience have found that organization size can be a significant barrier to adoption of comprehensive SPI methods such as CMMI. Both the prior and the current study support this finding. Small organizations tended not to adopt CMMI for reasons of infeasibility (what we called "could not" reasons) in contrast to larger organizations, which tended not to adopt CMMI because it was perceived to be unbeneficial (that is, for "should not" reasons).

The results of this study are generally consistent with the earlier study. Table 3 and Table 4 provide comparative reasons and reason groupings for both studies. There are differences in the frequency order of the main reasons provided, but we hypothesized that this might occur due to socio-cultural differences. Being a replication, the study does not permit specific conclusions to be drawn about the antecedents of these differences, other than to acknowledge that national difference may have influenced these variations. Socio-cultural differences will require specific investigation in future studies.

5.1 Limitations

The study involves a group of companies within a government program's selection criteria, which does not represent a random sample of the population. The subsidy program targeted SME ICT companies, selected from a Malaysian government database. As the program targeted organizations between 10 and 50 staff (due mainly to the Malaysian definition of an SME), there is little representation of either very small or large organizations in the study. Most of the sample population companies were medium-sized (20-199 staff). However, this bias is similarly found in the earlier study.

Organization size may present a threat to the validity of the study. Size categories were determined based on using rounded figures from a government agency database. At the margins, a small company with 19 actual staff may have been recorded as 20, which fits in the medium sized organization category. This may have distorted the coding. Also, the size represents the size of the entire organization rather than size of the software development unit. However, this issue is more likely to have had the effect of over-estimating size, and therefore weakening the findings about the relationship between small size and reason, rather than strengthening them. We also conducted a sensitivity analysis to address this threat. We collapsed SMALL + MEDIUM organizations and tested for any changes in significance. We then collapsed MEDIUM + LARGE organizations to test if there was any impact on the significance test between organization size and reasons provided. We recorded no impact of collapsing organization categories on whether the relationships were statistically significant.

Finally, the data was collected as part of a market research and sales effort rather than by using a survey instrument designed for scientific study. Using a controlled survey may have reduced ambiguity in the reasons given and provided a better understanding of why the companies decided not to adopt CMMI.

5.2 Implications

The study has implications for research and practice. For research, understanding organization level barriers to SPI adoption is as critical as identifying implementation and sustainment success factors. The current paper has replicated and supported the only prior study on this issue. Further research is necessary, however, to understand the underlying drivers of the reasons given, especially with respect to how they might vary with organizational size and national culture.

With respect to size, the majority of software development companies are SMEs, so there is a need to further explore the unique issues that face these companies' use of SPI. SMEs have less capacity to perform SPI tailoring in-house, but currently most SPI methods require more tailoring and adaptation for SMEs than for large organizations. To improve SPI adoption, more appropriate and accessible methods are needed to assist SMEs in the adoption of improved software development processes. Furthermore, as yet, there is no compelling, widely accepted cost-benefit justification for adopting SPI or defined causal path to track software process improvement through to business value creation. To achieve broader impact in practice, SPI researchers and practitioners need to fundamentally recognise SPI as a business investment rather than just as an incremental product or process quality improvement technique. In addition, it is important for researchers and practitioners to consider how SPI deployment strategies can reduce risk and shorten the time required to realise the promised benefits of SPI.

6 Conclusions

This study has replicated the research questions and methodology of a prior study that focused on understanding the reasons why organisations fail to overcome the first hurdle along the path to software process improvement by deciding not to adopt CMMI. The main variable between the two studies was a different country. Replication studies are fundamental to the scientific method in general and to empirical software engineering in particular, to build a knowledge base on SPI adoption and establish the boundaries of the underlying theory. The study supports and is largely consistent with the results of the earlier study. Our study contributes further insight into the problem of SPI adoption. It suggests that future progress requires developments both within adopting organizations, in terms of understanding the business value that can be generated from SPI, and in SPI methods, to provide improved entry options and a closer fit to a wider range of organizational needs, especially ones that are size dependent.

Acknowledgements

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

References

1. Abrahamsson, P., Iivari, N.: Commitment in Software Process Improvement: In Search of the Process. In: Proceedings of 35th Hawaii International Conference on System Science (HICSS 2002), vol. 8. IEEE Computer Society, Washington (2002)
2. Alstrom, D., Bruton, G.: Turnaround in Asia: What do we know? Asia Pac. J. Manage. 21, 5–24 (2004)
3. Baddoo, N., Hall, T.: De-motivators for Software Process Improvement: an Analysis of Practitioners' Views. J. Syst. & Softw. 66(1), 23–33 (2003)
4. Brodman, J.G., Johnson, D.L.: What Small Business and Small Organizations Say about the CMM: Experience Report. In: Proceedings of the 16th International Conference on Software Engineering (ICSE 1994), pp. 331–340. IEEE Computer Society, Los Alamitos (1994)
5. Chrissis, M.B., Konrad, M., Shrum, S.: CMMI: Guidelines for Process Integration and Product Improvement. Addison-Wesley, Reading (2003)
6. Coleman, G., O'Connor, R.: Investigating Software Process in Practice: A Grounded Theory Perspective. J. Syst. & Softw. 81(5), 772–784 (2008)
7. Conradi, R., Fuggetta, A.: Improving software process improvement. IEEE Softw. 19(4), 92–99 (2002)
8. Fink, D., Laupase, R.: Perceptions of Web Site Design Characteristics: A Malaysian/ Australian Comparison. Internet Res. 10(1), 44–55 (2000)
9. Gibson, R.: Software Process Improvement: Innovation and Diffusion. In: Larsen, T.J., McGuire, E. (eds.) Information Systems Innovation and Diffusion: Issues and Directions, pp. 71–87. Idea Group Publishing (1998)
10. Guerrero, F., Eterovic, Y.: Adopting the SW-CMM in Small IT Organizations. IEEE Softw. 21(4), 29–35 (2004)
11. Hansen, B., Rose, J., Tjørnehøj, G.: Prescription, Description, Reflection: The Shape of the Software Process Improvement Field. Int. J. Inform. Manage. 24(6), 457–472 (2004)
12. Hersleb, J., Carleton, A., Rozum, J., Siegel, J., Zubrow, D.: Benefits of CMM-Based Software Process Improvement: Initial Results. Software Engineering Institute, CMU/SEI-94-TR-013 (August 1994)
13. Hofstede, G.J.: Cultures and Organizations: Software of the Mind. McGraw-Hill Professional, New York (2004)
14. Humphrey, W.S.: Software Process Improvement – A Personal View: How it Started and Where it is Going. Softw. Process Improv. & Pract. 12(3), 223–227 (2007)

15. Mehta, C.R., Patel, N.R.: ALGORITHM 643: FEXACT: a FOTRAN Subroutine for Fisher's Exact Test on Unordered rxc Contingency Tables. ACM Trans. on Math. Softw. 12(2), 154–161 (1986)
16. National SME Development Council: Definitions for Small and Medium Enterprises in Malaysia (2005)
17. Niazi, M., Ali Babar, M., Katugampola, N.M.: Demotivators of Software Process Improvement: An Empirical Investigation. Softw. Process Improv. & Pract. 13(3), 249–264 (2008)
18. R Development Core Team, R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna (2005)
19. Richardson, I., von Wangenheim, C.G.: Why Are Small Software Organizations Different. IEEE Softw. 24(1), 18–22 (2007)
20. Shull, F.J., Carver, J.C., Vegas, S., Juristo, N.: The role of replications in Empirical Software Engineering. Empir. Softw. Eng. 13, 211–218 (2008)
21. Smith, P.B.: Organizational Behaviour and National Culture. Br. J Manage. 3(1), 39–50 (1992)
22. Staples, M., Niazi, M., Jeffery, R., Abrahams, A., Byatt, P., Murphy, R.: An exploratory study of why organizations do not adopt CMMI. J. Syst. & Softw. 80(6), 883–895 (2007)
23. Staples, M., Niazi, M.: Systematic Review of Organizational Motivations for Adopting CMM-based SPI. Inform. & Softw. Technol. 50(7/8), 605–620 (2008)
24. Trewin, D.: Small Business in Australia: 2001. Australian Bureau of Statistics, 1321.0 (2002)
25. Wilkie, F.G., Rombach, D., Penn, M.L., Jeffery, R., Herndon, M.A., Konrad, M.: Motivating Small to Medium Sized Enterprises to Adopt Software Process. In: Proceedings of the International Process Research Consortium (October 2004)

Value-Based Multiple Software Projects Scheduling with Genetic Algorithm

Junchao Xiao[1], Qing Wang[1], Mingshu Li[1,2], Qiusong Yang[1], Lizi Xie[1], and Dapeng Liu[1]

[1] Laboratory for Internet Software Technologies, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China
[2] Key Laboratory for Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China
{xiaojunchao,wq,mingshu,qiusong_yang,xielizi, liudapeng}@itechs.iscas.ac.cn

Abstract. Scheduling human resources to multiple projects under various resource requirements, constraints and value objectives is a key problem that many software organizations struggle with. This paper gives a value-based human resource scheduling method among multiple software projects by using a genetic algorithm. The method synthesizes the constraints such as those of schedule and cost as well as the value objectives among different projects, and also the construction of comprehensive value function for evaluating the results of human resource scheduling. Under the guidance of value function, capable human resources can be scheduled for project activities by using the genetic algorithm and make the near-maximum value for organizations. Case study and the simulation results show that the method can perform the scheduling and reflect the value objectives of different projects effectively, and the results provide a concrete decision support for project managers.

Keywords: Value; human resource; multi-project scheduling; genetic algorithm.

1 Introduction

Software organizations often have multiple projects which are developed concurrently. Since the projects may have different stakeholders who bear different requirements and preferences, each project holds different constraints and different value objectives. One of the goals of an organization is to achieve the maximum value from the projects and response to the changing market timely [1].

Human resource scheduling among multiple projects should satisfy the requirement of the software development and make organizations obtain the maximum value [2, 3]. To achieve this goal, it is essential to resolve two problems: (1) Define the value obtained by scheduling according to constraints, value objectives and possible scheduling results in projects; (2) Provide a multi-project scheduling method which can obtain the (near-) maximum value for the organization.

Resource scheduling is proved to be NP-hard [4], and it is more complicated when concerning different constraints, value objectives, resource capabilities and

Q. Wang et al. (Eds.): ICSP 2009, LNCS 5543, pp. 50–62, 2009.

availabilities among multiple projects [5, 6]. There are some methods having resolved the multi-objective release planning under limited resources, which can bring the organization maximum value [2, 6-8]. However, those methods are either deficient in descriptions of resources or lack of scheduling them among multiple projects and project activities. Some researches use a simulation method to provide the strategies of resource management and scheduling [9, 10], however, they do not focus on the developers' characteristics. In the project portfolio management, resource scheduling usually focuses on the manufacturing industries[4, 11]. Aiming at the resource scheduling in software projects, several researchers give the human resource scheduling methods [5, 12-14].These methods lack a comprehensive consideration of the various project constraints and optimized scheduling among multiple projects.

This paper proposes a value-based multiple software projects scheduling method by using a genetic algorithm. In this method, a value function in multi-project environments is defined. It takes multiple constraints, value objectives of different projects and possible scheduling results into consideration, and is used to guide the scheduling and help software organizations achieve the near-maximum value. To tackle the problem of high complexity, genetic algorithm (GA) is adopted. GA is an evolutionary algorithm that can get nearly optimal solutions with high efficiency [15].

Section 2 presents a motivating example. Section 3 gives the value function for multi-project scheduling. Section 4 realizes the multi-project scheduling by using the genetic algorithm. Section 5 gives the case study and analysis of the simulation results. Section 6 provides the conclusions and future work.

2 Motivating Example

Assume there are 3 projects in an organization. Project P1 involves two modules to be realized, P2 is an upgrading project for a certain product, and P3 includes three modules. The processes used by these projects are described in Fig. 1.

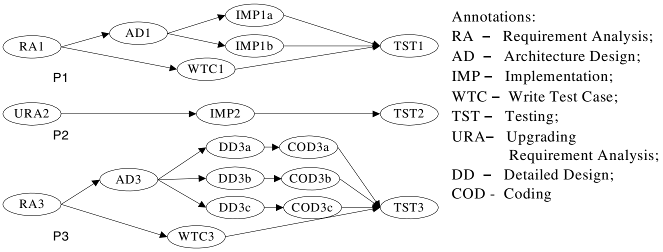


Fig. 1. Processes Used by Projects P1, P2 and P3

Each project has its own schedule and cost constraints as well as its preferences. If the project can be finished save time or below budget, some benefits can be obtained for the project, such as the increase in customer satisfaction, more money earned by the organization. If the project is postponed or overspent, there will be some penalty for it, such as compensation asked for by the customer and the decrease in customer satisfaction. Furthermore, the importance of projects may be different. The preferences and the importances which are determined by the negotiations among stakeholders reflect the value objectives of the projects. The example of these aspects is described in Table 1. Managers must take all the aspects into consideration and balance the resource requirements and value objectives among multiple projects during scheduling, thus making the organization obtain the maximum value.

Table 1. Constraints, preferences, benefits and penalties in different conditions

	P1	P2	P3
Schedule constraint	[2008-03-01, 2008-07-20]	[2008-04-10, 2008-6-30]	[2008-03-01, 2008-10-31]
Cost constraint	$2*10^5$	$5*10^4$	$3.5*10^5$
Preference	Cost preference	Schedule preference	Cost preference
Schedule ahead benefit ($/day)	400	500	500
Schedule postpone penalty ($/day)	400	500	500
Cost saved benefit ($)	Equal to saved	Equal to saved	Equal to saved
Cost exceeded penalty ($)	Equal to exceeded	Equal to exceeded	Equal to exceeded
Project failure penalty($)	10^6	10^6	10^6
Project importance preference	4	1	2

3 Value Function for Multi-project Scheduling

This section describes the projects, human resources and the value function.

3.1 Description of Projects

Suppose there are *k* projects, P_1, P_2, ..., P_k. A project is defined as follows:

P_i = ($ActSet_i$, $ConSet_i$, $PWSet_i$).

In this definition, P_i is the i^{th} project of the organization, $ActSet_i$ is the activity set of P_i, $ConSet_i$ is the constraint set of P_i, $PWSet_i$ is the preferential weight set of P_i.

(1) Activity Set

Through WBS (Work Breakdown Structure), a project process can be divided into a group of related activities. Suppose there are n_i activities in P_i, that is,

$ActSet_i = \{A_{i,1}, A_{i,2}, ..., A_{i,n_i}\}$

Each activity is described by the attributes including identification (ID), type (TYPE), precedent activity set (PREA), size (SIZE), and required skills for human resources (SKLR). Details of these attributes are described in [16].

(2) Constraint Set

The constraints of a project may come from the negotiations among stakeholders. It includes constraints such as the schedule constraint (SCH) and the cost constraint (CST). It could be expanded according to other demands, such as quality constraint. A constraint set is defined as follows:

$ConSet_i$ = {SCH_i, CST_i}

SCH_i includes the start date (SD) and due date (DD) constraint of P_i, it is described as SCH_i=[SD_i, DD_i]. CST_i is the cost constraint for P_i.

(3) Preference Weight Set

It includes the preference weight of the project (PPW), the preference weight of the schedule (SPW) and the cost (CPW) of the project. Their may also come from the negotiations among stakeholders. It could also be expanded according to the other demands. A preference weight set is defined as follows:

$PWSet_i = \{PPW_i, SPW_i, CPW_i\}$

When PPW_i is high, the value influence of project P_i on the organization is remarkable, so P_i should obtain resources on priority. When SPW_i or CPW_i is high, schedule or cost optimization should be taken into consideration for P_i scheduling.

3.2 Description of the Human Resources

The capabilities and availabilities of human resources affect their scheduling. A human resource is described by identification (ID), the capability attributes such as executable activity type set (EATS), skill set (SKLS), experience data (EXPD), salary per man-hour (SALR), and the availability attributes such as schedulable time and workload (STMW). Details of these attributes are described in [16].

If a human resource's EATS has an element that is the same as the activity type and SKLS has all the skills that activity requires, then the human resource has the capabilities to execute the activity and can be scheduled to the activity.

3.3 Multi-project Value Function

The value of an organization is determined by the value of each project in the organization. The project value is affected by its constraints and value objectives. The value objectives may include various kinds of preferences. This section explains the computation of the value by using schedule and cost constraints and preferences.

(1) Project Schedule Value (*SValue*)

The execution of a project usually begins from its start date (SD). Let AFD_i be the actual finishing date of project P_i. In terms of schedule, let CSB_i be the coefficient of schedule benefit when P_i is ahead of the schedule, that is, every one day ahead of the schedule will result in more benefits. The benefit ahead of schedule is:

$$SBenefit_i = CSB_i * \frac{|DD_i - AFD_i| + (DD_i - AFD_i)}{2}$$

If $DD_i \geq AFD_i$, the AFD is earlier than the due date constraint, then $SBenefit_i = CSB_i * (DD_i - AFD_i)$; if $DD_i < AFD_i$, the project is delayed, then $SBenefit_i = 0$.

If P_i is delayed, we define a coefficient CSP_i for schedule delay penalty, that is, the penalty caused by every one day delay of schedule. Then, the penalty for the delay is:

$$SPenalty_i = CSP_i * \frac{|AFD_i - DD_i| + (AFD_i - DD_i)}{2}$$

If $DD_i \geq AFD_i$, the AFD is earlier than the due date constraint, then $SPenalty_i = 0$, if $DD_i < AFD_i$, the project is delayed, then $SPenalty_i = CSP_i * (AFD_i - DD_i)$;

Therefore, the schedule value brought to the P_i is described as follows:

$$SValue_i = SBenefit_i\text{- } SPenalty_i$$

(2) Project Cost Value (*CValue*)

Suppose the scheduled human resources for a certain activity j in P_i are $HR_{i,j,1}$, $HR_{i,j,2}$, ..., $HR_{i,j,m_{ij}}$. If the workload that $HR_{i\,j,r}$ ($1 \leq r \leq m_{ij}$) affords in activity j is $E_{i,j,r}$, then the cost of $HR_{i,j,r}$ in activity j of P_i is $E_{i,j,r}$* $HR_{i,j,r}.SALR$. Thus, the cost of activity j in P_i is the sum of cost of all the human resources involved in the activity:

$$AActCST_{i,j} = \sum_{r=1}^{m_{ij}} \left(E_{i,j,r} * HR_{i,j,r}.SALR\right)$$

The cost of P_i is the sum of the cost of all the activities involved in the project:

$$APrjCST_i = \sum_{j=1}^{n_i} \sum_{r=1}^{m_{ij}} \left(E_{i,j,r} * HR_{i,j,r}.SALR\right)$$

From the aspect of cost, if the cost of P_i is saved, we define a coefficient CCB$_i$ for cost save benefit. Then, the benefit obtained from the save of the cost is:

$$CBenefit_i = CCB_i * \frac{|CST_i - APrjCST_i| + (CST_i - APrjCST_i)}{2}$$

If $CST_i \geq APrjCST_i$, the cost is saved, then $CBenefit_i$ = $CCB_i * (CST_i - APrjCST_i)$, if $CST_i < APrjCST_i$, the cost is overspent, then $CBenefit_i$=0.

If P_i overspends, we define a coefficient CCP_i for cost overspent penalty. Then, the penalty obtained from the overspent is described as follows:

$$CPenalty_i = CCP_i * \frac{|APrjCST_i - CST_i| + (APrjCST_i - CST_i)}{2}$$

If $CST_i \geq APrjCST_i$, that is, the cost is saved, then $CPenalty_i = 0$, if $CST_i <$ $APrjCST_i$, that is, the cost is overspent, then $CPenalty_i = CCP_i * (APrjCST_i - CST_i)$.

Therefore, the cost value brought to P_i is described as follows:

$$CValue_i = CBenefit_i \text{-} CPenalty_i$$

Since every project may have specific schedule and cost preference, schedule and cost play different importance on value. Therefore, the value of project P_i is:

$$Value_i = SPW_i * \left(SBenefit_i - SPenalty_i\right) + CPW_i * \left(CBenefit_i - CPenalty_i\right)$$

Though the above value only takes the schedule and cost into consideration, it is an open formula and could be extended according to more value objectives.

If P_i fails because of the scarcity of certain resources, we set the penalty for P_i be PP_i. This penalty may come from the customers. Then the value of P_i is:

$$Value_i = -PP_i$$

In an organization, the importances and preferences of different projects may also be different, for example, some urgent projects are more important than others. Thus the total value obtained by the organization is:

$$Value_{multi} = \sum_{i=1}^{k} \left(PPW_i * Value_i\right)$$

4 Multi-project Scheduling with Genetic Algorithm (GA)

Scheduling problems are described by means of chromosome and fitness is defined by value function. Through GA, the scheduling with near-maximum value is realized. The steps of scheduling are as follows:

(1) Determine the potential capable human resources for the activities on the basis of the description of activities and human resources. Establish the structure of the chromosome according to their relationships and generate initial population.
(2) For each generation, decode each chromosome to a multi-project scheduling scheme according to the chromosome structure. Compute the value for each chromosome on the basis of the value function.
(3) Select the chromosomes with higher fitness for certain times thus preparing the population for the next generation.
(4) Crossover and mutate for the new generation.
(5) Evaluate the new generation. If the result satisfies the stopping criterion or a particular number of generations have been reached, then go to (6). Otherwise, go to (2)
(6) In the last generation, select the chromosome with the highest fitness and generate the scheduling result accordingly.

Since the chromosomes with higher value will be inherited, after a number of generations, the value of the scheduling will be increased to the highest or sub-highest.

4.1 Structure of the Chromosome

The basis for resolving the multi-project scheduling with GA is the description of the scheduling problem with chromosome.

Encode: Suppose there are k projects $P_1, P_2, ..., P_k$. Each P_i includes n_i activities. Let the total number of activities be $N = \sum_{i=1}^{k} n_i$. We construct an activity queue $A_1, A_2, ..., A_N$, it ensures that the precedent activities of a activity be in front of that activity in the queue. On the basis of *TYPE* and *SKLR* described in the activity as well as the *EATS* and *SKLS* described in the human resource, the capable human resource for executing each activity can be determined. Suppose the human resources who have the capabilities to execute A_i are $HR_{i,1}, HR_{i,2}, ..., HR_{i,ti}$. According to the activity queue, a human resource queue can be settled:

$HR_{1,1}, HR_{1,2}, ..., HR_{1,t1}, HR_{2,1}, HR_{2,2}, ..., HR_{2,t2}, ..., HR_{N,1}, HR_{N,2}, ..., HR_{N,tN}$.

Next, a queue of genes can be generated, where a gene represents the corresponding human resource in the human resource queue. This is the first part of the chromosome, which is donated in the left part of Fig. 2. The length of this part is $T = \sum_{i=1}^{N} t_i$ .If a gene is "1", it means the corresponding human resource is scheduled to the corresponding activity. If a gene is "0", it means the corresponding human resource is not scheduled to the corresponding activity.

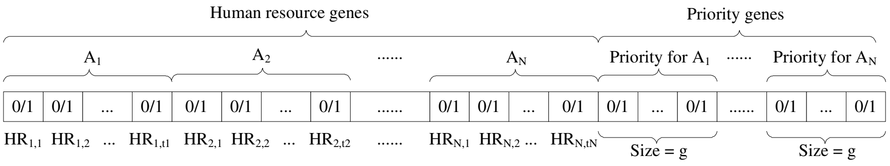


Fig. 2. Structure of the Chromosome

If a human resource has the capabilities and is scheduled to execute several activities concurrently, there should be some priorities to determine which activity should be scheduled first. Therefore, we set the priority gene for each activity. Let the priority gene size for each activity be *g* (g is an input of GA), then the genes whose size are *N*g* construct the second part of the chromosome, which are donated in the right part of Fig. 2. The priority of each activity is the value of the corresponding *g* binary code.

Therefore, the length of the chromosome is:

$$CL = \sum_{i=1}^{N} t_i + N * g$$

Decode: Each chromosome encoded by the above method can be decoded as a scheduling scheme. The decoding process is described as follows:

(1) Select all the activities that do not have precedent activities or whose precedent activities have been assigned. If no such activity exists, then decoding is completed.

(2) Sort all selected activities as a queue according to their priority from high to low.

(3) For each activity "*ACT*" in this queue, do the following steps:

   a) Set the capable human resources whose corresponding gene value is "1" as the scheduled human resources for *ACT*.

   b) Set the start date of *ACT* as the current date.

   c) Allocate all the schedulable workload of all the scheduled human resources in the current date to *ACT* and update the availability state of the resources.

   d) If the scheduled workload to *ACT* can complete *ACT*, then set current date be the due date of *ACT* and update the start date of the activities whose precedent activity is *ACT* as the current date. Go to (3).

   e) Add one day to the current date, go to (c).

(4) Go to (1).

After the above process, all the activities have the scheduled human resources that make up the scheduling scheme. Since the activities with higher priority will be scheduled earlier, they have higher priority for the resources. If some activities do not have enough scheduled resources, then the scheduling for this activity fails which further causes the project failure and brings the negative value for the chromosome.

4.2 Fitness Function of the Chromosome

Since the goal of scheduling is to obtain the maximum value for the organization, the fitness function is set from the value function. For the scheduling scheme generated from chromosome, let $Value_{multi}$ be its value. Since this value may be negative, we use the following formula as the fitness function of the chromosome:

$$Fitness = \begin{cases} (Value_{multi})^2 & \text{if} \quad Value_{multi} > 1 \\ 1 & \text{if} \quad Value_{multi} \in [-1,1] \\ \left({}^{1}\!\!/_{-Value_{multi}} \right)^2 & \text{if} \quad Value_{multi} < -1 \end{cases}$$

The square is used in the formula to make the difference of the fitness more distinct for different value, which can accelerate the evolution speed.

4.3 Running the Genetic Algorithm

Before running GA, the following parameters should be set:

(1) Population scale (*PS*): the number of the chromosomes.

(2) Mutation rate (*MR*): the possibility of mutation to chromosome.

(3) Maximum generation number: if the generation number arrives at the maximum generation number, then the running of the GA will be terminated.

(4) Termination condition: when the running of the GA should be terminated. It can be the obtaining of a value or running GA for maximum generation number times.

After parameter setting, the scheduling will be performed according to GA steps.

Generate the initial population with *PS* chromosomes. For generating a new generation, the Roulette Wheele is used for the chromosome selection. Let the fitness of chromosome *K* be $Fitness_k$, the probability of *K*'s selection is $\frac{Fitness_k}{\sum_{k=1}^{PS} Fitness_k}$.

Crossover combines the features of two parent chromosomes and generates two offspring. The two parents are paired and the crossover point is generated randomly.

During mutation, the mutation chromosomes and mutation points are also selected randomly according to *MR*. At the mutation point, the gene value will be changed.

When the termination condition is achieved or the generation number arrives at the maximum generation number, the running of GA is terminated.

5 Case Study

For analyzing the effectiveness of the method proposed in this paper, this section compares the simulation results of the multi-project scheduling by setting different coefficients and weight for the example described in section 2.

5.1 Description of the Projects and Human Resources

For the projects described in Section 2, the initial parameters are listed in Table 2.

Table 2. Coefficients and Weight of the projects

	CSB	CSP	CCB	CCP	P	SPW	CPW	PPW
P1	400	400	1	1	10^6	1	2	4
P2	500	500	1	1	10^6	4	1	1
P3	500	500	1	1	10^6	1	2	2

For these projects, we assume there is a group of candidate human resources that can be scheduled are described in Table 3. These human resources have stable productivity in each of the executable activity types. They are all available from 2008-03-01 to 2009-12-31, and the schedulable workload is 8 hours per day. In the *SALR* column, the unit is "*$/Man-Hour*". The skill requisites of the activities and the skill set possessed by human resources are omitted because of space considerations.

Table 3. Human resource description

	EATS	EXPD(KLOC/Man-Hour)	SALR
HR1	RA	$P_{RA} = 0.05$	65
HR2	RA	$P_{RA} = 0.05$	65
HR3	RA	$P_{RA} = 0.05$	70
HR4	RA	$P_{RA} = 0.04$	45
HR5	RA	$P_{RA} = 0.05$	50
HR6	AD, DD	$P_{AD} = 0.06$, $P_{DD} = 0.05$	65
HR7	AD, DD	$P_{AD} = 0.055$, $P_{DD} = 0.05$	60
HR8	AD, DD	$P_{AD} = 0.05$, $P_{DD} = 0.055$	50
HR9	AD, DD	$P_{AD} = 0.04$, $P_{DD} = 0.06$	65
HR10	IMP, DD, COD	$P_{IMP} =0.025$, $P_{DD}=0.05$, $P_{COD}=0.03$	55
HR11	IMP, DD, COD	$P_{IMP} = 0.025$, $P_{DD}= 0.05$, $P_{COD}= 0.03$	50
HR12	IMP, DD, COD	$P_{IMP} = 0.02$, $P_{DD}= 0.05$, $P_{COD}= 0.03$	45
HR13	IMP, DD, COD	$P_{IMP} =0.02$, $P_{DD}=0.03$, $P_{COD}=0.03$	40
HR14	IMP, DD, COD	$P_{IMP} =0.02$, $P_{DD}= 0.03$, $P_{COD}= 0.03$	40
HR15	COD	$P_{COD} = 0.02$	20
HR16	COD	$P_{COD} = 0.025$	20
HR17	COD	$P_{COD} = 0.025$	20
HR18	WTC, TST	$P_{WTC} = 0.045$, $P_{TST} = 0.04$	55
HR19	WTC, TST	$P_{WTC} = 0.04$, $P_{TST} = 0.04$	50
HR20	WTC, TST	$P_{WTC} = 0.045$, $P_{TST} = 0.035$	45
HR21	TST	$P_{TST} = 0.035$	40
HR22	TST	$P_{TST} = 0.03$	20

The activities are described in Table 4. For convenience, the units of the *SIZE* column are represented as "*KLOC*". From the capability match, the capable human resources for each activity are also listed in Table 4. From Table 4, there are several resource competitions such as activity RA1, URA2 and RA3 which all require resources HR1-HR5. The length of the capable human resource gene in the chromosome is 85.

Table 4. The attributes and capable human resources of the activity

	TYPE	SIZE	PREA	Capable Human Resource
RA1	RA	25	No element exist	HR1, HR2, HR3, HR4, HR5
AD1	AD	25	RA1	HR6, HR7, HR8, HR9
IMP1a	IMP	10	AD1	HR10, HR11, HR12, HR13, HR14
IMP1b	IMP	15	AD1	HR10, HR11, HR12, HR13, HR14
WTC1	WTC	25	RA1	HR18, HR19, HR20
TST1	TST	25	IMP1a,IMP1b,WTC1	HR18, HR19, HR20, HR21, HR22
URA2	RA	10	No element exist	HR1, HR2, HR3, HR4, HR5
IMP2	IMP	10	URA2	HR10, HR11, HR12
TST2	TST	10	IMP2	HR18, HR19, HR20, HR21, HR22
RA3	RA	45	No element exist	HR1, HR2, HR3, HR4, HR5
AD3	AD	45	RA3	HR6, HR7, HR8, HR9
DD3a	DD	10	AD3	HR6,HR7,HR8,HR9,HR10,HR11,HR12,HR13,HR14
DD3b	DD	20	AD3	HR6,HR7,HR8,HR9,HR10,HR11,HR12,HR13,HR14
DD3c	DD	15	AD3	HR6,HR7,HR8,HR9,HR10,HR11,HR12,HR13,HR14
COD3a	COD	10	DD3a	HR10,HR11,HR12,HR13,HR14,HR15, HR16, HR17
COD3b	COD	20	DD3b	HR10,HR11,HR12,HR13,HR14,HR15, HR16, HR17
COD3c	COD	15	DD3c	HR10,HR11,HR12,HR13,HR14,HR15, HR16, HR17
WTC3	WTC	45	RA3	HR13, HR14, HR15
TST3	TST	45	COD3a, COD3b, COD3c, WTC3	HR13, HR14, HR15, HR16, HR17

5.2 Simulating Run of the Scheduling and Analysis

For running the GA, we set the following value for the parameters:

Population scale: 32.

Priority gene size: 3, thus the length of the chromosome is: CL = 85 + 3*19 = 142.

Mutation rate: 0.01.

Maximum generation: 500

Based on the descriptions of projects and human resources in section 5.1, the scheduling by using GA is performed. Fig.3 shows the organization value obtained in each generation of three simulation runs of the algorithm. As the number of generation increases, the value increases and will obtain the (near-) maximum value for the organization.

After the scheduling, the results of schedule, cost and value of the projects for one run are described in Table 5.

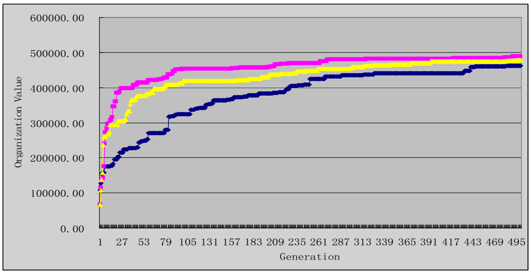


Fig. 3. Organization value in each generation

Table 5. Scheduling result

	Actual Schedule	Actual Cost	Value
P1	[2008-3-1, 2008-7-25]	166680	64640
P2	[2008-5-10, 2008-6-26]	46680	11320
P3	[2008-3-1, 2008-10-24]	303600	96300

In order to demonstrate the effectiveness of the value function for the scheduling, we will change the project and schedule preference weight, simulating the scheduling on the basis of the values generated by these parameters.

5.2.1 Simulation Results for Different Project Preference Weights

If the preference weight of a project increases, the impact on the organization value of this project will increase simultaneously. Fig. 4 shows the value of projects (primary axis) and organization value contributed by P1 and P2 (secondary axis) in one simulation run when P3's preference weight increases and other coefficients and preference weights are unaltered.

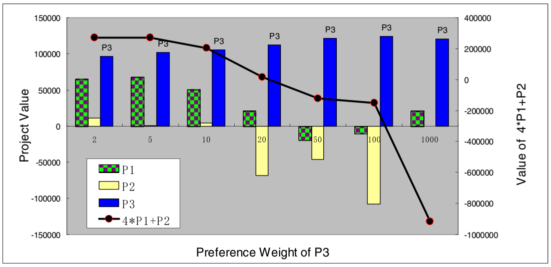


Fig. 4. Project value affected by P3's PPW

From Fig.4, when P3's preference weight increases, P3 has the increasing priority for the human resources to make its value maximum. Organization value contributed by P1 and P2 decreases (donated by the broken line). However, when the project weight of P3 reaches a high value (such as 50 in the figure), this value will not further increase distinctly. Since the organization only has limited resources, all the resources are already scheduled to P3 when the weigh reaches the high value.

5.2.2 Simulation Results for Different Schedule Preference Weights

If the schedule preference weight of one project grows, the impact of the schedule on the project value will increase simultaneously. Fig. 5 shows the precedent number of dates of three projects (primary axis) and the saved cost of P3 (secondary axis) in one simulation run when P3's schedule preference weight increases and other coefficients and preference weights are unaltered.

From *Fig.5*, the precedent number of dates of P3 increases when the schedule preference weight grows. On the contrary, the saved cost of P3 decreases (donated by the broken line). However, when the schedule weight of P3 reaches to a high value (such as 50 in the figure), the precedent number of dates of P3 will not further increase distinctly for the same reason we have explained in 5.2.1. When the schedule weight reaches 1000, since the increased value of P3 is higher than the penalty of P2's failure, the schedule for P2 fails.

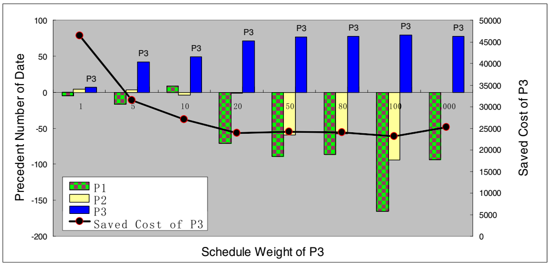


Fig. 5. Precedent number of dates according to P3's SPW

5.3 Benefit Discussions

From the scheduling method introduced in this paper and the simulation results generated under various conditions, the following benefits can be obtained:

(1) **A value function is defined**: it takes into account the constraints and preferences of different projects; it can be used to balance the constraints and value objectives among different projects in the organization.

(2) **The scheduling results can reflect the value objectives of the organization**: the coefficients and preference weights in the value function are set by the organization according to the value objectives. Through the value function, the scheduling results will reflect the value objectives of the organization.

(3) **Provide the decision support for project managers**: by setting different coefficients and preference weights, project managers can compare the results of the resource scheduling easily. Therefore, they can determine whether their project adds value to their organization.

6 Conclusions and Future Work

From the value-based view, this paper provides a multi-project scheduling method by using a genetic algorithm. The value function takes full consideration of the essential elements that affect the optimizing goal of scheduling such as schedule and cost. Based on this value function, the multi-project human resource scheduling method by using a genetic algorithm is implemented, which allows the organization to obtain a near-maximum value. Case study shows the method can take into account the value objectives of the organization that uses this method and effectively reflect the organization value and provide decision support for managers.

One premise of this paper is that the capabilities of the human resource are stable and deterministic. In the future, the learning curve will be taken into consideration. Another premise is that the number of the human resources who participate in one activity does not reflect the effort for communication. In future, factors related to communication will be taken into account. Overwork of human resources is another potential research issue (that is, some people will work more than 8 hours per day). The comparison of GA with other algorithms as well as analysis and justification of GA parameters such as population scale and mutation rate are also part of the future work.

Acknowledgments. Supported by the National Natural Science Foundation of China under grant Nos. 90718042, the Hi-Tech Research and Development Program (863 Program) of China under grant No.2007AA010303, 2007AA01Z186, as well as the National Basic Research Program (973 program) under grant No. 2007CB310802.

References

[1] Biffl, S., Aurum, A., Boehm, B., Erdogmus, H., Grünbacher, P.: Value-Based Software Engineering. Springer, Heidelberg (2005)

[2] Nejmeh, B.A., Thomas, I.: Business-Driven Product Planning Using Feature Vectors and Increments. IEEE Software, 34–42 (2002)

[3] Amandeep, Ruhe, G., Stanford, M.: Intelligent Support for Software Release Planning. In: Bomarius, F., Iida, H. (eds.) PROFES 2004. LNCS, vol. 3009, pp. 248–262. Springer, Heidelberg (2004)

[4] Pinedo, M.: Scheduling: Theory, Algorithms, and System, 2nd edn. Pearson Education, Inc., London (2005)

[5] Duggan, J., Byrne, J., Lyons, G.J.: Task Allocation Optimizer for Software Construction. IEEE Software, 76–82 (2004)

[6] Ruhe, G., Saliu, M.O.: The Art and Science of Software Release Planning. IEEE Software, 47–53 (2005)

[7] Bagnall, A.J., Rayward-Smith, V.J., Whittley, I.M.: The next release problem. Information and Software Technology 43, 883–890 (2001)

[8] Kapur, P., Ngo-The, A., Ruhe, G., Smith, A.: Optimized Staffing for Product Releases and Its Application at Chartwell Technology. Journal of Software Maintenance and Evolution: Research and Practice 20, 365–386 (2008)

[9] Abdel-Hamid, T.K.: The Dynamics of Software Project Staffing: A System Dynamics Based Simulation Approach. IEEE Transactions on Software Engineering 15, 109–119 (1989)

[10] Antoniol, G., Lucca, G.A.D., Penta, M.D.: Assessing Staffing Needs for a Software Maintenance Project through Queuing Simulation. IEEE Transactions on Software Engineering 30, 43–58 (2004)

[11] Goncalves, J.F., Mendes, J.J.M., Resende, M.G.C.: A Genetic Algorithm for the Resource Constrained Multi-project Scheduling Problem. European Journal of Operational Research 189, 1171–1190 (2008)

[12] Alba, E., Chicano, J.F.: Software Project Management with GAs. Journal of Information Sciences 177, 2380–2401 (2007)

[13] Barreto, A., Barros, M.d.O., Werner, C.M.L.: Staffing a software project: A constraint satisfaction and optimization-based approach. Computer & Operations Research 35, 3073–3089 (2008)

[14] Chang, C.K., Christensen, M.: A Net Practice for Software Project Management. IEEE Software (November/December 1999)

[15] Holland, J.H.: Adaptation in natural and artificial systems. MIT Press, Cambridge (1992)

[16] Xiao, J., Wang, Q., Li, M., Yang, Y., Zhang, F., Xie, L.: A Constraint-Driven Human Resource Scheduling Method in Software Development and Maintenance Process. In: Proceedings of 24th International Conference on Software Maintenance (ICSM 2008), pp. 17–26 (2008)

Meta Model Based Architecture for Software Process Instantiation

Peter Killisperger[1,2], Markus Stumptner[1], Georg Peters[3], Georg Grossmann[1], and Thomas Stückl[4]

[1] Advanced Computing Research Centre, University of South Australia, Adelaide, Australia

[2] Competence Center Information Systems, University of Applied Sciences - München, Germany

[3] Department of Computer Science and Mathematics, University of Applied Sciences - München, Germany

[4] System and Software Processes, Siemens Corporate Technology, München, Germany

Abstract. In order to re-use software processes for a spectrum of projects they are described in a generic way. Due to the uniqueness of software development, processes have to be adapted to project specific needs to be effectively applicable in projects. This instantiation still lacks standardization and tool support making it error prone, time consuming and thus expensive. Siemens AG has started research projects aiming to improve software process related activities. Part of these efforts has been the development of a New Software Engineering Framework (NSEF) enabling a more effective and efficient instantiation and application of processes. A system supporting project managers in instantiation of software processes is being developed. It aims to execute instantiation decision made by humans and to automatically restore correctness of the resulting process.

1 Introduction

Explicitly defined software processes for the development of software are used by most large organizations. A software process is defined as "the process or the processes used by an organization or project to plan, manage execute, monitor, control and improve its software related activities" [1].

At Siemens AG, business units define software processes within a company-wide Siemens Process Framework (SPF) [23] by using semi-formal Event-Driven Process Chains (EPC) and Function Allocation Diagrams (FAD) [22]. Because of their size and complexity, they are not defined for projects individually but in a generic way as reference processes for application in any software project of the particular business unit.

Due to the individuality of software development, reference processes have to be instantiated to be applicable in projects. That is, the generic description of the process is specialized and adapted to the needs of a particular project. Until now, reference processes are used as general guideline and are instantiated only minimally by manual creation of project specific plans. A more far reaching

Q. Wang et al. (Eds.): ICSP 2009, LNCS 5543, pp. 63–74, 2009.

instantiation is desirable, because manual instantiation is error-prone, time consuming and expensive due to the complexity of processes and due to constraints of the Siemens Process Framework. A New Software Engineering Framework (NSEF) [19] has been defined for improving current practice. An integral part of the NSEF is gradual instantiation of software processes to project specific needs. Here we define *instantiation* as tailoring, resource allocation and customization of artifacts.

- *Tailoring* is "the act of adjusting the definitions and/or of particularizing the terms of a general process description to derive a new process applicable to an alternative (and probably less general) environment" [15].
- *Resource allocation* is the assignment of resources to activities to carry them out [2].
- *Customization of artifacts* is the individualization of general artifacts for a project and their association with files implementing them.

The area of project specific composition and adaptation of software processes and methods has attracted significant attention in recent years as in, e.g., Brinkkemper's Method Engineering (ME) proposal [12] as an approach for the creation of situational methods. However, no existing approach provides a thorough solution for instantiating Siemens processes. For example, Brinkkemper's situational method configuration process emphasized bottom-up assembly of project specific methods from fragments, requiring very detailed fragment specifications. Contrary to ME, approaches like Bandinelli's SLANG [6] regard processes as programs, enacted by machines [13]. Here however, we are concerned with flexible method engineering in the large and deal with semi-formal process models offering high level guidance for humans.

Existing tools provide only minimal support for instantiation. Decisions made by humans have to be executed mostly manually. For example, Rational's Method Composer (RMC) [16] allows changes on software processes, but although approaches have been developed making the user aware of inconsistencies caused by instantiation operations [17], the actual correction of the process is still left to humans.

For example, consider an activity a_1 connected by an control flow to an activity a_2 which in turn is connected by an control flow to an activity a_3 ($a_1 \rightarrow a_2 \rightarrow a_3$). If a project manager wants to delete a_2 he selects a_2 and removes it from the process. Additionally, he has to take care of restoring correctness e.g. establish the broken control flow between a_1 and a_3, take care of affected information flows and resource connections.

In order to noticeably reduce the considerable effort for instantiation, tool support has to be extended. The goal is to derive a flexible architecture for systems that execute instantiation decisions made by humans and automatically restore correctness of the resulting process. We define a process to be *correct* when it complies with the restrictions on the process defined in a method manual. A method manual is a meta model defining permitted elements and constructs, derived from restrictions of the used process definition language and organizational restrictions (e.g. SPF).

The paper is structured as follows: Section 2 briefly introduces the NSEF and describes the fundamentals of its instantiation approach. Section 3 describes the developed architecture for systems implementing these theoretical fundamentals. A procedure for the actual development of such a system is defined, followed by an evaluation of our findings. Section 4 discusses related work and in section 5 we draw some conclusions.

2 New Software Engineering Framework

On the basis of information collected in interviews with practitioners at Siemens AG, a New Software Engineering Framework (NSEF) (Figure 1) for improving the instantiation and application of software processes has been developed [19].

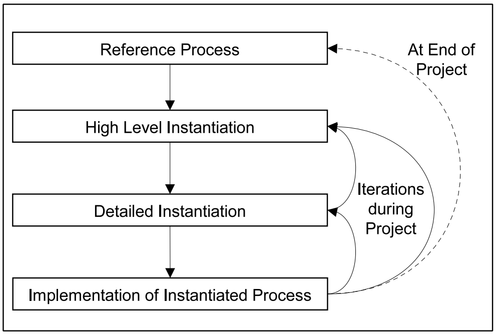


Fig. 1. Instantiation Stages in the New Software Engineering Framework (NSEF)

The NSEF consists of a *Reference Process*, gradual instantiation by *High Level* and *Detailed Instantiation* and an *Implementation of the Instantiated Process*. *High Level Instantiation* is a first step towards a project specific software process by adapting the *Reference Process* on the basis of project characteristics and information that can already be defined at the start of a project and are unlikely to change. Such characteristics can be, e.g., the size of a project (a small project will only use a subset of the process) or required high reliability of the software product which requires certain activities to be added.

High Level Instantiation is followed by *Detailed Instantiation* which is run frequently during the project for the upcoming activities. A step by step approach is proposed, because it is often unrealistic to completely define a project specific process already at the start of a project [8].

The resulting instantiated process can be used in projects in different ways including visualization of the process and management of project artifacts.

Although instantiation in the NSEF is split into two distinct stages, it is advantageous if both are based on the same principles. A set of elemental *Basic Instantiation Operations* have been defined which are used for both stages [18]. Examples are: "Deleting an Activity" or "Associating a Resource with an

Activity". In *High Level Instantiation*, *Basic Instantiation Operations* are executed on process elements as batch (i.e. predefined process adaptations depending on the project type) and in *Detailed Instantiation* individually. Using the same principles enables flexible definition and adaptation of predefined instantiation-batches. For instance, if a particular adaptation operation is not to be executed in *High Level Instantiation* any more, the operation can be easily shifted to detailed instantiation or vice versa.

Existing tools (as mentioned in the introduction) allow to perform those changes but they do not restore correctness of the resulting process.

The task of instantiation is further complicated by additional restrictions. Examples are: 'A *ManualActivity* is only allowed to be executed by a *Human*' or 'an *Artifact* has to be output of exactly one *Activity*'.

```
context ExecutesResourceConnection
   inv: (self.source.oclIsKindOf(ManualActivity)
        implies self.target.oclIsKindOf(Human))

context Artifact
   inv: self.outputInformationFlow->size()=1
```

The presented framework executes instantiation decisions made by humans and automatically restores correctness of the resulting process defined in a method manual.

For ensuring correctness, the framework considers the environment the change is performed in. For instance, if milestones have restrictions on the type of element they can succeed and precede, obviously correctness depends on the preceding and successive element of the milestone. Thus, when inserting a milestone every possible combination of preceding and successive elements has to be considered. A set of entities and relationships outside the actually affected process step that influence the correctness of a change operation is what we call a context of the operation. The contexts an operation can be executed in can be automatically derived from the method manual.

In order to guarantee correctness, every context has to be associated with an particular implementation which executes a basic instantiation operation in this explicit context. However, not every context requires an individual implementation that is only applicable in this particular context. From this follows that a number of contexts can be associated with one implementation (Figure 2).

The approach described above is a general framework for instantiation of processes and is not restricted to Siemens processes or a particular process definition language. The way the framework is implemented in detail depends on the requirements of the applying organization and on the method manual of the process. That means the required basic instantiation operations, their definition (i.e. how the process is adapted by the operations) and the way correctness is restored depend on the applying organization and on the method manual.

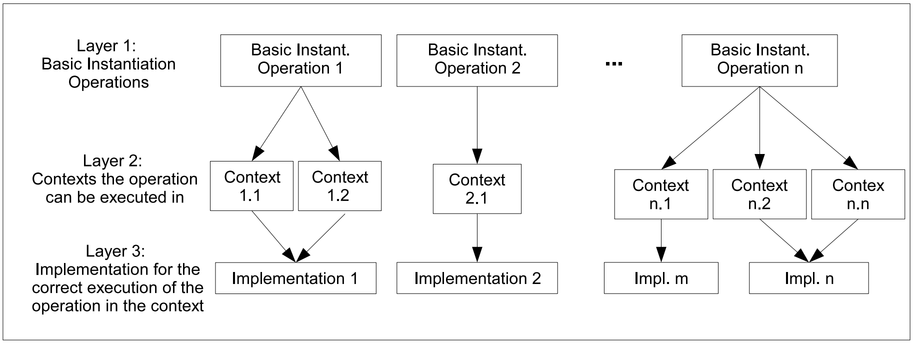

Layer 1:
Basic Instantiation Operations
Basic Instant. Operation 1
Basic Instant. Operation 2
...
Basic Instant. Operation n
Layer 2:
Contexts the operation can be executed in
Context 1.1
Context 1.2
Context 2.1
Context n.1
Context n.2
Contex n.n
Layer 3:
Implementation for the correct execution of the operation in the context
Implementation 1
Implementation 2
Impl. m
Impl. n

Fig. 2. 3-layer instantiation approach

3 Instantiation System Architecture

3.1 Architecture Meta Model

An instantiation system implementing the concept described above has to meet a number of requirements:

- The architecture of an instantiation system has to support differing method manuals, differing basic instantiation operations and thus differing contexts and implementations, since instantiation is organization specific and depends on the method manual of the process.
- An instantiation system has to be flexible to allow adaptations due to changes in the method manual or changes in the extend an organization wants to instantiate its processes.
- Contexts have to be modularized in order to avoid multiple implementations of parts of contexts, since they are likely to be similar for a number of basic instantiation operations. For example, "Deleting an Activity" is likely to have similar contexts to "Inserting an Activity".
- Implementations of basic instantiation operations have to be modular in order to avoid multiple development efforts, since parts or even whole implementations might be used in several contexts.

By taking into account the requirements defined above, the architecture described in Figure 3 has been developed. It consists of the components *ProcessInstance*, *Element*, *Operation*, *Implementation*, *Operator*, *Condition* and *Method Manual*. In the following we describe the components of the architecture.

The class *ProcessInstance* corresponds to a project-specific software process, consisting of a number of elements. A process instance is created by copying the reference process for a particular project. Before instantiation (i.e. at the start of a project) the process instance for a project is equal to the reference process.

Instances of the class *ProcessElement* are the elemental building blocks of a *ProcessInstance*. They can be of different types. Examples of the type "Activity" are "Develop Design Specification" or "Implement Design". Elements and thus

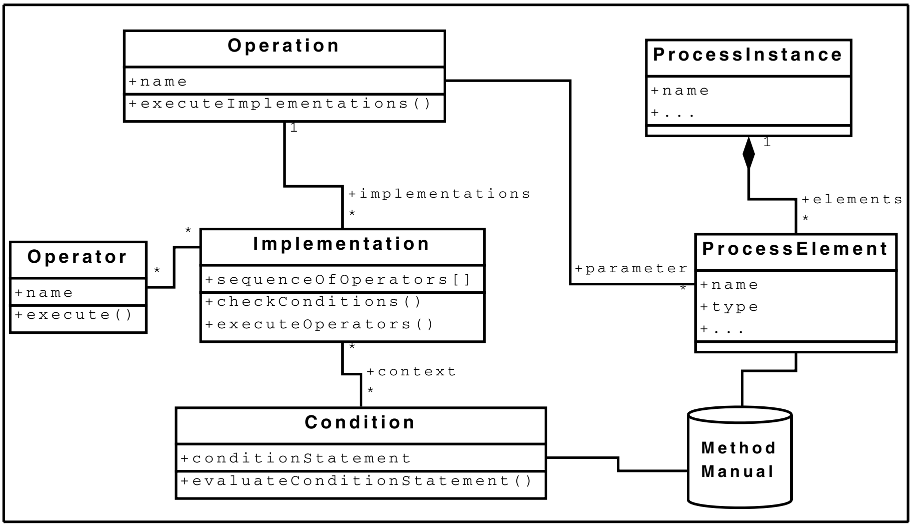


Fig. 3. Architecture for Instantiation Systems

process instances are adapted to project specific needs by running instances of *Operation* on them.

The class *Operation* correspond to basic instantiation operations. How a basic instantiation operation is executed on elements depends on the context in which the affected elements are nested in the process instance. Therefore instances of *Operation* can be associated with more than one instance of *Implementation*. When an *Operation* is executed, the *Implementation* which relates to the context at hand is executed. This adapts the *ProcessInstance* according to the *Operation* for the specific context and results in a correct *ProcessInstance*.

For finding an *Implementation* that relates to a specific context *Operation. executeImplementations()* calls *Implementation.checkConditions()* for each of its implementations. If "true" is returned, the correct implementation has been found.

The actual process adaptations are carried out by a sequence of operators (*squenceOfOperators[]* in *Implementation*) which execute the actual changes in the process instance. Operators are elemental actions which create, update and remove elements of a process instance. Examples are the creation of a new control flow or the creation of a new activity. The entirety of operators is stored in a repository. Their number is not fixed but can be further extended if necessary. The use of operators enables encapsulation which fosters re-usability and information hiding. The former is the use of an instance of *Operator* by several implementations. The latter allows experts with limited or no technical knowledge to create or change implementations since technical details are hidden.

As described earlier, the implementation of an operation depends on the context the operation is executed in. A context consists of one or several elemental *Condition*-instances. Examples of conditions are <self.source.type = 'activity'> (type of source element is an activity) or <self.performer.type = 'human'>

(type of performer of the activity is "human"). In order for a context to be true, all conditions associated with the context have to be evaluated *true.* As with operators, the entirety of conditions is stored in a repository. However, they do not have to be defined by humans and their number is limited. Conditions are elemental statements about states and relationships of elements. The method manual defines the set of allowed statements for elements of a process instance. Conditions are therefore automatically derived. The use of elemental conditions provides encapsulation enabling re-use and information hiding.

Figure 4 shows a class diagram with the same structure as the one depicted in Figure 3 but extended by examples. An *ProcessInstance*-instance has been created by copying the reference process for a particular project "xyz". Since it is not needed in this project, the project manager wants to delete the *ProcessElement* "Check Patent Application". He runs the *Operation* "Delete Activity" on it by executing *executeImplementations().* The method runs through all implementations of "Delete Activity" and executes *checkConditions()* for each until one of the implementations returns "true". *checkConditions()* runs through the *Condition*-instances of the context and executes *evaluatedConditionsStatement().* If all method calls return "true" the *Implementation*-instance at hand is responsible for executing the deletion of "Check Patent Application". It then calls *executeOperators()* which in turn calls *execute()* of all operators. By doing so the activity is deleted and correctness restored.

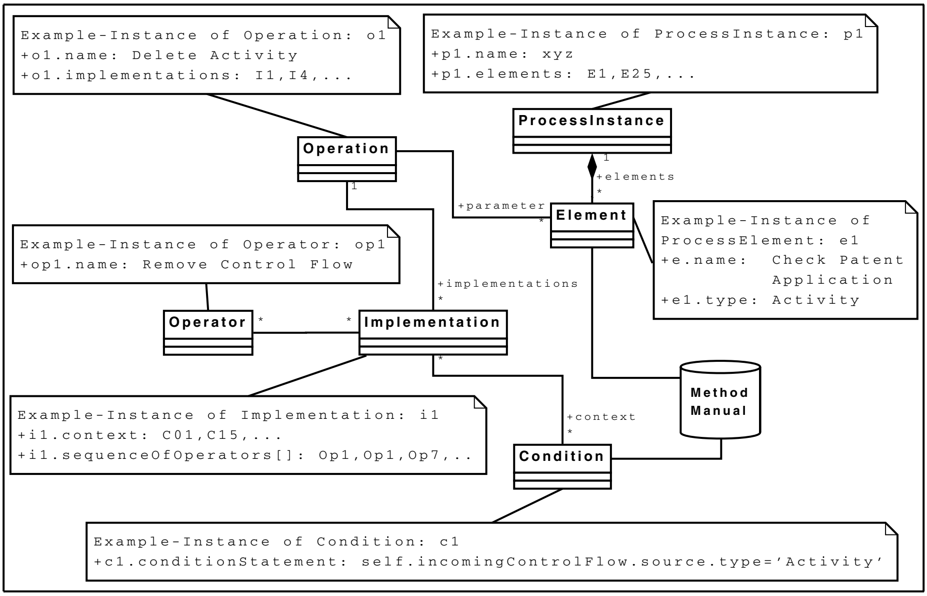

Example-Instance of Operation: o1
+o1.name: Delete Activity
+o1.implementations: I1,I4,...
Example-Instance of ProcessInstance: p1
+p1.name: xyz
+p1.elements: E1,E25,...
ProcessInstance
Operation
1
+elements
*
+parameter
*
Element
1
Example-Instance of Operator: op1
+op1.name: Remove Control Flow
+implementations
*
Example-Instance of
ProcessElement: e1
+e.name: Check Patent
Application
+e1.type: Activity
Operator
*
*
Implementation
*
Method
Manual
Example-Instance of Implementation: i1
+i1.context: C01,C15,...
+i1.sequenceOfOperators[]: Op1,Op1,Op7,..
+context
*
Condition
Example-Instance of Condition: c1
+c1.conditionStatement: self.incomingControlFlow.source.type='Activity'

Fig. 4. Example of Implementation of Architecture

3.2 Implementing the Architecture

The architecture can be used to instantiate a variety of software processes. The implementation of a system will differ depending on the method manual and on the extent an organization wants to instantiate its processes. For implementing an instantiation system for a particular organization or for a particular method manual the following has to be considered.

1. Definition of operations differ between organizations or might change after some time within one organization. Implementations (i.e. sequences of operators) have to be defined executing operations as defined.
2. Organizations use differing method manuals. From this follows that an operation can be executed in different contexts.
3. Differing method manuals affect the way an operation is executed (i.e. the sequence of operators has to result in a process that complies with the method manual).
4. Diversity in sequences of operators may require development of new operators if they are not in the repository yet.

From this follows that organizations have to be supported in implementing the architecture for their use. The architecture must therefore provide functionality to facilitate the development of an organization specific instantiation system.

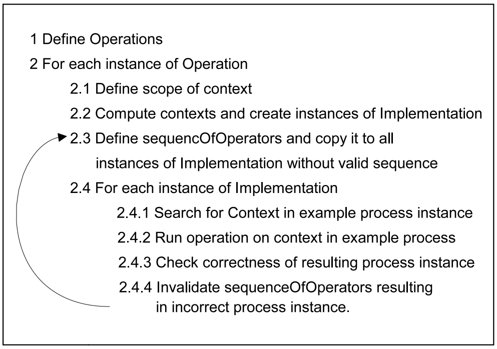
1. Offer functionality for domain experts to implement new operations and change the implementation of existing operations.
2. Offer functionality to compute all contexts an operation can be executed in and for creating instances of *Implementation* for each context.
3. Offer functionality to domain experts to define and change sequences of operators. Functionality is required to test whether the developed sequence of operators results in a correct process when executing the operation in a particular context.
4. Offer functionality to add new operators to the repository.

For implementing a new instantiation system for an organization the following procedure is suggested (Figure 5).

In a first step, domain experts have to define the operations required for instantiation of processes in their organization. Then, for each operation the scope of its context has to be defined which is the abstraction of all contexts the operation can be executed in.

For example, the operation ”inserting an activity” might depend on the type of source element of the selected control flow where the activity is to be inserted and the type of target element of this control flow. Thus, the scope is the type of source element and the type of target element of the selected control flow.

Since the scope of each context is described for each context of an operation, all possible contexts the operation can be executed in can be computed and created. For each operation a standard sequence of operators is defined by domain experts and used in all instances of implementations for this operation.



```
1 Define Operations
2 For each instance of Operation
    2.1 Define scope of context
    2.2 Compute contexts and create instances of Implementation
    2.3 Define sequencOfOperators and copy it to all
        instances of Implementation without valid sequence
    2.4 For each instance of Implementation
        2.4.1 Search for Context in example process instance
        2.4.2 Run operation on context in example process
        2.4.3 Check correctness of resulting process instance
        2.4.4 Invalidate sequenceOfOperators resulting
              in incorrect process instance.
```

Fig. 5. Procedure for developing an instantiation system

Each implementation is tested by looking up its context in an example process instance (containing all elements and constructs allowed by the method manual) and executing its sequence of operators on it. The correctness of the resulting process instance is checked and all sequences of operators are invalidated which resulted in an incorrect process. For invalidated implementations an alternative sequence of operators is defined by domain experts. This procedure is iterated until there are no implementations which result in an incorrect process when executed in their context.

3.3 Evaluation

A prototype of a system for instantiating a reference process for a particular Siemens business has been developed. In order to enable automatic processing, the reference process has been exported to XPDL 2.0 [24] and from the textual method manual a machine readable version in XML has been created. Two exemplary basic instantiation operations namely "Inserting an Activity" and "Deleting an Activity" have been chosen for testing and the architecture has been implemented accordingly.

For the operation "Inserting an Activity" the system automatically computed 43 possible contexts from the information given in the method manual and created for each context an instance of *Implementation*. Experts at Siemens AG defined a standard sequence of operators for executing the insertion of an activity. Each implementation was executed with this sequence. The resulting process instances were checked by a debugger-class regarding their conformity based on an extended version of the XPDL standard and on the basis of the XML-based method manual. The tests showed no violations of the resulting process instances.

For the operation "Deleting an Activity" the same procedure was chosen. The system identified 49 contexts and created the corresponding instances of *Implementation*. After executing all implementations with a first sequence of operators, 44 resulted in a correct process. Violations of the remaining implementations were written with clarifying comments to a text file and solved by iteratively developing the correct sequences of operators for them.

4 Related Work

Instantiation of processes to project specific needs has been subject to intensive research in recent years. However, in early software process approaches it was thought that a perfect process can be developed which fits all software developing organizations and all types of projects [10]. It was soon recognized that no such process exists [7], [20]. This lead to the introduction of reference processes as general guidelines which are adapted for specific project needs.

Early approaches to overcome this problem have been developed for example by Boehm and Belz [10] and Alexander and Davis [3]. The former used the Spiral Model to develop project specific software processes. The latter described 20 criteria for selecting the best suited process model for a project.

Many different adaptation approaches have been proposed since then and the need for adaptation of processes is recognized in industry which is shown by a high number of publications about tailoring approaches in practice e.g. Bowers et al. [11], Fitzgerald et al. [14].

Although much effort has been put into improving the adaption of software processes to project specific needs, the approaches proposed so far still suffer from important restrictions and none has evolved into an industry accepted standard.

An important reason is the variety of meta models for processes used in practice. For instance, Yoon et al. [25] developed an approach for adapting processes in the form of Activity-Artifact-Graphs. Since the process is composed of activities and artifacts, only the operations "addition" and "deletion" of activities and artifacts are supported as well as "split" and "merge" of activities. Another example is the V-Model [9], a process model developed for the German public sector. It offers a toolbox of process modules and execution strategies. The approach for developing a project specific software process is to select required process modules and an execution strategy. Due to these dependencies on the Meta models, none of the existing approaches offers a complete and semi-automated method.

Because of the close relationship between Siemens software and business processes, adaptation approaches for the latter are also of interest. Approaches for processes and workflows of higher complexity are often restricted to only a subset of adaptation operations. For instance, Rosemann and van der Aalst [21] developed configurable EPCs (C-EPCs) enabling the customization of reference processes. However, the approach only allows activities to be switched on/off, the replacement of gateways and the definition of dependencies of adaptation decisions.

Armbrust et al. [5] developed an approach for the management of process variants. A process is split up in stable and variant parts. The latter depend on project characteristics and are not allowed to be dependent on each other. The process is adapted by choosing one variant at the start of a project. Although the need for further adaptations during the execution of the process has been identified, no standardization or tool support is provided.

Allerbach et al. [4] developed a similar approach called Provop (Process Variants by Options). Processes are adapted by using the change operations *insert*, *delete*, *move* and *modify attributes* which are grouped in Options. Options have to be predefined and can be used to adapt processes, but they do not guarantee correctness.

In conclusion, none of the existing approaches offer a comprehensive, flexible and semi-automated adaption of processes as required for the diversity of processes and software development encountered in large enterprises.

5 Conclusion

Siemens is currently undertaking research efforts to improve their software process related activities. Part of these efforts is the development of a system that supports project managers in instantiation of reference processes. The system aims not only to execute decisions but to restore correctness of the resulting process when violated by the execution of the decision. Since the implementation of such a system is organization-specific and depends on the permitted elements and constructs in the process, a flexible architecture has been developed and described. A procedure for implementing the architecture was described and the feasibility of the developed concepts verified by the implementation of a prototype. Future work will include enhancement of the prototype and its evaluation in software development projects at Siemens AG.

Acknowledgements

This work was partially supported by a DAAD postgraduate scholarship.

References

1. ISO/IEC 15504-9 Tech. Software Process Assessment Part 9: Vocabulary (1998)
2. Aalst, W.v.d., Hee, K.v.: Workflow Management-Models, Methods, and Systems. The MIT Press, Cambridge (2004)
3. Alexander, L.C., Davis, A.M.: Criteria for Selecting Software Process Mod- els. In: Proceedings of the Fifteenth Annual International Computer Software and Applications Conference, pp. 521–528 (1991)
4. Allerbach, A., Bauer, T., Reichert, M.: Managing Process Variants in the Process Life Cycle. In: Proceedings of the Tenth International Conference on Enterprise Information Systems. ISAS, vol. 2, pp. 154–161 (2008)
5. Armbrust, O., Katahira, M., Miyamoto, Y., Münch, J., Nakao, H., Ocampo, A.: Scoping Software Process Models - Initial Concepts and Experience from Defining Space Standards. In: Wang, Q., Pfahl, D., Raffo, D.M. (eds.) ICSP 2008. LNCS, vol. 5007, pp. 160–172. Springer, Heidelberg (2008)
6. Bandinelli, S., Fuggetta, A.: Computational Reflection in Software Process Modeling: The SLANG Approach. In: ICSE, pp. 144–154 (1993)
7. Basili, V.R., Rombach, H.D.: Support for comprehensive reuse. Software Engineering Journal 6(5), 303–316 (1991)
8. Becker, U., Hamann, D., Verlage, M.: Descriptive Modeling of Software Processes. In: Proceedings of the Third Conference on Software Process Improvement, SPI 1997 (1997)

9. BMI. The new V-Modell XT - Development Standard for IT Systems of the Federal Republic of Germany (2004), http://www.v-modell-xt.de (accessed 01.12.2008)
10. Boehm, B., Belz, F.: Experiences With The Spiral Model As A Process Model Generator. In: Proceedings of the 5th International Software Process Workshop Experience with Software Process Models, pp. 43–45 (1990)
11. Bowers, J., May, J., Melander, E., Baarman, M.: Tailoring XP for Large System Mission Critical Software Development. In: Wells, D., Williams, L. (eds.) XP 2002. LNCS, vol. 2418, pp. 100–111. Springer, Heidelberg (2002)
12. Brinkkemper, S.: Method engineering: engineering of information systems development methods and tools. Information & Software Tech. 38(4), 275–280 (1996)
13. Feiler, P.H., Humphrey, W.S.: Software Process Development and Enactment: Concepts and Definitions. In: ICSP, pp. 28–40 (1993)
14. Fitzgerald, B., Russo, N., O'Kane, T.: An empirical study of system development method tailoring in practice. In: Proceedings of the Eighth European Conference on Information Systems, pp. 187–194 (2000)
15. Ginsberg, M., Quinn, L.: Process tailoring and the software Capability Maturity Model. Technical report, Software Engineering Institute, SEI (1995)
16. IBM. Rational Method Composer (2008), http://www-01.ibm.com/software/awdtools/rmc/ (accessed 26.11.2008)
17. Kabbaj, M., Lbath, R., Coulette, B.: A Deviation Man- agement System for Handling Software Process Enactment Evolution. In: ICSP, pp. 186–197 (2008)
18. Killisperger, P., Peters, G., Stumptner, M., Stückl, T.: Instantiation of Software Processes, An Industry Approach. In: Information Systems Development: Towards a Service Provision Society. Springer, Heidelberg (2008) (forthcoming)
19. Killisperger, P., Stumptner, M., Peters, G., Stückl, T.: Challenges in Software Design in Large Corporations A Case Study at Siemens AG. In: Proceedings of the Tenth International Conference on Enterprise Information Systems. ISAS, vol. 2, pp. 123–128 (2008)
20. Osterweil, L.J.: Software Processes Are Software Too. In: ICSE, pp. 2–13 (1987)
21. Rosemann, M., van der Aalst, W.: A Configurable Reference Modelling Language. Information Systems 32(1), 1–23 (2007)
22. Scheer, A.-W.: ARIS- business process modelling. Springer, Heidelberg (2000)
23. Schmelzer, H.J., Sesselmann, W.: Geschäftsprozessmanagement in der Praxis: Produktivität steigern - Wert erhöhen - Kunden zufrieden stellen, 4th edn. Hanser Verlag, Muenchen (2004)
24. WFMC. WFMC-TC-1025-03-10-05 Specification for XPDL v2.0 (2005), http://www.wfmc.org (accessed: 28.11.2008)
25. Yoon, I.-C., Min, S.-Y., Bae, D.-H.: Tailoring and Verifying Soft- ware Process. In: APSEC, pp. 202–209 (2001)

Distributed Orchestration Versus Choreography: The FOCAS Approach

Gabriel Pedraza and Jacky Estublier

LIG, 220 rue de la Chimie, BP53
38041 Grenoble Cedex 9, France
{Gabriel.Pedraza-Ferreira,Jacky}@imag.fr

Abstract. Web service orchestration is popular because the application logic is defined from a central and unique point of view, but it suffers from scalability issues. In choreography, the application is expressed as a direct communication between services without any central actor, making it scalable but also difficult to specify and implement. In this paper we present FOCAS, in which the application is described as a classic service orchestration extended by annotations expressing where activities, either atomic or composite, are to be executed. FOCAS analyzes the orchestration model and its distribution annotations and transforms the orchestration into a number of sub-orchestrations to be deployed on a set of distributed choreography servers, and then, deploys and executes the application. This approach seemingly fills the gap between "pure" orchestration (a single control server), and "pure" choreography (a server per service). The paper shows how FOCAS transforms a simple orchestration into a distributed one, fitting the distribution needs of the company, and also shows how choreography servers can be implemented using traditional orchestration engines.

1 Introduction

Web services represent the ultimate evolution towards loose coupling, interoperability, distribution and reuse. Among the properties of interest, Web services do not have explicit dependencies, i.e., from a client's point of view a Web service does not call other Web services, therefore Web services are context independent, which greatly improves their reuse capability [1]. Conversely, building applications based on Web services is not easy since each Web service is independent and only answers to client requests [2]. A way to define such an application is to express the set of web service invocations. Typically, the result returned by a Web service, after transformation (if required), is fed as input to other Web services.

With the aim of specifying Web service-based applications, orchestration and choreography have been defined roughly at the same time.

In the orchestration approach, the application definition is a workflow model, i.e. a graph where nodes are web service executions and arcs are data flows. In [3] it has been defined as "an executable business process that can interact with both internal and external Web services". It means that a workflow engine interprets the orchestration model and calls the web services, with the right parameters at the right moment. This approach has many strong points, being the major one the availability of

Q. Wang et al. (Eds.): ICSP 2009, LNCS 5543, pp. 75–86, 2009.

standards and implementations. The actual orchestration language definition standard is WS-BPEL [4]. A number of good implementations are available along with a number of associated tools like graphical editors, generators, code analyzers, optimizers, etc. The other main advantage of using an orchestration language is that the whole application is defined in a single piece of information (the workflow model), which abstracts away many technical implementation details (like communication protocols, data formats, and so on). Technically, the fact the model is interpreted by a single engine on a single machine eases the implementation, administration, and monitoring of the orchestration. Conversely, it means that a single machine is at the heart of the system, with all communication going to and coming from that machine, potentially becoming a bottleneck and a strong limitation to scalability.

Choreography expresses a Web service collaboration, not through a single workflow model, but “simply” by the set of messages that are to be exchanged between the web services [3] [5]. In this view, Web services are directly communicating with each other, and not through the central machine. This divides by two the number of messages to be exchanged and also eliminates the central machine. Clearly this approach has much better scalability properties than orchestration. In opposition, the application is defined as a set of messages which is low level and confusing. Since there is no global view of the application, modeling an application is very difficult, the standards that have been proposed to do so [6][7] are complex to use and understand. From the implementation point of view, choreography is problematic. Each machine is responsible for routing messages to the next Web service(s) without any global view. Choreography implicitly requires deploying code to all machines involved in order to execute message routing, which is not always possible. Among the disadvantages of this approach, any problem occuring during execution is very difficult to manage, and dynamic selection of services is not easy to perform. It is therefore not too surprising that no industrial strength implementations are available, years after being defined.

Obviously the “perfect” system should be a combination of both approaches, i.e. a service based application described by a centralized model, with a number of optional centralized services like administration and monitoring, but using an efficient and scalable execution model like in service choreography.

We believe that “pure” choreography suffers from too many limitations. The major one being that it is not always possible to install code on the machines running the involved Web Services, and the second one being that it prohibits dynamic Web service selection. We propose instead, a flexible decentralized orchestration execution, in which a “traditional” orchestration model is executed on an arbitrary number of nodes, i.e., *choreography servers*. It is up to an administrator to decide the level of distribution convenient for the application, and to decide on which machines the application should run. This approach seemingly fills the gap between “pure” orchestration (a single control server), and “pure” choreography (a server per web service). The paper shows how FOCAS transforms a simple orchestration into a choreography that fits the distribution needs of the company, and how choreography servers can be implemented using traditional orchestration engines.

The paper is organized as follows. In section 2, a general outline of the FOCAS approach is presented. Section 3 describes how the logical level for distributed orchestration is defined. In section 4, the physical level is presented. Section 5 relates our work with the existing literature. Finally, section 6 concludes the paper.

2 FOCAS: An Extensible Orchestration Framework

FOCAS (**F**ramework for **O**rchestration, **C**omposition and **A**ggregation of **S**ervices) is an environment dedicated to the development and execution of service-based applications. It is a model-based framework around a basic workflow framework. It is extensible in the sense that it can support either different functional domains, through model and metamodel composition [8], or different "non-functional" properties through annotations on orchestration models [9]. The annotation approach has been used, for example, to support security in orchestration [10]. FOCAS carefully separates a logical layer, in which the service based application is defined in abstract terms, and a physical layer in which the real services are dynamically selected and invoked.

This paper discusses how the FOCAS annotation mechanism has been used to support *"distributed orchestration"*.

2.1 FOCAS Architecture and Approach

Following a model driven approach, we believe that the needs are to:

- separately design and specify the application's business logic, without regard to the technical and implementation details;
- transform the specification into executable artifacts in order to fit the execution and administration requirements.

In FOCAS, this separation, between logical aspects and technical aspects of an application, is always performed.

Like most orchestration approaches, the logical layer relies on a workflow model. Unfortunately, current orchestration languages like WS-BPEL lack abstraction [11] (no decoupling between abstract and concrete services [12]), are not extensible [13], and are unable to express a number of non-functional concerns (transaction, security, etc). In FOCAS, the logical definition is made of the composition of different functional models expressing the need on different domains. In the default implementation, these domains are control (the workflow model), data (the information system) and service (the abstract services description).

Non-functional aspects of a business model can be expressed as annotations over the control (APEL) model. Each type of annotation is associated with a specific concern, and is handled by a number of tools and adaptors. In this paper we explain how we use an annotation technique in order to handle the distribution concern.

Concerning the physical layer, FOCAS allows a flexible mapping between abstract and concrete services. This allows, for instance, dynamically selecting the actual web service to be invoked, and transforming the logical data into the parameters required by that web service.

Because of lack of space, we only present the control model in this paper, i.e. the APEL formalism used in our choreography approach. More details about our orchestration definition can be found in [14].

2.2 FOCAS Logical Layer: APEL and Orchestration Models

APEL (Abstract Process Engine Language [15]) is used to express the control model in FOCAS. APEL is a high level process definition language containing a minimal set of concepts that are sufficient to understand the purpose of a process model.

The main concept in APEL is *activity*. An activity is a step in the process and results in an action being performed. The actual action to be performed is not defined in the process model and can be either a service invocation (a Web Service, DPWS service, an OSGi service), any kind of program execution (legacy, COTS), or even a human action. Activities in APEL can be composed of sub-activities. Sub-activities permit dealing with different abstraction levels in a model. Ports are the activity communication interface; each port specifies a list of expected products.

A Product is an abstract object (records, data, files, documents, etc) that flows between activities. Products are represented by variables having a name and a type and are simply symbolic names (e.g., *"client"* is a product of type *"Customer")*. This property does not define nor constrain the actual nature, structure or content of the real data that will circulate in the process. Dataflows connect output ports to input ports, specifying which product variables are being transferred between activities.

APEL has a graphical syntax. An activity (from outside) is represented as a rectangle with tiny squares on sides which denotes its ports. Internally an activity is represented as a rectangle containing sub-activities, and its ports are represented as triangles. This dual representation is used to navigate across a complex model composed of several levels of embedded activities. Finally, a dataflow is represented as a line that connects ports, and products are labels on dataflows.

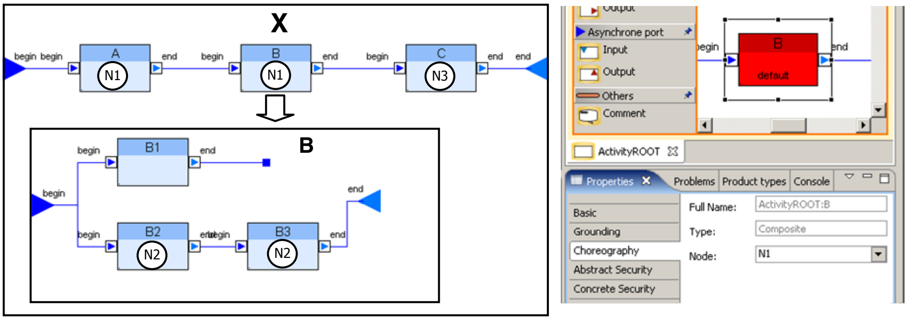
3 Logical Level: Service-Based Application Modeling

We believe that the real issue is not to build an application using orchestration or choreography, but to design, develop and execute service-based applications that fit a company's specific requirements. It is currently accepted that a centralized model describing the business logic, in terms of a workflow of abstract services, is a good way to design and specify service-based applications. Therefore, we believe that FOCAS, which uses a workflow-based and model-driven composition approach, is satisfactory, from a design point of view, for defining service-based applications.

3.1 Orchestration Annotations

Clearly, if scalability and efficiency are of concern, distributing the application execution should be addressed. However, being in a logical layer, distribution should also be addressed in abstract terms. To do so, we have designed a distribution domain which relies on an abstract server topology model and on orchestration annotations.

A server topology model takes the form of a graph where nodes are choreography servers, and arcs are communication links. Annotations are simply an association of a server identifier with an orchestration activity, either atomic or composite. In FOCAS, the graphical orchestration editor can be extended by annotations. The right part of Fig. 1 is a screenshot of a choreography extension. It shows when activity *B* is associated with node *N1*. Also shown are the security extensions created using the same mechanism.



Full Name: ActivityROOT:B
Type: Composite
Node: N1

Fig. 1. Annotations on a orchestration model

Fig. 1, left side, shows a simple orchestration model in APEL. The model, called X, is made of the activity sequence A, B and C. The activity *B* itself is made of three sub-activities, *B1*, *B2*, *B3*, where B1 and B2 are executed in parallel while *B3* is executed after the termination of *B2*. For clarity, the annotations are indicated on the schema by the numbered circles; activities *A* and *B* must be executed on server *N1*, activity *C* on server *N3,* activities *B2* and *B3* are annotated to run on *N2.* Activity *B1* is not annotated at all, in this case it will run on the same server as its parent *B*: *N1*.

The orchestration model is independent from its annotations, that is, the same orchestration may be associated with different annotations, allowing different execution topologies of the same application. This property permits an easy adaptation of an application to a particular infrastructure, and different execution characteristics (security, efficiency, scalability and so on). For example, if a large sub-set of an application is to be executed in the premises of a subcontractor, it may be convenient to delegate that part to a choreography server running inside that sub-contractor’s local network.

In this way, application designers only have to deal with business concerns and define the application as a traditional orchestration model. Administrators can then annotate the orchestration model with information about its logical distribution on several choreography servers on which the application has to be executed.

3.2 Logical Model Transformation

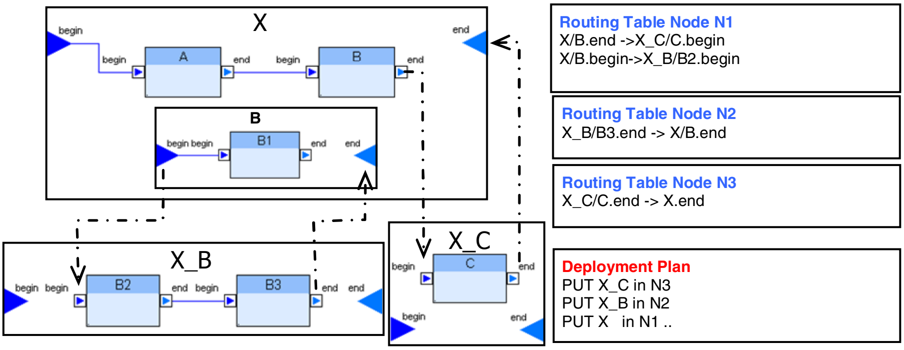
The transformation from an orchestration model and distribution annotations to a distributed orchestration is performed in three steps:

- The orchestration model is transformed in a set of sub-models, each one representing a fragment of the orchestration that has to be executed by a different choreography server.
- Logical routing information is generated, used by choreography controllers (routing and communication mechanisms) for dynamically connecting the sub-models at execution time.
- Deployment information is generated, used by the environment for deploying the models fragments and choreography routing information on the correct choreography servers, and to start the application.

For example, using the orchestration model shown above, a site model consisting on three choreography servers (called *N1*, *N2* and *N3)*, and the annotations shown in Fig. 1, FOCAS generates the following information (Fig. 2):

- Three orchestration models, called *X*, *X_B* and *X_C*,
- A deployment plan, indicating on which server to run each sub-model,
- A routing table for each site running a choreography server.

The algorithm for computing the sub models from the global orchestration and its annotations is as follows. We start from the top level activity X. For each one of its sub-activities two cases are possible: it will be executed in the same node or in a different node. If executed in the same node, it is not modified, that is, it remains defined in the context of its parent activity (*A* and *B* activities in the example). If the sub-activity will be executed in a different node (*C* in the example), an artificial parent activity is created for it (*X_C*) to give an execution context to all sub-activities that will be performed in this node. The same process is repeated on each composite sub-activity. Dataflows between activities spread to different servers become choreography communication links and are indicated in the routing table of the site origin of the dataflow.



Routing Table Node N1
X/B.end ->X_C/C.begin
X/B.begin->X_B/B2.begin

Routing Table Node N2
X_B/B3.end -> X/B.end

Routing Table Node N3
X_C/C.end -> X.end

Deployment Plan
PUT X_C in N3
PUT X_B in N2
PUT X in N1 ..

Fig. 2. Distributed orchestration models

For example in the *N1* routing table, the first line *X/B.end -> X_C/C.begin* states that when activity *B* of process *X* (executing on server node *N1*) reaches its output port called *end*, the data found in that port is to be transferred to node *N3*, and set in port *begin* of activity *C* of *X_C*. This choreography data flow is symbolized by the doted line on the left part of the figure. For the example, four choreography data flows are generated. The deployment plan simply states that activity *X* runs on server *N1, X_B* on *N2*, and *X_C* on *N3*. It is important to see that, at that level, servers are only known by their symbolic name, nothing is said about physical localization and characteristics.

4 Physical Layer: Distributed Orchestration Execution

So far, the models we have presented pertain to the logical layer. However when it comes to execution, these models must be transformed so that the concrete services can be invoked. The FOCAS physical layer is in charge of this process.

4.1 Service Binding

We call *Binding* the mechanism which assigns a service implementation (a real functionality) to an abstract service (a functionality definition). The *Binding* step introduces flexibility because it offers the possibility of selecting, changing and adding new service implementations, at any time, including execution time if required. If a service implementation has been defined independently from the abstract service, it is likely that its interface or technology does not directly fit the abstract service. To benefit from the full reuse potential of service implementations, it is possible to introduce mediators in an adaptation layer, allowing a service implementation to become a valid implementation of an abstract service, even in the presence of syntactic incompatibilities.

SAM (Service Abstract Machine), our binding tool, actually supports services implemented in various technologies, such as: Web Services, OSGi, EJB and Java. It also supports an invocation mechanism that hides the physical location of service instances. In this way, an instance in one node can be invoked by a client located in another node as a local instance. SAM machines are identified by a logical name and use a discovery mechanism to find each other (peer-to-peer infrastructure). SAM has also been designed as a deployment machine which provides an API that allows moving dynamically service implementations (deployment-units to be precise), as well as meta-data (resources) between two SAM machines,.

4.2 Choreography Servers

A choreography server is present in each node used for executing an application as a decentralized orchestration. It is composed of a "traditional" orchestration engine extended by choreography controllers in charge of routing mechanisms.

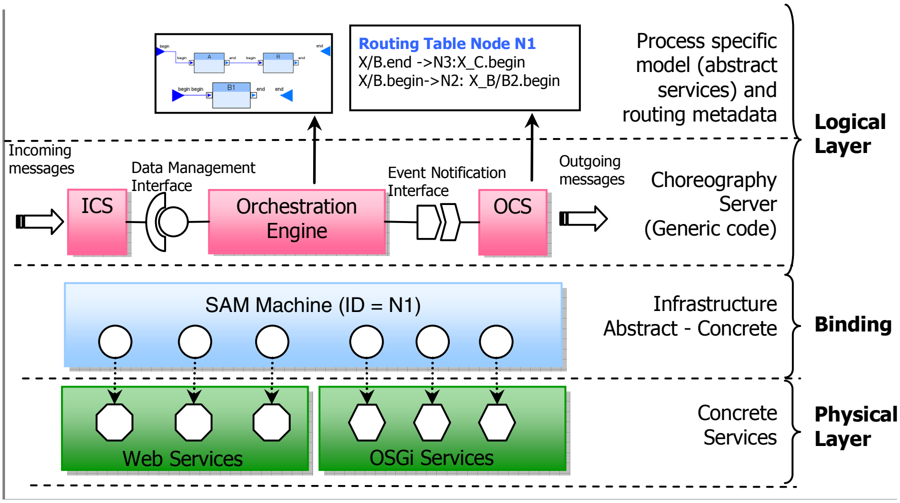


Fig. 3. Choreography Server Architecture in N1

The orchestration engine is unmodified; it interprets an orchestration model in a classic way and provides an API allowing to set information in its ports and to start a process instance. The engine sends messages on relevant events, like when a port is full and ready to send its data along the dataflows. Based on these two standard interactions with the orchestration engine, we have developed the choreography controllers.

The Output Choreography Server (OCS) receives event notifications about full ports coming from the orchestration engine. For each event received, it checks in its routing table if a choreography dataflow is associated with the event. If true, it builds a message containing the relevant information (also found in the routing table), and calls the ICS of the destination choreography server along with the process instance identifier (in order to differentiate several instances of the same process), activity and port in which the information must be set.

The Input Choreography Server (ICS) receives messages coming from an OCS, and simply performs the action requested, i.e., it sets the information in the right instance-activity-port, using the engine API.

It is important to see that the three components of a choreography server are totally generic. In other words, they do not depend on the current orchestration or choreography underway. Once installed, a choreography server can simultaneously execute as many process instances as needed, pertaining to the same or different applications. The knowledge available in a choreography server is limited to the local orchestration fragment and the routing table. It minimizes the amount of data to be transferred and, because it ignores the global process, it suffices to change the routing table dynamically in order to change the application topology at run time. This is particularly important if load balancing, network failure, and scalability are important issues.

Not discussed in this paper, a Choreography Administration Server (CAS) is associated to each choreography server. The CAS interprets a configuration file that indicates what to monitor, where to send monitoring information, where to send information about exceptions and failures, and has an API that allows administration and (re)configuration of the server, which is useful in cases such as failure recovery.

4.3 Deployment

The deployment of an application using our approach is achieved using the deployment plan produced in the transformation phase, and a model of the physical infrastructure.

First, the deployment agent checks the presence of a choreography server on each node on which the application will be deployed. If it is not the case, the deployment agent installs a choreography server (the orchestration engine, ICS and OCS). Being generic, the choreography server is packaged only once and deployed in the same manner everywhere. We only need a SAM machine in each node in order to provide its physical state model (SAM uses runtime models [16]), and the mechanisms (API) required by the deployment agent to install executable code on the platform.

The application can then be deployed. This means that each choreography server must receive its specific sub-process (in the form of an xml file) and its routing table (another xml file). To do so, the deployment agent uses the generated deployment plan and the physical network topology provided by each SAM machine. Each logical node is associated with a physical location. The deployment agent sends the relevant information to the ICS of the correct SAM machines. The ICS uses the engine API to install the sub-process and the OCS API to merge the routing information. Finally, the sub-processes are started, the root one being started last.

Being fully automated, application deployment is not (explicitly) modeled. Similarly, service deployment is not explicitly addressed; each SAM machine is supposed to be able to find and call the relevant services, either because they are web-services, or because the local service has been previously deployed. In the case of a problem, an exception is reported, and convenient reaction is expected from the recovery service. Exception management is currently being researched.

4.4 Distributed Orchestration Architecture

Fig. 4 presents an overall architecture of applications using the distributed orchestration execution. Each node contains a SAM and an orchestration server. The ICS and OCS are themselves SAM compliant services. This characteristic permits the OCS to "discover" the remote ICS instances it requires in order to send information from one node to another. Because SAM dynamically creates proxies to services being executed in another machine of the peer-to-peer infrastructure, an OCS communicates with each ICS using simple method invocation. SAM hides the underlying communication protocol, which can be RPC, a MOM, a SOAP-based communication infrastructure, or even a mixed approach depending on the specific network.

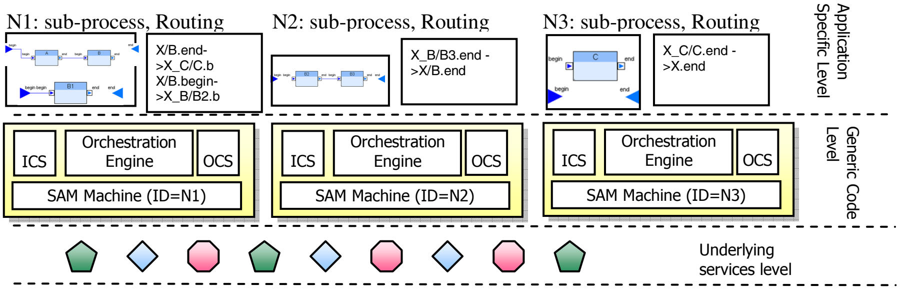


Fig. 4. Distributed Platform Architecture

5 Related Work

In [17], a workflow execution architecture for pervasive applications is presented. It deals with problems as distributed control and assignment of some parts of workflows to be executed by devices. The architecture considers the implementation of a protocol for controlling the communication of different devices participating in the execution. The protocol is heavyweight, in each communication the complete plan of execution (workflow model) is passed between the nodes. In addition, all relevant data is also transferred between devices. Computing the sub-model to be executed in a device is performed in each interaction degrading the performance of the overall system during execution. In comparison, our architecture uses a generic and lightweight protocol; only the relevant data is transferred between nodes and sub-models computation is performed at deployment time to get optimal performance.

The SELF-SERV system [18] also proposes a peer-to-peer execution of a services orchestration. The system proposes an IDE for developing compositions using states charts as description formalism and a runtime for supporting the execution. The deployer in SELF-SERV does not explicitly specify how the orchestration is distributed between the nodes executing the orchestration. Instead, an algorithm based on physical node information and services distribution (service selected at design time) is used to compute it. In our approach the orchestration model is explicitly annotated with logical node information, and then at deployment time this information is used to choose the physical nodes for execution. In addition, physical distribution of services is hidden by our execution runtime; this property permits selection of services at runtime (and even service replacement). Additionally, our system also permits dynamically changing the distribution information.

In [19] is presented a decentralized execution of an orchestration expressed in BPEL4WS. In a similar way as in our approach, the responsibility of executing the composition is shared by a set of controllers distributed in a network, and it uses a set of communication mechanisms to ensure message distribution (SOAP/HTTP, SOAP/JMS or JMS). An interesting idea is the use of a monitoring node to receive notifications from execution nodes in order to handle error propagation and application recovery. However, this approach focuses only on system performance. Suppositions about service location are done at the generation phase. In our approach, distribution criteria is up to the administrator (but an automatic technique can be used instead), and tools for definition, deployment and mechanisms of selection are provided by our framework.

6 Conclusion

The construction of applications using the service oriented computing paradigm is increasingly popular. SOC provides important characteristics like services independence and late binding, which allow flexible construction of applications by assembling services. However, these properties also make difficult the expression and execution of service compositions. In order to solve these problems, two paradigms have been used in SOC: orchestration, which expresses the interaction from a central point of view, and choreography, which expresses applications as a set messages exchanged between services, each one having its advantages and limitations.

Our paper describes an approach which borrows the facility of expression of the orchestration approach with the scalability and performance offered by choreography, since execution is fully distributed. The FOCAS framework supports the creation of this kind of application, proposing an extensible Model-Driven approach around a workflow domain. FOCAS divides applications into two levels of abstraction; the *logical level* in which developers express the business model, without regard of the technical details of the underlying services or platforms; and the *physical level,* in charge of performing the execution on the actual platform, using the actual services.

FOCAS has to deal with a number of challenges: composition of functional concerns (workflow, services, data); composition of non-functional concerns (security, distribution, transactions); transformation of the artifacts produced in the logical level into artifacts needed for execution in the physical level; hiding the heterogeneity of the underlying platforms at execution time; and deploying the application on a network of computers. Our experience has shown that addressing all these challenges by hand is virtually impossible. The main goal of FOCAS is to provide an environment and tools that "ease" the development and execution of demanding service-based applications. To a large extent, this goal is reached.

FOCAS has been in use for the last two years. Nevertheless, many challenging extensions are still to be clarified, like monitoring policies, strategies for error handling and dynamic reconfiguration at execution. Other fundamental issues are still to be addressed, like interaction between different concerns (e.g. security and distribution), or domain composition semantic interference.

Our approach fills the gap between traditional orchestration (fully centralized) and choreography (fully distributed), providing the administrator easy means to select the right compromise, not only at deployment time, but also at execution time. More generally our work shows how any process model, in any domain (other than service-based applications), can be transparently executed on a network of computers, still reusing the original process interpreter. It allows the same process model to be executed in different contexts, on different networks, and with different execution characteristics by simply changing annotations, even during execution. We believe this constitutes an important improvement with regard to traditional approaches.

References

1. Alonso, G., Casati, F., Kuno, H., Machiraju, H.: Web Services - Concepts, Architectures and Applications. Springer, Heidelberg (2003)
2. Papazoglou, M., van den Heuvel, W.: Service oriented architectures: approaches, technologies and research issues. The VLDB Journal 16(3), 389–415 (2007)
3. Peltz, C.: Web services orchestration and choreography. Computer 36(10), 46–52 (2003)
4. Cubera, F.e.a.: Web Services Business Process Execution Language. Specification (April 2007), http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf
5. Weske, M.: Business Process Management: Concepts, Languages, Architectures. Springer, Heidelberg (2007)
6. W3C: Web Services Choreography Interface (WSCI). Specification (August 2002), http://www.w3.org/TR/wsci/
7. W3C: Web services choreography description language version (November 2005), http://www.w3.org/TR/2004/WD-ws-cdl-10-20040427/
8. Estublier, J., Ionita, A., Nguyen, T.: Code generation for a bi-dimensional composition mechanism. In: Central and East European Conference on Software Engineering Techniques (2008)
9. Pedraza, G., Dieng, I., Estublier, J.: Multi-concerns composition for a process support framework. In: Proceedings of the ECMDA Workshop on Model Driven Tool and Process Integration, FOKUS, Berlin (June 2008)
10. Chollet, S., Lalanda, P.: Security specification at process level. In: IEEE International Conference on Services Computing (SCC 2008) (July 2008)

11. Koehler, J., Hauser, R., Sendall, S., Wahler, M.: Declarative techniques for model-driven business process integration. IBM Systems Journal 44(1), 47–65 (2005)
12. Nitzsche, J., van Lessen, T., Karastoyanova, D., Leymann, F.: Bpellight. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 214–229. Springer, Heidelberg (2007)
13. Charfi, A., Mezini, M.: Hybrid web service composition: business processes meet business rules. In: ICSOC 2004: Proceedings of the 2nd international conference on Service oriented computing, pp. 30–38. ACM Press, New York (2004)
14. Pedraza, G., Estublier, J.: An extensible services orchestration framework through concern composition. In: Proceedings of International Workshop on Non-functional System Properties in Domain Specific Modeling Languages, Toulouse (September 2008)
15. Estublier, J., Dami, S., Amiour, M.: Apel: A graphical yet executable formalism for process modeling. Automated Software Engineering: An International Journal 5(1), 61–96 (1998)
16. France, R., Rumpe, B.: Model-driven development of complex software: A research roadmap. In: Future of Software Engineering, 2007. FOSE 2007, pp. 37–54 (May 2007)
17. Montagut, F., Molva, R.: Enabling pervasive execution of workflows. In: CollaborateCom 2005, 1st IEEE International Conference on Collaborative Computing:Networking, Applications and Worksharing, p. 10. IEEE Computer Society Press, Los Alamitos (2005)
18. Benatallah, B., Dumas, M., Sheng, Q.Z.: Facilitating the rapid development and scalable orchestration of composite web services. Distributed and Parallel Databases 17(1), 5–37 (2005)
19. Chafle, G., Chandra, S., Mann, V., Nanda, M.: Decentralized orchestration of composite web services. In: WWW Alt 2004: Proceedings of the 13th international World Wide Web conference, pp. 134–143. ACM, New York (2004)

An Architecture for Modeling and Applying Quality Processes on Evolving Software

Fadrian Sudaman[1], Christine Mingins[1], and Martin Dick[2]

[1] Faculty of IT, Monash University, Australia
fadrian.sudaman@infotech.monash.edu.au,
christine.mingins@infotech.monash.edu.au
[2] School of Business IT, RMIT University, Australia
martin.dick@rmit.edu.au

Abstract. Software process and product views should be closely linked in order to better manage quality improvement. However until now the two views have not been effectively synchronized. Current approaches to Software Configuration Management (SCM) are strongly based on files and lacking in logical and semantic understanding. Some impediments faced when modeling and analyzing software evolution include additional effort for dealing with language dependent source code analysis and continuous mining of the evolving system. By leveraging features offered by modern VMs and other enabling technologies, we have developed a language neutral architecture with extensibility mechanisms to support continuous Software Evolution Management (SEM). Our research aims to contribute to an SEM infrastructure where semantic artifacts can be consistently accessed, tracked and managed for performing software evolution analytics beyond the file-based model. This paper presents compelling factors for our infrastructure, the architecture we have developed, and then sketches a case study to demonstrate its application.

1 Introduction

SCM has been used for decades to manage changes in complex systems over time[6]. It provides a wealth of evolutionary history and change reasoning about software systems and offers a wide range of tools for managing and controlling the development and evolution of software systems [15]. The file based and platform independent concepts of SCM make it simple, easy to use and give it the flexibility to handle all kinds of software artifacts. However, many weaknesses also emerge from the same concepts. Too little knowledge of the underlying software product affects the effectiveness of the SCM in modeling and analyzing the product evolution [17]. Evolutionary information, versioning and differentiation are limited to file units without considering the deeper meaning of the target artifacts [6][16]. This also does not match up with the logical abstractions (such as in the object-oriented context) that software practitioners are familiar with [16], nor with commonly used software architecture or design models such as UML [17]. The disconnection or isolation of SCM from product knowledge and other development environments force a one-way connection to the SCM tool. Consequently, SCM cannot leverage the advances made in other domains to deliver a more targeted service to the product it manages [6].

Q. Wang et al. (Eds.): ICSP 2009, LNCS 5543, pp. 87–99, 2009.

Modern virtual machines (VM) and new development platforms such as the Java VM and the Common Language Infrastructure (CLI[1]) provide language independent logical models and rich metadata support beyond primitive text and binary files. Semantic gaps between source code and the developer's intention can be bridged through the use of custom metadata for defining additional semantic attributes. Leveraging on these, we can refine the current state of SCM tools to better support SEM by bringing together the software product, semantic artifacts and quality process. This research aims to address some of the weaknesses of SCM for managing software semantic evolution by contributing to an infrastructure where semantic artifacts can be continuously interpreted and managed within a SEM context and made accessible at all times for performing software evolution analytics. We refer to 'semantic artifact' in this paper as the logical structure of the software in terms of the object-oriented programming model (such as class, field and inheritance) and any other artifacts of interest to the actor-user that can be modeled and extrapolated from the underlying model. These artifacts can be defined explicitly via declarative metadata, or may relate to implicitly available metadata such as product information.

2 Trends in SEM

Software systems continue to evolve overtime due to process improvement and changing business requirements. The significance of software evolution has been widely recognized for the last three decades, pioneered by the work of Lehman [9]. According to the laws of software evolution, software structure and quality tends to degrade over time and make the task of future evolution even more difficult. Empirical studies have proven that code integrity progressively deteriorate during the course of modification [9]. Eiks et al. refer to this phenomenon as 'code decay' [13].

SEM focuses on extracting and analyzing evolutionary information about software systems with the aim of better understanding the software changes over time. The effective use of SEM enables the building of predictive and retrospective models [9]. These models can be used by developers or managers in many areas such as analyzing the software processes, change reasoning, risk management, project estimation and evolution patterns. Some major challenges that affect the effectiveness of SEM include the availability of empirical information and coping with huge amounts of data that can be complex and unstructured or not consistently defined [8]. These problems intensify as the size and complexity of the system increases over a lengthy evolution period. Some of the most successful approaches for managing evolutionary information involve the use of SCM, software metrics and visualizations [8][15].

Today, SCM systems are widely accepted to be essential to any real-world software project as a means of obtaining team productivity, quality processes and products. It is also important for software projects to demonstrate conformance to the CMMI Level 2, which requires configuration management processes to be repeatable. Related work in mainstream SCM development including IDE integration, decentralized repositories, change set management and web engineering are still focusing strictly on file-based models and neglect the software module abstractions [6][16].

[1] CLI is an ISO Standard ISO/IEC 23271. Ms. NET CLR and Mono implement this standard.

Although recent research projects in SCM offer richer modeling capabilities, their approaches often rely on source code parsing or require additional effort to extract and map between the source artifacts and the repositories (see section 6). These reveal the needs for a refined SEM architecture that offers well defined infrastructure with rich functionality and yet retains sufficient simplicity.

3 Evolution Analysis beyond Source Code

In the context of SEM, static software analysis is used extensively to gather metrics ranging from simple counting of LOC to complex coupling between objects for studying software evolution [3][17]. This section explores some of the motivational factors that make our project compelling as the architecture for continuous SEM.

3.1 Language Neutrality

Modern VMs provide language independent instruction set commonly referred to as bytecode. It includes a full type system that enables program structure and semantic attributes to be preserved as OO model with rich metadata that can be extracted through reflection [3]. Although the richness of bytecode may be compromised when mapping from high level language, reverse engineering from bytecode using decompilers (e.g. JODE and Lutz's Reflector) demonstrates minimal loss of program semantics. Today, at least 20-30 languages target the .NET platform, and similarly, many languages such as Java, Groovy, Jython and JRuby are targeting the Java platform. Comparing to bytecode analysis, source code analysis will require complex parsers similar to the language compiler for each language to be developed [3]. This is quite an alarming effort to support the wide variety of languages supported by modern VMs in order to achieve the same broader impact as targeting the bytecode directly.

3.2 Systematic Monitoring

The conventional approach of manually taking system snapshots and deriving comparison metrics has some drawbacks. The lack of integration and sharing of extracted information forces the need to reapply the metric analysis for each type of metric computation performed. The availability of past snapshots becomes the responsibility of individuals performing the analysis and often the data is limited by the period captured within the snapshot. Gathering metrics with ad-hoc static analysis may lack consistency and can be resource consuming [5]. Our approach is continuous and systematic whereby the build process is automated to produce bytecode snapshots of the system regularly. Analysis is then performed to identify and automatically store new and changed artifacts to the repository. These artifacts are available continuously for applying metrics and further evolution analytics. This approach offers a foundation that is a consistent, uniform and reliable for continuous SEM.

3.3 Logical Abstractions

Capturing software evolution at the structural level aligns with common practices such as typical OO methodologies and is crucial for correlating and narrowing the design-implementation gap and enabling easier detection of architecture drift and software erosion issues [13][17]. Despite all the strong arguments for needing to use a logical data model in SCM, we cannot ignore the fact that most widely used SCM (such as CVS and Subversion) and IDEs (such as Visual Studio and Eclipse) use a file-based model. Therefore, our approach is not to replace existing file-based SCM; instead we enrich the model with a parallel logical data model to provide richer views to suit users in different roles. We retain the use of a file-based SCM for controlling and recording software evolution as it has been done all along, in parallel with a logical data model to represent program structure and custom metadata captured in a relational database for modeling and analyzing software evolution. The two repositories exist in parallel and interconnect to form a virtual repository that is capable of providing deeper insights and alternative views of the system it manages.

3.4 Custom Metadata

The support for custom metadata enables extensions to the underlying type system to seamlessly incorporate user-defined semantic artifacts representing new, project-oriented perspectives into software analysis. In the CLI, developers can add custom metadata using syntax like [UnitTest] or @UnitTest for Java developers. We recognize that it may appear challenging to realize these benefits as it relies on a standardized set of custom metadata to be defined for easy interpretation, and developers' diligence to annotate the source code. However in practice, custom metadata is already being defined, used and often enforced extensively as part of framework libraries such as the .NET framework, Web Services and NUnit; hence we can take advantage of it immediately without imposing any additional effort. Developers can choose to add their own custom metadata, or use simple toolset such as code generation, editor plugin or even through an automated code injection to embed additional metadata. Our approach offers native support for understanding custom metadata, but without imposing the obligation of defining it or penalties for not having it.

3.5 Preserving Quality Attributes and Conceptual Integrity

Typically, major architectural decisions and design heuristics about the software system are decided prior to implementation stage. As the system becomes more complex, a lack of knowledge and understanding may lead to a poor implementation that deviates from the design intention hence causing software erosion []10[13]. To address this, our approach introduces an infrastructure for applying quality process whereby a policy can be defined and applied on any semantic artifact such as a method, a class, or entire class library and trigger add-in code modules (plugins) for verification and notification when violation is encountered as the artifact evolves. For example, project decisions have identified several security related methods (e.g. method A and B) in the system as being a sensitive code area and any modifications must trigger an email alert for code review. A policy can be easily defined to automatically identify such modifications and act on it accordingly as the system evolved. Other quality attributes about the system such as complexity, reuse, stability or volatility of a class or package can also be regulated using the same process. This places quality monitoring process into the development process to facilitate the preservation of the system integrity and quality systematically evolving system.

4 OSSEM Architecture

To demonstrate our research ideas presented in section 0 above and to strengthen our research impact, OSSEM (Object-oriented Software Semantic Evolution Management) has been developed. OSSEM offers an infrastructure for modeling, guiding and managing software evolution while retaining the entire functionality provided by SVN for recording file-based software evolution.

4.1 High Level Architecture

OSSEM provides automatic monitoring and systematic processing of information about evolving software artifacts from the source code stored in the SCM into a well-defined data model and stores them into a relational database. This information is then accessible at any time for applying analysis, metrics and visualization.

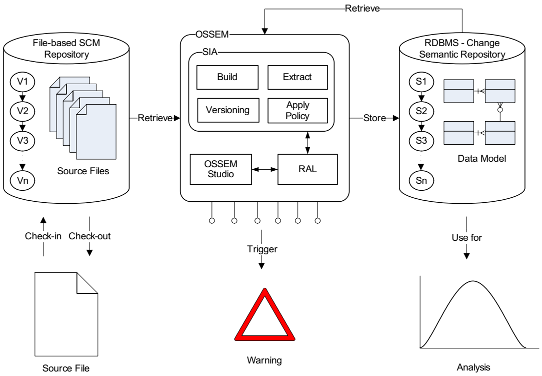


Fig. 1. OSSEM Overview

The architecture of OSSEM consists of three core modules: Semantic Instrumentation Agent (SIA), Repository Access Library (RAL) and OSSEM Studio. SIA is responsible for monitoring the file based repository to retrieve and build the source code into bytecode necessary for extracting semantic artifacts of the evolving system. A snapshot of the system is captured in the database each time the SIA process is performed. The RAL acts as the data access layer between SIA and the file based repository, and for reading and writing data to the change semantic repository (CSR). RAL also provides higher level repository access functionality to retrieve evolutionary information on captured and derived artifacts. OSSEM Studio is the client application providing a UI for configuring the OSSEM infrastructure, querying captured evolutionary information of the system in the form of diff, metrics, diagrams and graphs (more details in section 4.3). The OSSEM architecture also encompasses the facility to apply a policy checking process. Any changes on semantic artifacts with preconfigured policies will activate the policy verification process, which may then trigger violation notifications. Policy is configured through OSSEM Studio and persisted through RAL and applied by SIA during execution.

Figure 1 shows the OSSEM high level information flow where V1, V2, V3...Vn denote the evolving project versioning in the SCM repository and S1, S2, S3...Sn denote the evolving project snapshots captured.

4.2 Captured Artifacts

Each system evolving in OSSEM has a history that captures one or more project versions. The logical program structure (such as Namespace, Class and Method) is modeled as entities. Each entity has a corresponding versioning entity. Each entity may have declarative attributes and their associated fields attached to it. Each version entity points to its previous version for easy navigation. This forms the entity version history. Figure 2 shows the high level logical model of our semantic artifact versioning data store.

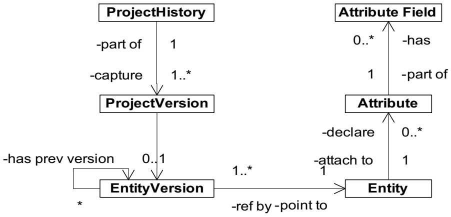


Fig. 2. Semantic Artifacts Versioning Meta-model

A new version of an entity is created whenever a change to the entity is detected. In very simple terms, the change versioning detection algorithm works by comparing each current entity definition against its last active version in terms of its signature, body and compositions. Any difference detected is considered as a change that triggers the creation of a new version for this entity. This may also ripple up and cause a new version to be created for its parent entity. To give a simple illustration: if class C1 has only one method M1 and only the implementation of M1 changes, a new version will be created for M1 and the current version of C1 will now refer to the new version of M1. If new method M2 is added to class C1 and nothing else, a new version of C1 will be created to consist of the latest version of M1 and M2, whilst the original version of C1 remains unchanged and consists of latest version of M1 only. Even though each entity can evolve independently, OSSEM always has frozen snapshots of all the entity versions for an evolving system at a particular time hence analysis and metrics can be done accurately for each snapshot in isolation.

From a more concrete perspective, we can articulate semantic artifacts in OSSEM as all the artifacts are captured in the form of evolutionary versions; relationship between entities such as inheritance and composition; and entity definitions such as name, type and the bytecode of the evolving system in a well defined relational database. Thus we can access artifacts from the top level down: a project consists of number of assemblies, each assembly may define a number of namespaces, all the way down to a method of a class that has a number of parameters, declares a number of local variables and defines a number of method calls. Custom artifacts defined using declarative attributes on entities are also captured and versioned and are accessible at all times across different snapshots. The full object model to access the semantic artifacts is also made available by OSSEM through its RAL module. The RAL module simplifies programmatic access to these artifacts in a consistent manner without having to use SQL queries. It also exposes additional semantic artifacts derived by aggregating or applying intelligence on existing artifacts so that basic metrics can be easily acquired. Examples of these metrics include number of operations, number of instructions, method complexity, weighted methods per class (WMC), coupling efferent (Ce), coupling efferent (Ca), relational cohesion ratio (H) and instability ratio (I) [12]. Because custom defined artifacts are automatically tracked across versions by OSSEM, they are also readily available to derive richer or bespoke metrics such as number of unit test classes, number of web services methods on the evolving system.

4.3 Implementation

This section discusses the technical implementation and the enabling technology of the three core OSSEM modules.

4.3.1 Semantic Instrumentation Agent (SIA)

The SIA extracts, interprets, collects and stores semantic information about changes made to the system controlled by the SCM. Figure 6 depicts an overview of the systematic process followed by the SIA. Basically, this agent performs four tasks in sequential order: continuous build, software semantic analysis, policy verification and change semantics persistence of the evolving software. The current implementation of SIA targets CLI[1] and deals with bytecode in the form of CIL. In developing the module for performing continuous builds, OSSEM drew on proven concepts and tools: CruiseControl and Ant as its underlying technologies. The design of the software change semantic analysis and policy checking modules is very much inspired by the work of Microsoft FxCop which allows for rich customization and extensibility.

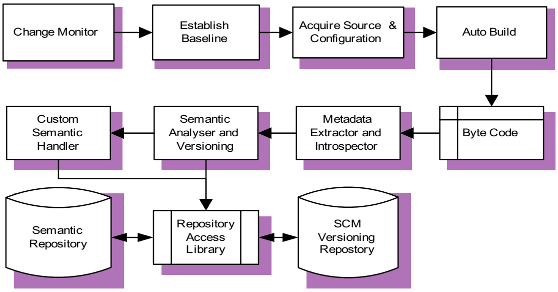

Change Monitor
Establish Baseline
Acquire Source & Configuration
Auto Build
Custom Semantic Handler
Semantic Analyser and Versioning
Metadata Extractor and Introspector
Byte Code
Semantic Repository
Repository Access Library
SCM Versioning Repostory

Fig. 3. Semantic Instrumentation Agent Process

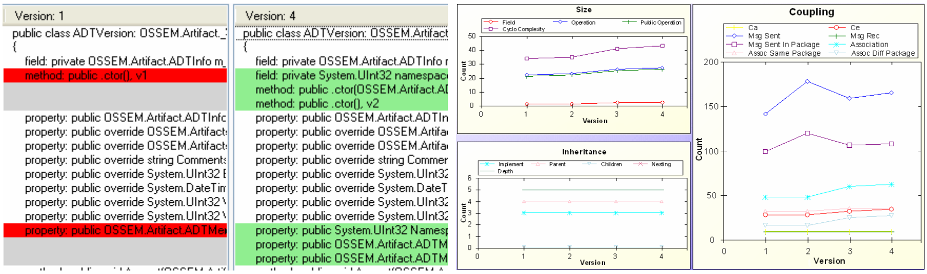
SIA works like an integrated build server and is capable of running on a machine co-located with the SVN server or on a separate machine. The instrumentation process is triggered by an automated process that continuously monitors the SVN repository. Ideally, whenever a new change set is committed, the SIA will immediately perform the processes described above. In practice, such an approach may cause a large build queue in an active or large development environment and put high pressure on system resources. To solve this problem, the implementation of the SIA adopts the periodic monitoring technique whereby the SVN repository is continually monitored for committed changes at defined intervals. Once a change is detected, the agent will determine if the last committed change set is longer ago than a user defined period of *x* minutes. This determines if the changes are considered stable for SIA to launch the auto build process. Each project managed by OSSEM must have an accompanying build configuration file and settings, which are supplied through OSSEM Studio (see 4.3.3). When analyzing the semantic artifacts, SIA will make default interpretations of known declarative metadata, and apply the policy checking mechanism. If a policy is configured for a semantic artifact, appropriate policy verification will be activated. Policy verification modules are implemented as plugins to the SIA by conforming to a predefined interface contract. The extracted semantic artifacts will be compared and analyzed against the latest version of the artifacts in the repository using a predefined algorithm. Detected differences will trigger new versions of the entities to be created and stored in the repository guided by the OSSEM versioning policy.

4.3.2 Repository Access Library (RAL)

At the core of the OSSEM implementation is the RAL component. The RAL serves as the entry point for all data access to the OSSEM data store as one virtual repository that is made up of a physical SVN file versioning repository and a change semantics repository (CSR) that contains the associated bytecode and extracted metadata of the systems managed by OSSEM. All semantic evolutionary information gathered by OSSEM is stored in the CSR implemented as a MySQL relational database. Data stored in the CSR can be retrieved using RAL programming interfaces or direct query to the database using SQL. Both the SIA and RAL integrate with the SVN repository through the client library API and the hook extension mechanism provided by SVN. This separation allows a high level modularization of OSSEM and SVN hence allowing OSSEM to evolve independently from SVN and target other SCM tools in the future. RAL exposes a rich set of interfaces necessary for client applications to access and manage all configuration settings, the evolutionary information captured and all other aggregated evolutionary information exposed by OSSEM. Given the structured data model and richness of the semantic artifacts provided by the OSSEM, static software evolution analysis such as visualizing an entity at a given point in time; ‘diff’-ing versions of a method, class or namespace; and gathering software size, design and growth metrics are relatively straightforward.

4.3.3 OSSEM Studio

The OSSEM Studio is a rich client application built on top of RAL to demonstrate the richness, capability and usability of OSSEM in collecting, storing and retrieving semantic artifacts of interest. Basic functionality includes browsing the evolution tree of the semantic artifacts, performing a diff of two different snapshots of a semantic artifact and drilling down into its children and displaying various static software analysis metrics in various formats. OSSEM Studio also provides an interface for configuring OSSEM projects, evolution policies on specific semantic artifacts and adding custom metadata for a specific artifact such as [OSSEMLabel (“Stage1”)], which in many instances is more suitable than embedding the metadata in the source code. Because all logic intelligence is centralized in the RAL, it is possible to build different types of client applications with similar capabilities to OSSEM Studio with minimal effort. IDE plugins to Visual Studio or Eclipse can also be built with ease to integrate OSSEM to development environments in the future by taking advantage of the rich interface offered by the RAL.



```
Version: 1
public class ADTVersion: OSSEM.Artifact._:
{
    field: private OSSEM.Artifact.ADTInfo m_
    method: public .ctor(), v1

    property: public OSSEM.Artifact.ADTInfc
    property: public override OSSEM.Artifact:
    property: public override OSSEM.Artifact:
    property: public override string Comments
    property: public override System.UInt32 E
    property: public override System.DateTim
    property: public override System.UInt32 \
    property: public override System.UInt32 \
    property: public OSSEM.Artifact.ADTMe
```

```
Version: 4
public class ADTVersion: OSSEM.Artifact
{
    field: private OSSEM.Artifact.ADTInfo r
    field: private System.UInt32 namespac
    method: public .ctor(OSSEM.Artifact.AI
    method: public .ctor(), v2
    property: public OSSEM.Artifact.ADTIn
    property: public override OSSEM.Artifa
    property: public override OSSEM.Artifa
    property: public override string Commer
    property: public override System.UInt32
    property: public override System.DateT
    property: public override System.UInt32
    property: public override System.UInt32
    property: public System.UInt32 Namesp
    property: public OSSEM.Artifact.ADTM
    property: public OSSEM.Artifact.ADTM
```

Fig. 4. 'Diff'-ing and Design metrics over versions

To showcase this, we have built several visualization functionalities into OSSEM Studio. Figure 3 shows the 'version compare' feature of a class entity in OSSEM Studio. To implement this 'diff'-ing feature, OSSEM Studio uses the RAL module to retrieve the class definition for Version 1 and 4 of the inspected entity. It then uses the provided EntityTextVisualizer class by RAL to produce the textual stream for both versions of the entities and feed them to the standard text diff control to show the differences graphically. To implement the design metric graph, OSSEM Studio uses the RAL module to retrieve the class definition for all the versions of the inspected entity. It then uses the provided ClassDesignMetrics class in RAL to automatically calculate and produce relevant metric values use for plotting the graphs.

4.4 Storage and Performance

It goes without saying that OSSEM needs to be built with performance and storage optimization in mind in order to be practical for use in a real life evolving system. Some of the implementation challenges we faced include: dealing with very large datasets, resource usage for processing, high database hit rates and timely execution. To overcome these challenges, to date we have adopted strategies such as using sandboxes to effectively load and unload data, parallel processing, smart pre-fetching, compression and caching of data and batching of database operations.

Table 1 shows the physical bytecode size, CSR storage size and execution time taken by SIA for the first time to collect the semantic artifacts for three different sized projects. OSSEM takes approximately 5 times the storage space over and above the physical bytecode size and is capable of processing approximately 400 KB of bytecode per minute. Subsequent execution of SIA on the project will perform the incremental change analysis and versioning. Our experiments show that subsequent execution of SIA on projects with < 10% changes on the artifacts, takes approximately 25% of the base execution time (about 0.44 minutes for OSSEM SIA and 32.8 minutes for Ms .NET Framework) to execute with the storage space usage growing consistently to the above statistics (e.g. 5 methods with total of 1 KB bytecode changed will yield approximately 5 KB of storage in OSSEM repository) to capture

Table 1. Project storage and execution statistics

	OSSEM SIA	Ms. WCF	Ms. .NET 2.0
Bytecode Size	0.62 MB	7.46 MB	44.5 MB
Storage Size	4.19 MB	41.14 MB	196.31 MB
Execution Time	1.77 mins	21.66 mins	131.2 mins
ADT Count	484	4963	23676

the incremental changes. Because OSSEM only stores the changed artifacts, our experiments of applying SIA on the rapidly evolving OSSEM project itself repeatedly shows that the storage usage and growth are manageable. All tests above were performed on a Pentium D 3GHz machine with 2GB of RAM running Windows XP operating system.

Putting the bytecode size into perspective, at the time of writing this paper, the compiled size of OSSEM SIA is 616 KB, a result of compiling 284 files that contain 484 ADTs with approximately 79000 SLOC. We categorized this as a small size project and the Ms.NET framework with approximately 23676 ADTs as a large and very complex project. We categorized the Ms. Workflow Communication Foundation (WCF) framework as a typical medium to large size project. The number of classes (NCL) size metrics of WCF is very similar to the Mozilla open source project (approximately 5000 NCL) and hence we have decided to use this framework as the performance and storage benchmark test for OSSEM in the future.

5 Illustrative Scenario

The software company, ACE, has a new project that runs over 18 months with stage 1 involving redevelopment of a legacy system and all the new features to be delivered in stage 2. The client expects ACE to deliver stage 1 in 12 months and thereafter any changes to the sign off code-base needs to be documented. During the course of the project, ACE is required to report regularly. OSSEM is configured for this project.

Taking a simple size heuristic, Number of Class (NCL), suppose the project has 1000 classes at a specific time T5, 150 were code generated, 200 were library and 50 were unit test classes, hence only 500 were the actual domain classes. Changes and growth of these domain classes over time will be of more interest as opposed to the code generated classes. A crude analysis based on all the classes to derive estimates may not yield a realistic insight. Utilizing semantic artifacts collected by OSSEM, project size and change heuristics can be easily derived by isolating classes, namespaces or even entire assemblies based on their names or other annotations associated with them (e.g. Ms. NET Unit Test Framework requires all unit test classes to have [TestClass] attributes) in the retrieval process to then develop perspectives of the system based on code categories. Similar reports can also be easily derived such as: the ratio of test cases to production code; areas of library code have been subject to redevelopment since last milestone; top ten domain classes in terms of volatility.

From a technical perspective, ACE has identified several project risks: signed-off code may be changed and not reported; the architectural integrity of the system may deteriorate over time. To mitigate these risks, a set of OSSEM policies are developed to guide the project evolution. First, to ensure changes to sign off code are recorded, "S1SignOff" tag is added to the project using OSSEM Studio and applied recursively to all the methods in the project prior to commencing stage 2 development. "SignOffChanged" policy is defined to trigger a plugin module for inspecting all methods changed; if the method has a predecessor version with the "S1SignOff" attribute, an email notification is triggered. This ensures that changes made to signed-off code are always identified and made known to the appropriate person systematically. To mitigate architecture integrity degradation, a number of design guidelines are identified and enforced. Here we choose the loose coupling design heuristic as an example. ACE enforces coupling trends over time by dictating that changes on any methods within a designated area must add no more than three parameters and three non system type local variables. A policy "CouplingMonitor" is defined for the project and the plugin module is implemented to count the number of parameters and local variables for the changed method and subtract it against its previous version. Any method that violates this policy would trigger an alert with details of the method and violated rules listed. Other examples of design heuristic policy that can be applied may include monitoring metrics for a class or package such as weighted method per class, relational cohesion ratio or instability ratio [12] do not fall outside a given range or change more than allowable range when compared against previous version. This can help to ensure the architectural integrity is constantly monitored and preserved as the system evolved.

The use case scenario above illustrates one of the most important contributions of OSSEM: easy yet disciplined extraction of statistical metrics on evolving systems; the ability to define generic project level artifacts based on internal software elements, evolution policies and associated plugins; a reusable policy library that defines a repeatable software evolution process for preserving integrity of the evolving system.

6 Related Work

Many software evolution toolsets and research projects have explored the use of richer and flexible abstraction beyond the file model. NUCM architecture [1] defines a generic distributed repository model that separates CM artifacts storage and the policies. Applying NUCM in OO SEM requires programming and mapping of the policy and logical abstraction to atoms and collections and also faces restriction in modeling composite objects [1][16]. OSSEM shares the core idea of Molhado infrastructure [16] and Stellation system [7] of capturing evolutionary information at the logical abstraction level with well defined data model. However they move away completely from file-based SCM, whilst OSSEM relies on SVN for source code versioning control and itself focus on analysis, modeling and guiding the software evolution. We perceive that divergence from the widely used file-based SCM to be risky and may impede its adoption in the industry. While the meta-model of OSSEM is analogous to HISMO [15], OSSEM focuses on the entire systematic process of modeling, obtaining and capturing the data rather than just modeling as in HISMO (implemented in MOOSE).

Another desirable feature of OSSEM that sets itself apart from other related work SCM system is the policy verification support. The Darwin environment [11], like our approach also focuses on the continuous detection of violations. It uses rules to regulate exchanging of messages between object, which is also possible in our approach (by inspecting called methods). Our work covers many additional aspects such as regulating entity structure, changes, size and growth overtime and ability to draw the comparisons against any historical snapshots. This is in a way similar to the work done by Madhavji and Tasse [10] but different in the fact that OSSEM operation on a much more fine grained level. OSSEM policy can target specific or an aggregation of semantic artifacts (such as a method, an attribute or a namespace) which are accessible natively within its infrastructure. In addition, OSSEM extensibility model based on plugin architecture allows the leveraging of programming language features and flexibility.

Fact extractor type projects such as Bloof [2], Kenyon [5], CollabDev [14] and JDevAn [17] are closest to the OSSEM approach. Although they share many motivations and goals, they differ considerably in their approaches and applications. JDevAn is implemented as plugin to Eclipse, hence the availability of snapshots relies on local code base and the diligence of the developer. Bloof extracts their historical information based on file changes and other commit metadata (such as date, author and comment) and is therefore clearly lacking in support for OO SEM. CollabDev strives to achieve more than just mining evolutionarily information, but also support for document management and knowledge sharing facilities. Recent vendors' products such as Jazz by IBM and Visual Studio Team System (VSTS) by Microsoft are similar to CollabDev, which are focusing on collaborative platforms for managing software development across the lifecycle; nevertheless they have little focus on OO SEM specifically. Although Kenyon offers more flexibility with its custom fact extractors it also still focuses on file and code changes, commit and configuration metadata. There is no evidence to suggest that it supports fact extraction at the logical abstraction level, or modeling and versioning similar to OSSEM.

7 Conclusions and Future Work

Although the quality of software process greatly influences the quality of the software product, it does not guarantee the final outcome. Until now, software process and product views were not effectively synchronized, and often the product quality evaluation is performed separately from the development process at later phase. OSSEM bridges this gap by capturing the product view and allowing quality measures and processes to be applied in parallel with development process, hence quality trends can be continuously monitored and any deviation from design objectives can be detected and rectified in a timely manner. OSSEM brings together the product and process views of the evolving software and facilitate software process improvement. This can be further illustrated within an outsourced development context, where OSSEM can elevate the process of project communication, reduce the risks of diverging visions, offer better project control and support improved relationships by promoting evidence-based reporting and monitoring process [4].

OSSEM offers native support for capturing and retrieving semantic artifacts hence standard object-oriented size and coupling metrics and design heuristics can be easily derived and reported. Its extensibility model along with well defined data access and process allow policy to be developed relatively easily to guide and inform the software evolution. Compared to OSSEM, design heuristic analysis tools such as

FxCop are able to police coupling (e.g. methods should have < 7 parameters) based on absolute heuristics deriving from a single snapshot only while OSSEM is able to draw comparative heuristics from current and historical snapshots. We believe the OSSEM architecture facilitates the application of the CMMI Level 2 (Repeatable) process model where a systematic process is in place for continuously identifying and analyzing semantic artifacts; Level 3 (Defined) where data models and access mechanisms to the semantic artifacts are standardized; Level 4 (Quantitatively Managed) where metric models, management process and design policies can be precisely defined based on semantic artifacts. The metrics and policies once defined can be consistently applied for reporting, developing predictive models for future development tasks or continuously guiding the software evolution process.

References

[1] Hoek, A., Carzaniga, A., Heimbigner, D., Wolf, A.L.: A Testbed for Configuration Management Policy Programming. IEEE Trans. on Software Eng. 28(1) (January 2002)

[2] Draheim, D., Pekacki, L.: Process-Centric Analytical Processing of Version Control Data. In: Proc. of the Sixth IWPSE 2003 (2003)

[3] Lance, D., Unteh, R.H., Wahl, N.J.: Bytecode-based Java Program Analysis. In: Proc. of the 37th Annual Southeast Regional Conf. (1999)

[4] Sudaman, F., Mingins, C.: Evidence-based Management of Outsourced Software Projects. In: Proc. of the 2nd Conf. on SEAFOOD, Zurich (July 2008)

[5] Bevan, J., et al.: Facilitating Software Evolution Research with Kenyon. In: Proc. of the 10th European Software Eng. Conf., Lisbon, Portugal, September 5-6 (2005)

[6] Estublier, J., et al.: Impact of Software Engineering Research on the Practice of Software Configuration Management. IEEE TOSEM (2005)

[7] Chu-Carroll, M.C., Wright, J., Shields, D.: Supporting aggregation in fine grained SCM. In: Proc. of the 10th ACM SIGSOFT Symposium on Foundation of S/w Eng., USA (2002)

[8] Lanza, M.: The Evolution Matrix: Recovering Software Evolution using Software Visualization Techniques. University of Bern, Switzerland (2001)

[9] Lehman, M.M.: Laws of Software Evolution Revisited. In: Proc. of the 5th European Workshop on Software Process Technology, pp. 108–124 (1996)

[10] Madhavji, N.H., Tasse, T.: Policy-guided Software Evolution. In: 19th IEEE Int'l Conf. on Software Maintenance (ICSM 2003) (2003)

[11] Minsky, N.H., Rozenshtein, D.: A Software Development Environment for Law-Governed Systems. In: Proc. of the ACM SIGSOFT/SIGPLAN, vol. 24 (February 1989)

[12] Reißing, R.: Towards a Model for Object-Oriented Design Measurement. In: ECOOP Workshop on Quantative Approaches in OO Software Eng. (2001)

[13] Eick, S.G., et al.: Does Code Decay? Assessing the Evidence from Change Management Data. IEEE Trans. on Software Eng. 27(1) (January 2001)

[14] Sarkar, S., Sindhgatta, R., Pooloth, K.: A Collaborative Platform for App. Knowledge Mgmt. in Software Maintenance Projects. In: 1st Bangalore Annual Compute Conf. (2008)

[15] Girba, T., Ducasse, S.: Modeling History to Analyze Software Evolution. Int'l Journal on Software Maintenance and Evolution (JSME) (2006)

[16] Nguyen, T.N., Munson, E.V., Boyland, J.T.: An Infrastructure for Development of OO, Multilevel Configuration Management Services. In: Proc. of the 27th ICSE (2005)

[17] Xing, Z., Stroulia, E.: Analyzing the Evolutionary History of the Logical Design of OO Software. IEEE Trans. on Software Eng. 31(10) (October 2005)

Evaluating the Perceived Effect of Software Engineering Practices in the Italian Industry

Evgenia Egorova, Marco Torchiano, and Maurizio Morisio

Politecnico di Torino, corso Duca degli Abruzzi 24,
10129 Turin, Italy
{eugenia.egorova,marco.torchiano,maurizio.morisio}@polito.it

Abstract. A common limitation of software engineering research consists in its detachment from the industrial practice. Researchers have analyzed several practices and identified their benefits oand drawbacks but little is known about their dissemination in the industry. For a set of commonly studied practices, this paper investigates diffusion, perceived usefulness, and effect on the success for actual industrial projects. We found a match between academia recommendation and industry perception for more than 3 / 4 of the best practices. But we also found a few misperceptions of well-known practices.

Keywords: Software Process, Project Factors, Survey, Case-Control Study.

1 Introduction

According to IEEE glossary "Software engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software". Usually it involves work of a team headed by a project manager and interaction with customers in a competitive and volatile environment.

First problems with software development were identified during the NATO Garmisch conference in 1968. Since then academic research is focused on defining better practices for software testing, documenting, etc. However the rate of software project failures remains high [5].

Researchers identified several software engineering practices – both useful and harmful – and their expected effects on a software process outcome. Unfortunately the actual adoption of practices is not driven by scientific empirical knowledge. One reason for this could be poor communication between academic and industrial worlds.

Among the factors influencing the adoption of software engineering practices an important role is played by their perceived usefulness. This is one of the main factors identified in the technology acceptance model (TAM) [6].

The main goal of this work is to assess the perceived effects of several well-known software engineering practices on the success or failure of the projects. The assessment was performed on the basis of a series of interviews with practitioners working in the industry. We devised a classification schema for perceived usefulness and applied it to the survey results. As a result we obtained a deeper understanding of the practices' nature, their dissemination and perception among practitioners.

Q. Wang et al. (Eds.): ICSP 2009, LNCS 5543, pp. 100–111, 2009.

The paper is organized as follows: Section 2 gives an overview of related work. In Section 3 we describe practices selected for the study. In Section 4 we classify project factors and formulate hypotheses. Section 5 describes case study and Section 6 provides data analysis. In Section 7 we discuss the findings of the study. In Section 8 we address treats to validity. Conclusion and future work are presented in Section 9.

2 Related Work

In this section we describe works related to analysis of software engineering practices and their industry-wide adoption. We call these practices project factors.

2.1 Project Factors

There are mainly three types of studies focusing on the topic of successful process of software development: marketing writings that summarize sparse knowledge about how to make successful software projects; case studies that analyze success or failure of one or several projects by going into details; and survey studies that carry out analysis of data collected via questionnaires or interviews.

Papers of the first type present lists of general success factors, i.e. "making smart decisions" or "building the right team" [15]. Besides, often these studies go over the main points of why projects fail, mentioning "lack of top management commitment" [13] or "managers ignore best practices" [8]. Unfortunately appropriate practices often have poor implementation and even knowing the pitfalls of the bad practices for some reasons it seems impossible to avoid them in the real life projects.

The case studies analyze individual projects and investigate the reasons why they failed or the significant issues that led to the success [9], [19]. These studies are very informative and provide many details. The problem is that often this knowledge is difficult to apply in another settings.

And finally by analyzing empirical data of many projects, researchers test hypotheses and find the best practices or the biggest perils that are rather general and could be evaluated in various situations. These studies analyze data collected via questionnaires in order to answer such questions as "key factors for success in software process improvement" valid for small and large organizations [7], impact of the risk factors on time to market as one of the key success indicators [22] or best managerial practices for in-house projects [20]. Some consider only point of view of one group of stakeholders, i.e. developers in [14], or study different domains as in [1].

Besides there are few works [21], [23] that analyze possibility of prediction for success or failure of a software project based on the data from other projects.

2.2 Dissemination of Practices

Diffusion of practices is not easy. In 1991 Basili and Musa [2] stated that there was little success in transferring new knowledge and tools into active practice, explaining mainly that it is the process not the product that has to be adopted. In the same paper authors mention existence of the problems in communication between researchers and practitioners and need for proactive changes. In 1995, 96 two works by Iivari cover two aspects related to the CASE tools: factors that affected perception of the tools [11] and reasons why they were not widely used [12]. These results could be generalized for the rest of IT technology and summarized in the need for proper training, demonstration, time investment, management support, etc.

3 Selection of the Factors for the Study

The list of factors that influence project results could be almost unlimited. In order to perform a feasible study we selected the list of common factors in the recent literature, e.g. [1], [7] and number of others. We defined a list of 18 factors, divided into 4 sections: customer involvement, requirements engineering, project management and development process. Summary for all the factors is given in the Table 1.

Table 1. Factors

Customer Involvement	
CUST_INV	Customer was involved in the project
Requirements Engineering	
REQ_INIT	Complete and accurate requirements from the start
REQ_AFTER	Accurate requirements completed during project
REQ_METH	Use of specific method for requirements collection
REQ_TIME	Enough allocated time for requirements elicitation
Project management	
PROJ_SCHED	Good schedule estimations
PM_EXP	Experienced PM
PM_INSIGHT	PM understood the customer's problem
PM_LONG	PM supported long hours of his/her team
PM_REW_LONG	Staff was rewarded for long hours
CHAMPION	Commitment and support from sponsor/project champion
PM_CHANGE	Project changed PM
RISK_IDENT	Risk identification before project start
STAFF_ADD	Extra personnel added to meet schedule timetable
TEAM_SPIRIT	Having a good team sprit
Development process	
METRICS	Software process was monitored using metrics
QUALITY_RED	Reduction of quality (non-functional requirements, testing, etc) maintaining the requirements in order to finishing in Time/Budget
ELIM_REQ	Elimination of requirements maintaining the quality in order to finish in Time/Budget

Out of these 18 factors we consider STAFF_ADD, QUALITY_RED and ELIM_REQ to be dangerous software practices. We are not sure about effect of two factors (PM_CHANGE, PM_LONG) and will evaluate it empirically. Based on the prior research we believe that other 13 factors should be beneficial for a software project.

4 Factor Classification and Hypotheses

The research model of the study includes 18 risk factors, presented above. We believe that these factors could have different frequencies in terms of their occurrence within the projects. For example, thorough testing was performed in 60% of all projects. In addition, we argue that the factors could have different effects on the results of the projects. For instance, having collected all requirements at project beginning could be more important than working extra hours. In other words, each risk factor could be present or absent during a software project. Depending on whether each factor was present or absent, it could be evaluated as positive, negative or of no influence on the results. As a consequence, we formulate our high-level research questions as follows:

What is the perception of a factor X?

What is the effect of presence (or absence) of a factor X on the outcome?

Here and further on *factor X* is a general representation of the risk factors summarized in Table 1.

In the section 4.1 we propose classification of these factors. In section 4.2 we formulate a set of null hypotheses that would place each of the factors in its category according to the classification and help to answer research questions.

4.1 Classification

To the best of our knowledge, most of the existing studies typically addressed the issue concerning whether some factor has happened or not during project execution. Usually they correlate level of factor fulfillment (for example, level of PM experience) with success or failure. However there is a limited knowledge about perceived effect on the results of the project depending if a factor was present or not.

The perceived effect can be measured both when the factor is present and when it is absent. To achieve a unique result we classify the factors based on the combination of the two above. This combination provides a deeper understanding of a factors nature independent of the fact whether project actually failed or succeeded. In Table 2 we present possible combinations of the perceived effects based on whether factor was absent or present. It should be noted that "no effect" is coded as "0", "positive effect" as "+" and "negative effect" as "–". Obviously, total number of the patterns would be 9. Out of these 9, we exclude patterns such as [factor present/absent–negative effect] and [factor present/absent-positive effect], since we hardly imagine the situation in which any factor played opposite role.

For each of the categories we propose a key name. In order to better understand the taxonomy we provide a short explanation or an example for each category.

Categories *"verified"* and *"dangerous"* are considered as extremes of our classification. Having a good project manager is a factor that we expect to be in *"verified"* category. On the other hand, we suppose that requirements elimination is a bad practice, which could be put in *"dangerous"* category.

Table 2. Taxonomy of factors' categories based on perceived effect depending on whether factor was present or absent in project execution

		Factor present		
	Perceived effect	–	0	+
Factor absent	–		*underestimated*	*verified*
	0	*bombs*	*irrelevant*	*unrecognized*
	+	*dangerous*	*overestimated*	

Factors in the *"underestimated"* category are possibly commodities that are given for granted by those who posses them while their lack is felt as negative. For instance, think of a fridge in our houses: we may not consider having some positive influence on our life-style, but when it is missing or broken we realize how important it is.

In order to clarify *"unrecognized"* category we bring an email example. Before someone starts to use email as a communication media he/she might not see its usefulness or necessity. Once a person gets used to email, it is difficult to do without it. This example is similar to what is happening in some project factors. For instance, advantages of using certain technologies could be doubtful for non-users. We suppose metrics could be considered as one of these technologies.

We believe that *"bombs"* are difficult to be recognized. The difficulty lies in the fact that certain decisions might seem to be correct by the time they are actually made. For example, changing project manager seems to be a solution for a problematic project. But in reality changing manager might bring even more instability, which leads in worsening the situation.

We think that a bad implementation of a good software engineering practice could be an example of an *"overestimated"* factor. For instance, those respondents who have experienced in past an ineffective implementation of quality assurance process would be skeptical about its usefulness in future.

An example of an *"irrelevant"* factor could be any technological or human factor with no positive or negative evaluation in case of presence as well as absence.

4.2 Hypotheses

We formulate a set of high-level hypotheses for each pair of factor and category. We summarize high-level hypothesis using following generic parameterized formulation: $\boldsymbol{H_{xy} : F_x \in C_y}$, where $\boldsymbol{x \in F\{1..18\}}$, factors are given in Table 1; $\boldsymbol{y \in C\{A..G\}}$, categories are given in Table 2.

This generic high-level hypothesis is decomposed into 7 detailed high-level hypotheses:

H_{xC_A} *: factor X belongs to "verified" category.*

H_{xC_B} *: factor X belongs to "underestimated" category.*

H_{xC_C} *: factor X belongs to "unrecognized" category.*

H_{xC_D} *: factor X belongs to "irrelevant" category.*

H_{xC_E} *: factor X belongs to "bombs" category.*

H_{xC_F} *: factor X belongs to "overestimated" category.*

H_{xC_G} *: factor X belongs to "dangerous" category.*

By testing these hypotheses we will evaluate perceived effect of each factor. But in order to test what factors were present in successful projects and absent in the failed ones (or vice versa) we have to test one more hypothesis:

H_{xSF} *: Factor X was equally present in successful vs. failed projects.*

5 Research Method

5.1 Population and Sample

Piedmont region is among the most economically active areas of Italy. Different types of industry are concentrated in and around Turin, from machinery to consulting, from food industry to mobile phones. For this particular study we considered only companies whose core business is IT. Though we do understand huge amount of the non IT sector producing internal software.

We performed initial selection of the companies from the database of the Turin Commerce Chamber [4], considering those where software development is a core activity (based on the ATECO code [10]) and number of employees is more then 3 people. Threshold of 3 people was introduced in order to exclude individual consultants. Total number was 243 of such companies. Then using random sampling we have selected companies for the interviewing. Due to rather low response rate (25%) and time limits we have stopped on 33 responses which covers 13,58% of the Turin ICT sector.

5.2 Variables and Measures

For each of the factors described in Table 1 we have asked 2 types of questions. Firstly respondents answered if this factor happened or not for a project, and secondly which effect it had on the project's run. First question had binary yes/no response scale. Second one had 7-point Likert scale: from -3 "strong negative" till +3 "strong positive" with 0 as "no effect". The respondents evaluated results of each project subjectively. This outcome was codified on the binary scale as success/failure.

We have one type of independent variable and two types of dependent variables. Variables are given in Table 3.

Table 3. Independent and dependent variables

Independent variables	Dependent variables
Factor happened or not	Effect of the fact that factor happened or not on the result
	Outcome of the project

5.3 Hypotheses Refinement

In the light of the measures collected, the detailed high-level hypotheses presented above can be formulated as a conjunction of low-level hypotheses. We shall attribute a factor to a category if its relative high-level hypothesis is confirmed. High-level hypothesis is confirmed if its low-level hypotheses are all alternative. In Table 4 we provide subset of the null low-level hypotheses for each high-level hypothesis, where PE stands for perceived effect.

5.4 Data Collection Procedure

We have used case-control type of study design. "***Case-control study*** is a method of observational epidemiological study. Case-control studies are used to identify factors that may contribute to a medical condition by comparing subjects who have that condition (the 'cases') with patients who do not have the condition but are otherwise

Table 4. Subset of the null low-level hypotheses for each of the high-level hypotheses

High-level hypotheses	**Hypotheses** / **Category**	$PE_{present} \leq 0$	$PE_{absent} \geq 0$	$\|PE_{present}\| \geq 1$	$\|PE_{absent}\| \geq 1$	$PE_{present} \geq 0$	$PE_{absent} \leq 0$
H_{xC_A}	Verified	**R**[1]	**R**				
H_{xC_B}	Underestimated		**R**	**R**			
H_{xC_C}	Unrecognized	**R**			**R**		
H_{xC_D}	Irrelevant			**R**	**R**		
H_{xC_E}	Bombs				**R**	**R**	
H_{xC_F}	Overestimated			**R**			**R**
H_{xC_G}	Dangerous					**R**	**R**

similar (the 'controls')" [3]. In another words it is an application of history-taking that aims to identify the cause(s) of a result. In the terms of our study history is the information gathered through questionnaire and result is a successful or failed project.

The questionnaire contained several sections: general information about company and types of the software they develop, success definition in general, metrics and context information (type of project, customer, software, etc.) for one successful and one failed projects. In total there were 80 questions. Responses were collected via phone or email. Each phone interview lasted around 30 minutes.

5.5 Data Analysis Techniques

We used non-parametric statistics, such as Fisher's exact test for the analysis of binary-binary data and Mann-Whitney test, for binary-Likert data. For all the tests we adopted significance level of 5%. All quantitative analysis was made using R-project.

6 Data Analysis

6.1 Descriptive Statistics

We have gathered answers from 33 respondents via phone and email contacts. In total we have answers for 33 successful and 29 failed projects. Response rate was about 25%. Our respondents were project managers and developers.

6.2 Categorization of Factors

In this section we present the results of testing the low-level hypotheses that were defined in the 5.3. The results of the Mann-Whitney tests are summarized in terms of p-values in the Table 5. Significant p-values are printed in bold.

[1] **R** stands for "reject". Factor is assigned to a category if both related null hypotheses are rejected.

Table 5. Results of testing hypotheses for factors evaluations

Hypotheses Factors	$PE_{present} \leq 0$	$PE_{absent} \geq 0$	$\lvert PE_{present}\rvert \geq 1$	$\lvert PE_{absent}\rvert \geq 1$	$PE_{present} \geq 0$	$PE_{absent} \leq 0$	**Category**
REQ_INIT	**<0,001**	**<0,001**	1	0,4	1	1	Verified
REQ_AFTER[2]	**<0,001**	**0.016**	1	1	1	1	Verified
REQ_METHOD	**<0,001**	**<0,001**	1	0,07	1	1	Verified
REQ_TIME	**<0,001**	**<0,001**	1	0,9	1	1	Verified
PROJ_SCHED	**<0,001**	**<0,001**	1	1	1	1	Verified
PM_EXP	**<0,001**	**0,002**	1	1	1	1	Verified
PM_INSIGHT	**<0,001**	**<0,001**	1	1	1	1	Verified
PM_LONG	0,11	0,7	0,4	**<0,001**	0,9	0,4	NC[3]
PM_REW_LONG	**<0,001**	0,04	1	**<0,001**	1	1	Unrecognized
CHAMPION	**<0,001**	0,17	1	**<0,001**	1	1	Unrecognized
PM_CHANGE	1	0,34	0,8	**0,002**	0,04	0,7	NC
RISK_IDENT	**<0,001**	**<0,001**	1	0,9	1	1	Verified
STAFF_ADD	0,04	0,63	1	**0,003**	1	0,5	NC
CUST_INV	**<0,001**	**0,004**	1	1	1	1	Verified
TEAM_SPIRIT	**<0,001**	**0,015**	1	1	1	1	Verified
METRICS	**<0,001**	0,03	1	**<0,001**	1	1	Unrecognized
QUALITY_RED	1	1	1	**<0,001**	**0,006**	1	Bombs
ELIM_REQ	0,2	1	0,8	**<0,001**	0,8	1	NC

Since we executed multiple significance tests (3 when the factor is present and 3 when it is absent) on the same data, the probability of making Type I errors (i.e. rejecting null hypothesis when it is true) increases. To mitigate this issue we apply the Bonferroni correction [18] to the alpha level (initially 0,05). With this correction (0,05/3) statistically significant alpha level equals 0,017 ≈ 0,02.

We are able categorize 14 out of 18 factors. With collected data and the factors that were chosen for this study we covered 3 out of 7 possible proposed categories. We could not categorize the following four factors: STAFF_ADD, PM_CHANGE, PM_LONG, ELIM_REQ. Though we shall mention that respondents reported neutral effect on the absence of those factors. Effect of the factors' presence varied from negative to positive so no categorization was possible.

6.3 Factor Frequency

We have tested H_{xSF} for all 18 factors. In the Table 6 we present relative p-values and odds-ratios. Besides we provide percentages of successful and failed projects where the factors were present. Alpha level equals to 0,05. Since each test is independent, we don't need to apply Bonferroni correction. Significant p-values are printed in bold.

[2] This factor was evaluated only when REQ_INIT = N.

[3] NC stands for Not Categorized.

Table 6. Test of Hypothesis H_{xSF}

Factor (expressed as metric)	% of successful projects where this factor was present	% of failed projects where this factor was present	p-value	Odds-Ratio
REQ_INIT	61	24	**0,005**	4,7
REQ_AFTER	39	55	0,31	0,53
REQ_METHOD	48	31	0,2	2,1
REQ_TIME	73	52	0,12	2,5
PROJ_SCHED	91	59	**0,006**	6,8
PM_EXP	94	66	**0,008**	7,9
PM_INSIGHT	100	45	**0,000**	INF
PM_LONG	67	48	0,2	2,1
PM_REW_LONG	61	45	0,31	1,9
CHAMPION	42	34	0,61	1,4
PM_CHANGE	9	21	0,28	0,4
RISK_IDENT	76	45	**0,019**	3,8
STAFF_ADD	42	62	0,14	0,5
CUST_INV	82	59	0,055	3,1
TEAM_SPIRIT	100	59	**0,000**	INF
METRICS	55	31	0,13	2,3
QUALITY_RED	15	41	**0,026**	0,3
ELIM_REQ	33	31	1	1

Odds-ratio indicates whether a factor is successful (odds-ratio>1) or failed one (odds-ratio<1). We reject null hypotheses for the following factors: REQ_INIT, PROJ_SCHED, PM_EXP, PM_INSIGHT, RISK_IDENT, TEAM_SPIRIT, QUALITY_RED. This means that theses factors apparently discriminate between successful and failed projects.

7 Discussion

We have conducted a case-control study and collected quantitative and qualitative data. We aimed at checking the alignment between academia empirical findings and industry perception regarding which factors are good or bad for software development and analyzing level of adoption of software engineering practices in industry.

We found that 10 out of 13 good practices belong to the "verified" category (Table 5). All these verified factors are considered good practices in the research community. Therefore we can state that there is an agreement between academia and industry about the importance of such factors as: collecting requirements from the beginning or in the initial phases of a project, using appropriate methods and allow sufficient time to collect requirements, customer involvement, proper project management, schedule and risk management issues, and team spirit.

Contrary to our expectations, a few factors, namely, PM_REW_LONG, CHAMPION, and METRICS are in the "unrecognized" category: respondents who did not apply them reported no negative effect, though those who adopted them agree on a positive evaluation.

In contrast to the Brooks law and results of several studies, there was no agreement among practitioners about the negative effects of adding staff to a late project (STAFF_ADD). Staff was added in significant number of both successful and failed projects. Evaluated effect varied from strongly negative to strongly positive with no clear prevalence.

Based on the literature, before analyzing data, we expected such practices as reducing quality (QUALITY_RED) and eliminating requirements to respect time/budget constraints (ELIM_REQ) to be in the "dangerous" category. Surprisingly both factors were perceived as having no effect when absent, i.e. keeping quality and requirements is not considered particularly important for the project success. Besides, though respondents did agree that quality reduction is dangerous, there was no such agreement on the requirements elimination.

All seven factors that discriminate between successful and failed projects were categorizable. This means that there was an agreement between respondents from different projects, developing various types of software on the perceived effect of these factors.

We noticed rather low adoption (Table 6) of such engineering practices as using metrics to monitor process (METRICS) and using a well-defined methodology to collect requirements (REQ_METHOD). Respondents did report negative effect of *not* using any method for requirements elicitation. Not using metrics for monitoring the process was not evaluated as something negative.

It is important to mention that this is not a stable categorization. We expect that factors may move from one category to another mainly because of two aspects:

Time: unrecognized factors could become verified or "bombs" move to dangerous category in the future as academia disseminate its result in the industry. For instance, we expect METRICS to become a verified factor; among practitioners should grow knowledge that *not* collecting metrics is a failure factor.

Context: we believe that some factor's perception depends on the project context and type. For example, requirements elimination was evaluated positively as well as negatively. We believe that for market-driven projects this could be an approach to reduce the time to market, while for bespoke projects usually it is behavior that is now appreciated by the customer.

8 Threats to Validity

Although the results we presented are reliable, as in any empirical study, there are a few threats to their validity.

Conclusion validity concerns the relationship between the treatment and the outcome. We have used proper non-parametric statistical tests for data analysis and, when necessary, applied α-level corrections to mitigate the possibility of type I errors.

An important threat to *internal validity* is represented by the possible lack of accuracy in independent variables' measurement. The independent variables consisted in determining whether a given set of practices had been applied in a project; we believe that, for most factors, subjective biases could not influence such an assessment in a relevant way. We realize that objectiveness for few factors, like manager's experience (PM_EXP) or team spirit (TEAM_SPIRIT), could have been affected by the "halo effect" [16]. We have tried to address this threat by proper formulation of the questions. In fact data analysis showed large percent of positive evaluations for these few sensitive factors in the failed projects.

Construct validity concerns the relationship between theory and observation. In the classification schema we devised, a few categories were marked as impossible for our type of the study to make a consistent model. In addition all hypotheses were tested for all factors, a factor was assigned to a category only as a result of hypothesis testing and we did not force any factor to a category.

External validity concerns generalization of the findings to other settings than the one studied. We have studied only Italian companies in one of the regions of the country. The data sample is not very large though we conducted a proper random sampling of the population. We think our result can be generalized to other areas in north Italy and central Europe with similar industrialization levels. Clearly only a replication of this study in other settings could confirm this.

9 Conclusions and Future Work

This paper reported results from case-control study aimed at understanding success and failure factors in software projects in terms of practitioners perception. We proposed a classification of these perceptions. Our study included 18 factors, very frequent in the literature; we were able to classify 14 of them. We found that 77% (10 out of 13) of the known good practices (e.g. importance of good project schedule or complete requirements' list) are perceived correctly by the industry. For a few other practices (having a champion's support, using metrics, reducing quality) we noticed a lack of awareness in the industry. Moreover we observe a surprising confusion on a few principles, in particular Brook's law.

For the future work we will need more data points and other factors. For the replications of our study we would suggest to collect structured context information in order to better understand different types of software projects and their success drivers. Future work should also improve our knowledge of the effect of context cofactors on software engineering practices perception.

References

1. Berntsson-Svensson, R., Aurum, A.: Successful software project and products: an empirical investigation. In: Proceedings of the ACM/IEEE international symposium on Empirical Software Engineering, pp. 144–153 (2006)
2. Basili, V., Musa, J.: The Future Engineering of Software: A Management Perspective. IEEE Computer Magazine 24(9), 90–96 (1991)
3. British Medical Association, `http://www.bmj.com`
4. Camera di commercio di Torino - UNIMATICA di Torino: L'ICT in Provincia di Torino: La sfida dell'innovazione nel mercato globale. Turin (2006)
5. CHAOS report, `http://www.standishgroup.com`
6. Davis, F.D.: Perceived usefulness, perceived ease of use, and user acceptance of information technology. MIS Quarterly 13(3), 319–340 (1989)

7. Dybå, T.: An Empirical Investigation of the Key Factors for Success in Software Process Improvement. IEEE Trans. Software Eng. 31(5), 410–424 (2005)
8. Field, T.: When BAD Things Happen to GOOD Projects. CIO Magazine, pp. 55–62 (1997)
9. Holland, C., Light, B.: A critical success factors model for ERP implementation. IEEE Software 16, 30–36 (1999)
10. Istituto Nazionale di Statistica: Classificazione delle attività economiche Ateco (2007), http://www.istat.it/strumenti/definizioni/ateco/
11. Iivari, J.: Factors affecting perceptions of CASE effectiveness. European Journal of Information Systems 4(3), 143–158 (1995)
12. Iivari, J.: Why are CASE tools not used? Comm. of the ACM 39(10), 94–103 (1996)
13. Keil, M., Cule, P.E., Lyytinen, K., Schmidt, R.C.: A framework for identifying software project risks. Commun. ACM 41(11), 76–83 (1998)
14. Linberg, K.R.: Software developer perceptions about software project failure: a case study. Journal of Systems and Software 49(2-3), 177–192 (1999)
15. Reel, J.S.: Critical Success Factors In Software Projects. IEEE Software 16(3), 18–23 (1999)
16. Rosenzweig, P.: The Halo Effect: And the Eight Other Business Delusions That Deceive Managers. Free Press (2007)
17. Saridakis, T., Maccari, A.: Software architecture in industry: misuse and non-use. Technical report HK/R-RES 99/13 SE, University of Karlskrona i Ronneby 1999 (1999)
18. Strassburger, K, Bretz, F.: Compatible simultaneous lower confidence bounds for the Holm procedure and other Bonferroni-based closed tests. Stat. Med. (2008)
19. Sumner, M.: Critical success factors in enterprise wide information management systems projects. In: Proceedings of the Americas Conference on Information Systems (AMCIS), pp. 232–234 (1999)
20. Verner, J.M., Evanco, W.M.: In-House Software Development: What Project Management Practices Lead to Success? IEEE Software 22(1), 86–93 (2005)
21. Weber, R., Waller, M., Verner, J., Evanco, W.: Predicting Software Development Project Outcomes. In: Ashley, K.D., Bridge, D.G. (eds.) ICCBR 2003. LNCS, vol. 2689, pp. 595–609. Springer, Heidelberg (2003)
22. Wohlin, C., Ahlgren, M.: Soft Factors and Their Impact on Time to Market. Software Quality Journal 4, 189–205 (1995)
23. Wohlin, C., Andrews, A.: Evaluation of Three Methods to Predict Project Success: A Case Study. In: Bomarius, F., Komi-Sirviö, S. (eds.) PROFES 2005. LNCS, vol. 3547, pp. 385–398. Springer, Heidelberg (2005)

Evidence-Based Insights about Issue Management Processes: An Exploratory Study

Vahid Garousi

Software Quality Engineering Research Group (SoftQual)
Department of Electrical and Computer Engineering, Schulich School of Engineering,
University of Calgary, 2500 University Drive NW, Calgary, AB, Canada T2N 1N4
vgarousi@ucalgary.ca

Abstract. Issue (e.g., defect) repositories usually contain rich information that can be used to mine evidence about team dynamics, issue management processes, and other aspects of software development. The exploratory case study reported in this paper uses quantitative issue tracking data of three open-source projects to derive insights into how issues emerge and are handled in open-source projects. The mined information provides empirical evidence for a few beliefs in the software engineering and process communities. For example, depending on their specific context factors, projects show different degrees of responsiveness to the occurrence of defects. Software engineers can use techniques similar to those presented in this paper to mine the issue repositories of their in-house development projects. This may serve to better characterize their issue management processes, to perform self-assessment and evaluation on them, and also to identify process smells (symptoms) in those processes.

Keywords: Qualitative analysis, evidence report, issue management, issue processing, issue repositories, open-source projects.

1 Introduction

SourceForge.net is often referred to as the world's largest online repository for Open-Source Software (OSS) development [1]. SourceForge provides tools that allow open-source projects to develop, distribute, and support software.

One of these tools is the *tracker* system which provides issue tracking features and also development benefits (e.g., streamlining the management of code patches). This tool is used by many active SourceForge projects to manage four types of issues: *bugs*, *support requests*, *feature requests*, and *source code patches* [2].

Issue repositories usually contain rich amount of information which can be used to perform analyses to mine useful findings about team (group) dynamics, development processes, issue management processes and so on [3].

The exploratory case study reported in this paper uses quantitative issue tracking data of three OSS projects to derive insights into how issues emerge and are handled in OSS projects. Formulated using the Goal-Question-Metric (GQM) approach [4], the goal of this study is: to analyze and mine OSS issue repositories for the purpose of characterizing and getting insights into issue processing processes and their related collaborations with respect to effectiveness, efficiency and also responsiveness in

Q. Wang et al. (Eds.): ICSP 2009, LNCS 5543, pp. 112–123, 2009.

issue handling from the viewpoint of process engineers or project managers. With the above goal in mind, the following Research Questions (RQs) were posed, and appropriate metrics were used (or defined) to investigate them.

RQ 1. What are the trends of submission and handling of different issue types?

RQ 2. How responsiveness different teams are to different issue types? i.e., how many days does it take to address issues?

RQ 3. How do different issue types pile up and what are the possible negative consequences of issue pile up on projects in the long run?

While investigating the above RQs, it was also deemed appropriate to mine empirical evidence on a few typical beliefs on the topic in the software engineering community, e.g., is it true in the context of OSS projects that (1) there are more issues (mostly bugs) submitted after new releases?, or (2) bug reports are handled more quickly than other issues (e.g., feature requests) [2]?

It is believed that the findings of this exploratory study can be useful for researchers and practitioners in the following ways. First and foremost, it is one of the first novel steps towards characterizing issue processing in OSS projects, and can be adapted as an approach to analyze the performance and efficiency of such processes. It also acts as an evidence report that some active OSS projects may have similar reaction to issues as closed-source (commercial) software projects. Two example typical beliefs that this study provides evidence for are: (1) Bugs do actually dominate most of the issues, and (2). More bugs are usually submitted in the beginning of a project.

Practitioners can use techniques similar to those presented in this work to mine the issue repositories of their in-house development projects to perform self assessment and evaluation of their issue management processes. This can lead to findings and insights which can tell useful stories about, e.g., the times when the testing or debugging team was very active or responsive, or not as active or responsive as it was expected to. In addition, practitioners can also use the information mined from issue repositories to identify process *smells* (symptoms) [5] in their issue handling processes, e.g., not being reactive enough to issues. Once process concerns are identified, process improvement guidelines can be used to enhance issue handling processes.

The focus of this paper also aligns well with the conference theme this year, since analyzing issue processing processes and making sure there is a healthy trustful end-user relationship and support in place (by addressing users' issues in a timely manner) are among the major steps towards having a trustworthy software project.

2 Related Works

Mining Software Repositories (MSR) [3] is a very active research area which is gaining increasing popularity in both academia and industry. However, there are only a handful number of works in the MSR literature relating to software processes in specific (e.g., [6-8]), while most works rather focus on other subjects such as source code repositories, and social networks among developers [3].

By analyzing issue repositories, the study in [6] empirically showed that timely bug fixing is considered as a success factor for OSS projects, and is correlated with other success factor, e.g., user satisfaction and involvement. One of the findings in [6] was that the bug fixing time is correlated with activity rank, and number of downloads of an OSS project. Among the projects studied in [6], projects with more developers reported more bugs and then fixed those bugs somewhat more quickly, i.e., as one would expect, there was a negative correlation between the bug fixing time and the project community size.

The article in [7] mined the coordination practices used for bug fixing in OSS teams. Based on a codification of the messages recorded in the bug tracking system of four projects, the authors identified the accomplished tasks, the adopted coordination mechanisms, and the role undertaken by both the development team and the open-source community. The findings revealed that the process is mostly sequential and composed of few steps, namely: *submit (open)*, *analyze*, *fix* and *close*. The authors believed that the bug fixing process seems to lack traditional coordination mechanisms such as systematic task assignment. As a consequence, labor is not equally distributed among process actors. Few people contribute heavily to all tasks whereas the majority just submit one or two bugs. Also, the organization structure involved in the process resembles the *bazaar* metaphor proposed by [9] for the open-source processes. Few actors (core developers), usually team project managers or administrators, are mostly involved in bug fixing. No evident association was found among coordination practices and project success.

The work in [8] presented a novel approach to discovering software processes from OSS web repositories, combining techniques for text analysis, HTML hyperlink analysis, and of repository usage and update patterns. The work proposed the use of text analysis techniques for extracting instances of process meta-model entities from the content of the community repositories, followed by hyperlink analysis to assert relationships between the mined entities in the form of process events. It then applied usage and update patterns to guide integration of the results of text and link analysis together in the form of a process model. Unfortunately, a real application of the technique was not presented in the paper [8].

A special-issue of “Computers in Industry” journal in 2004 was devoted to process and workflow mining [10]. However, there were no papers in that special-issue to focus on mining issue repositories and processes.

To the best of the author’s knowledge, [6, 7] are the only works in this particular area, but none of them analyze the issue repositories form the perspective of the three RQs in this study, nor do they provide evidence on the typical beliefs in this area, e.g., is it true that more bugs are submitted after new releases? Our case study is “exploratory” in that it aims at identifying the most challenging questions in this new (less-explored) area of research and opening the horizons for more future work.

3 Design of Case Study

For this empirical study, the issue repositories of four SourceForge projects were analyzed. The flow of different issue types along the issue processing process, supported by SourceForge, was carefully analyzed. Although SourceForge does not clearly specify the issue handling process for its projects, it pre-defines the following status codes for issues: *open*, *closed*, *pending* and *deleted*. These status codes are usually used by project administrators to streamline the issue handling process. By going through a few typical issues, it is possible to construct a typical issue handling process for SourceForge projects which is shown in Figure 1.

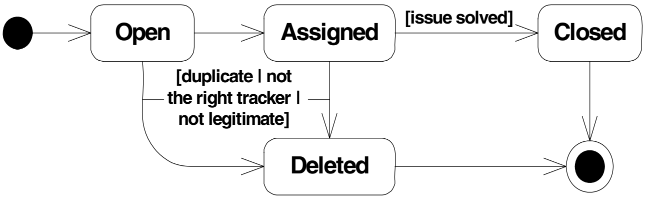


Fig. 1. A typical issue handling process for SourceForge projects

When an issue is first submitted, it is *open*. A project administrator either assigns it to a developer member of the project to be taken care of, or marks it "deleted" if either the issue is a duplicate (was reported in the past), not legitimate (a reported bug is not really a bug), or if the issue is posted in the wrong tracker. When an assigned issue is implemented or solved (e.g., via a bug fix), the issue is closed.

There were two sources of data for this study: (1) the tracker statistics from the SourceForge website, and (2) the SourceForge Research Data Archive (SRDA) [11]. The former provides interesting aggregate issue statistics such as number of issues opened or closed per month. SRDA [11] is a repository of SourceForge research data made available by a team of researchers from the North Carolina State University.

Four OSS projects hosted on SourceForge were chosen systematically as objects of this study. A set of different variability dimensions were used in the systematic sampling, e.g., programming language, LOC, time of registration in SourceForge, number of developers and popularity. To focus on realistic issue management processes, it was also desired to filter out inactive or small projects (with small LOCs). It was set that a minimum 10 KLOC is required. Also, a minimum of 90% SourceForge activity percentile was required.

It was observed that the majority of projects on SourceForge do not have tracker systems or their tracker systems are private (accessible to project administrators only), for instance, the MySQL project. It was also observed that a large number of projects, with reasonable activity percentiles, had public tracker systems, but they were mostly empty or had only a few items posted, e.g., the *DeveloperEdition* project. Thus, the other criterion was that the project should have at least 10 issues in its tracking system.

As of May 2008, there were 152,489 projects hosted on SourceForge [11]. Applying the above selection criteria (running the appropriate SQL SELECT statement on the SRDA source [11]) filtered the above large pool of projects to 2,368. The projects were then clustered to different programming languages, time of registration, and number of developers. By limiting our focus to projects written in Java, Assembly, C, C+, and Python as popular programming languages and applying a random selection, we selected four projects: jEdit, DrPython, FlasKMPEG and JFreeChart. Due to space constraints, the analysis based on only the first three projects are presented in this paper. The key information and statistics about the three projects are shown in Table 1.

jEdit (http://sourceforge.net/projects/jedit) is a source code editor written in Java. It can be configured as a powerful IDE through the use of its plug-in architecture. DrPython (http://sourceforge.net/projects/DrPython) is a customizable cross-platform IDE to aid programming in Python. FlasKMPEG (http://sourceforge.net/projects/flaskmpeg) is an MPEG trans-coding program.

Table 1. Key information and statistics about the three projects in this study

	jEdit	DrPython	FlasKMPEG
Programming language	Java	Python	C, C+
KLOC	71	15.4	78.5
Registered in SourceForge	Dec. 6, 99	Jan. 1, 05	Apr. 6, 2000
Number of project administrators (as of Sept. 2008)	9	2	2
Number of developers (as of Sept. 2008)	158	4	5
SourceForge activity percentile (as of the week of Sept. 1, 2008) :	99.98%	97.84%	91.42%
Average number of issues submitted (per month)	52.15	3.21	0.32
Average number of issues closed (per month)	46.28	3.14	0.02
Average number of downloads (per month)	45,112	2,171	58,563
Hits in Google (as of Sept. 2008)	820,000	38,900	164,000

The average numbers shown in Table 1, e.g., issues submitted per month, are calculated from the registration (inception) time of each project until (and including) June 2008. Apparently, the SourceForge statistics server experienced a few technical difficulties in July 2008 due to migration of their servers and, consequently, the data for July and August 2008 do not seem reliable. The two last rows in Table 1 are meant to provide simple measures for the popularity of each tool.

4 Analysis of Results

4.1 Research Question 1

For RQ1, the trends of issue submission and handling for different projects and different issue types after their new releases were analyzed. The version (release) information of the three projects including the release dates were retrieved from SourceForge. The number of issues opened and closed in each month for each project since its inception date was also retrieved.

The three *stacked* charts in Figure 2 present the above information. The x-axis denotes the dates. The left-side y-axis are the number of Issues Opened (IO) and Closed (IC) in each month. The right-side y-axis and the vertical lines inside each chart denote the release numbers of each project. For example, version (release) 4.3 of jEdit was released on April 2008. In this month, there were, respectively, 84 and 52 issues opened and closed for this project. Minor versions such as 4.3.2 have been converted to real numbers (e.g., 4.32) to make them representable in the chart. Note that, to present the results in the most possible effective way, the stacked chart type was chosen. Thus, the left-side y-axis should not be misinterpreted by considering IC values higher than IO values. The most notable observations based on Fig. 2 follow.

Scale of issues: Average number of issues submitted (per month) for the three projects (jEdit, DrPython, and FlasKMPEG), in order were: 52.15, 3.21, and 0.32. Also the average number of issues closed (per month) in order were: 46.28, 3.14, and 0.02. Thus, one could notice the difference of scale in the number of issues per project. Such a difference could be interpreted due to different possible reasons, e.g., (1) a naïve one being that, due to poor quality, jEdit really has more issues (e.g., bugs) compared to the two other systems, (2) jEdit has more users and since more people are using it, more people find bugs or suggest more feature requests. For the high rate of issue closing, one possible factor might be the high number of (active) community members (developers) and/or more proactive project administrators. Determination of the exact root cause(s) of the above observation would certainly require more investigations on the above possibilities.

Continuous flow of issues: There are steady numbers of opened and closed issues (IO and IC) in jEdit, e.g., there is no single month with zero IO or IC. However, those measures are quite sparse in DrPython and FlasKMPEG. This shows that, unlike the two other projects, the users of jEdit have kept submitting issues continuously since June 2000 (a few months after its inception in Dec. 1999).

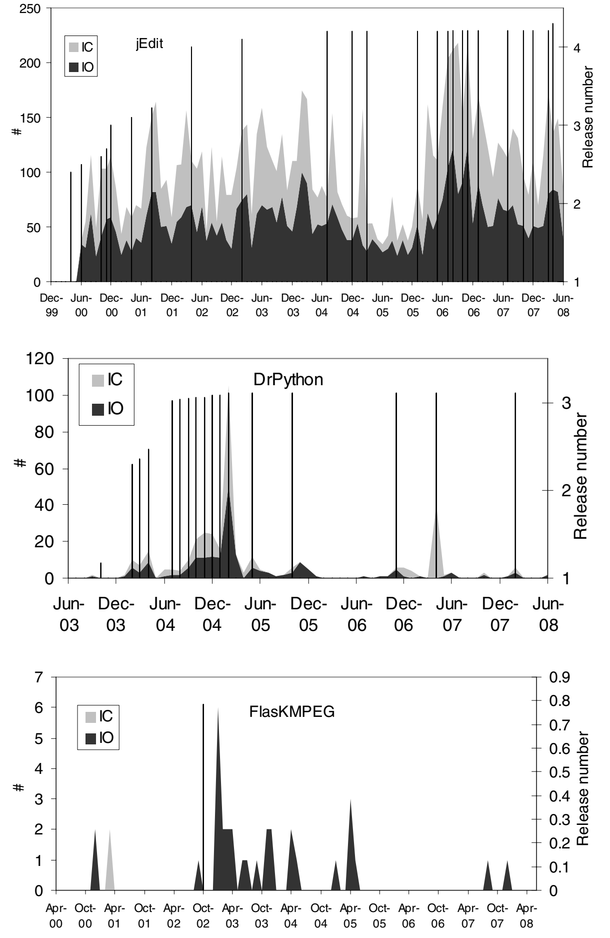


Fig. 2. (stacked chart) Trends of opened and closed issues vs. project versions

It also seems that the developers of jEdit have actively processed and closed the issues. To compare the three projects in term of variations in IO and IC, the standard deviation of the values could not be used since the scales of the measures in three projects are different. Alternatively, the *coefficient of variation* metric was used (defined as the standard deviation divided by the mean value). The coefficients of variations are shown in Table 2.

The coefficients indicate that statistically jEdit had the lowest variation (thus the highest stability) in terms of in- and out-flow issues. For DrPython and FlasKMPEG, issues are opened and closed in different months. This seems to indicate that their users and developers (issue handlers) did not find (and report) issues continuously.

Issues in and around releases: Issues for DrPython are not a continuous flow and they are rather submitted on or shortly after each of its releases. The situation for FlasKMPEG also seems similar. The new issues of jEdit seem to have been submitted in a burst pattern upon its releases.

Table 2. The coefficients of variation

Coefficient of variation	Projects under study		
	jEdit	DrPython	FlasKMPEG
The number of issues opened	46%	222%	322%
The number of issues closed	62%	278%	1260%

Continuous efforts on opening and closing issues: The allocations of effort to open and close issues are different among the three projects. In JEdit, continuous efforts are devoted to open and close (handle) issues. For DrPython, the two efforts somewhat appear together, e.g., more issues are opened around Feb. 05, and most issues are also closed in that time period. However, the situation for FlasKMPEG is different as more issues were opened in the middle of the period under study, while there is not much effort to address them. Such a trend, i.e., having *left-over* pending issues, might lead to negative symptoms in the long run, e.g., users will find their issues not addressed and might lose interest in the project, and issues will pile up in large numbers, requiring more and more efforts to handle them all. Assessing whether and to what extent those negative symptoms would occur in the down-stream of the process requires further investigations.

The trends of opened and closed issues per issue type were also investigated in the study. Due to space constraints, only the trends for three of the four default issue types of jEdit are presented in Figure 3 Support requests are not enabled in jEdit (the feature is turned off). Data series BO, PO and FO in Figure 3 denote, respectively, the number of bugs (B), patches (P), and feature requests (F) opened in each month. Similarly, BC, PC, and FC indicate the number of closed issues of the above types. The most notable observations based on Figure 3 follow.

Bugs dominate the issues: It seems a large proportion of all issues are bugs. Quantitatively speaking, about 60% (3,249 of 5,372) of all issues in jEdit as of July 1, 2008 were bugs.

Only bugs until March 2006: As the chart shows, all issues submitted before this date were only bugs. Patches and feature requests started to appear in the tracker afterwards. A naïve reasoning might conclude that no one submitted those issues until that particular date. But another more possible reason might be that the trackers for those issues were turned off before that date. Since the dates of such changes are not available to outsiders of a project, we could not determine what really the main reason was.

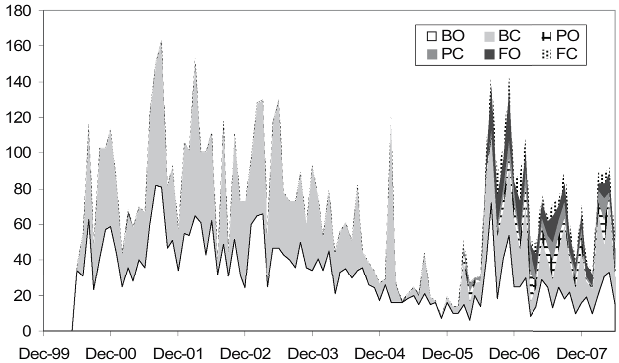


Fig. 3. (stacked chart) Trends of opened and closed issues per issue types for jEdit

More bugs in the beginning: It seems that more bugs were submitted (opened) in the beginning of the jEdit project compared to the later time frames. For a deeper analysis, the trend lines of BO and BC series were investigated (not shown here due to space constraints). It was observed that the data series of jEdit's opened bugs have had two major peaks; the highest one occurring around early-2002 (about 2 years after the project inception), and the second pick around late 2006. The first peak is usually normal in the initial releases of new projects, and it is considered healthy if the bug submissions do not keep a steady or a rising trend (which is the case in jEdit).

The second peak in bug submissions actually coincides with introducing a new major feature in jEdit, i.e., support for plug-ins. This brought in a wave of new bugs and was continued steadily for a few months.

The trend of closing bugs has some delay after they are opened: By an analysis on the trend lines of BO and BC, we found out that the BC trend almost follows BO with a slight delay, especially in the beginning of the timeline. This is an interesting observation and is an evidence that the bug closing efforts usually started after a pile of bugs were opened in this project.

Sharp rises in closed bugs: Variation coefficients of BO and BC for jEdit in Figure 3 are about 59% and 76%, respectively. This means that the rate of bug closings changes more sharply compared to the rate of bug openings. One possible justification for this phenomenon is that the jEdit developers (debuggers) perhaps wait for a while for bugs to pile up and then fix (close) them in one short period of time. Such an approach to bug fixing is often called *bug days* in the software industry.

4.2 Research Question 2

To study how responsive different teams are to different issue types, a new metric was defined for closed issues, referred to as: *Days To Close (DTC)=The closing date of a closed issue – Its opening date*. For example, if a bug is opened on Jan. 1, 2005 and is closed on Feb. 1, 2006, its DTC would be 396 days.

Figure 4 shows the histograms of DTC for different issue types for jEdit and DrPython. The DTC measures for FlasKMPEG are not shown since only 2 of all 32 open FlasKMPEG issues were closed as of July 2008 and the only two DTC measures are not representative for the purpose of this study.

Critical issues are handled fast: It is interesting to observe that in both projects, the issues which are closed, are closed soon after they were opened (in less than 10 days in most of the cases). Although high average item ages were observed in these projects, it seems that the team members prioritize the issues and process those which are more important (critical). By manually looking at the issue priorities for a few issues which were closed fast, this hypothesis was confirmed.

Bugs are handled faster than the other issue types: There are more bugs fixed in the same day (point 0 in the histograms) than the other two issue types (i.e., feature requests and patches) in Figure 4 Detailed statistics for closings of bugs, feature requests and patches are shown in Table 3. As of Sept. 2008, 90.4% of jEdit bugs are closed, 23% of which are closed bugs in the same day they were submitted. This is quite amazing as 677 jEdit bugs received a prompt same-day response (they were either fixed, marked as duplicate, etc.). 40% of DrPython's closed bugs were processed on the same day. By looking at the statistics in this table, one can observe that bugs were handled faster than the other issue types, in general. This situation is expected as bugs are considered the most serious type of issues.

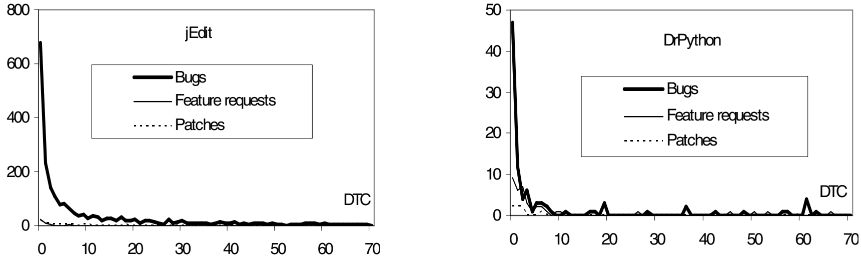


Fig. 4. Histogram of *Days To Close (DTC)* for different issue types

Table 3. Statistics for handling different issue types (from the project inception time to July 2008)

Issue Type	Metrics	Project	
		jEdit	DrPython
Bugs	Numbers of submitted bugs	3235	126
	Numbers of closed bugs	2926	118
	% of closed bugs	90.4%	93.6%
	% of closed bugs in the same day	23%	40%
	% of closed bugs in one day	8%	10%
Feature requests	Numbers of submitted feature requests	270	51
	Numbers of closed feature requests	99	51
	% of closed feature requests	36.6%	100%
	% of closed feature requests in the same day	21%	18%
	% of closed feature requests in one day	9%	12%
Patches	Numbers of submitted patches	153	18
	Numbers of closed patches	148	18
	% of closed patches	96.7%	100%
	% of closed patches in the same day	13%	11%
	% of closed patches in one day	7%	11%

4.3 Research Question 3

To investigate how different issue types pile up and what the possible negative consequences of issue pile up are on projects in the long run, the average issue age metric provided by SourceForge (for each issue type) was used. This metric denotes the average wait time of an issue in the queue until it is processed (closed).

Figure 5 visualizes the average ages of issues (in days) per issue type in each of the three projects. The following acronyms are used in this figure:

- ABA: Average Bug Age
- APA: Average Patch Age
- AFA: Average Feature request Age
- AIA: Average Issue Age (an issue can be either a bug, a patch, or a feature request)

For example, if the first issue is submitted on July 1st, and the second one on August 1st of the same year, the AIA on September 1st will be: (31+62)/2=46.5 days. The most notable observations based on Figure 5 follow.

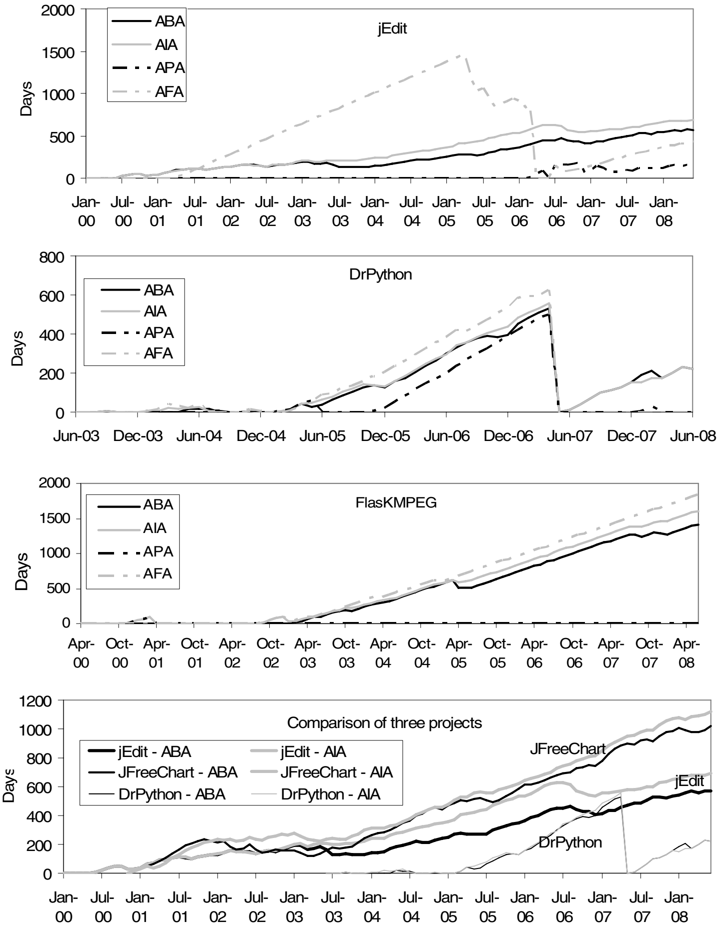


Fig. 5. The average age of different issue types

ABA follows AIA: In all three projects, the ABA mostly follows the trend of the AIA. This is mainly since bugs constitute most of the issues in all projects, and thus opening and closing bugs affects both ABA and AIA.

ABA is usually less than AIA: In all three projects, ABA is less than AIA in most months. This seems to indicate that bugs are given the highest priority among all issue types while the average age of other issue types is generally larger than that of bugs.

Different scales: As Figure 5-(d) shows, the three projects have different scales in terms of ABA and AIA. For jEdit, being the earliest project to start and with more issues submission per month, the AIA was 692.5 days (as of June 2008) and ABA=569.4 days. At the same month, for DrPython, AIA=223 and ABA=223, and for FlasKMPEG, AIA=1607.1 and ABA=1420.6.

Different approaches to process issues: One can also see that there are not drastic falls in AIA and ABA for jEdit or FlasKMPEG. However, the measures for DrPython have a sudden fall in April 2007. A closer look at the number of bugs and issues closed in this month for DrPython reveals that all the 40 open issues at the time (18 bugs, 10 patches and 12 feature requests) were closed in one month period. This might be an indicator of a somewhat extraordinary (brave!) move by a few project members to close all the pending (open) issues. Such drastic moves in AIA and ABA are not observable in jEdit or FlasKMPEG. Although one could see that there are slight falls in AIA and ABA in both of these projects which denotes the proactive intent to process open issues. There is a sudden fall in AFA in jEdit starting in April 2005 and ending in April 2006 (making AFA=0). A closer look at the AFA in this case reveals that the AFA was constantly growing until April 2005 (since there were only open 2 requests as of that date and they were *aging* on a constant rate). In the few months after April 2005, three new feature requests were submitted (thus lowering the AFA). All the 5 open requests were handled and closed on April 2006.

Different slopes (reactiveness to issues): According to the chart in the bottom of Figure 5, ABA and AIA have the lowest (growth) slopes in jEdit compared to the two other projects (except the sudden falls). This might be an indication of the jEdit team being more reactive to issues. Low responsiveness in handling issues can have various implications for the project, e.g., if a project team is not too reactive to issues, open issues can get older and older, which in turn can give the imprecision that the issues are not paid attention to (or perhaps, ignored). On the other hand, being more reactive can help control the queue of issues from getting very long. For example, imagine the possible queue size, AIA and ABA of jEdit issues if its issue pile-up curve had a similar slope as DrPython and FlasKMPEG.

Low responsiveness to issues, as a process smell (symptom), might lead to many negative results in the long run for a project, e.g., users (clients) will find their issues not addressed on-time and might lose interest in the project, or (in the case of industrial projects), they may decide to cancel their contract with the software company.

5 Conclusions and Future Works

This case study used the quantitative issue tracking data of three OSS projects and helped us get qualitative insights into the process of how issues are piled up and addressed. This in turn provided the evidence on a few typical beliefs on the topic in the community, e.g., (1) Bugs dominate the issues, (2) More bugs are usually submitted in the beginning of a project, and (3) Depending on other success or activity factors, different projects show different responsiveness to bugs and other issues.

Also, it was observed that a reasonable percentage of bugs were fixed in the same day or in just few days after they were opened. Such an observation seems to specially be the case for bugs with higher priorities. This is an evidence confirming that (critical) bugs are also usually taken seriously in more active (and successful) OSS projects similar to critical industrial projects.

Note that, due to space constraints, only a small part of the findings of our case study was reported in this paper. More results are expected to be published in other papers in the near future, e.g., the study findings for JFreeChart, and investigation of existing correlations between the trends of issue opening and closing, and other project success metrics, e.g., the number of downloads.

To take further steps in studying issues pile up in OSS projects, it is planned to apply the queuing theory and software process simulation on issue processing queues. Parameters such as issue arrival rate and issue processing rate can be used to model and analyze the behavior of the process. Also, there are plans to use the data mined in this work to systematically identify process *smells* (symptoms) in issue handling processes. For example, if the slope of the ABA curve in a time step is greater than a defined slope threshold, we could say that the process suffers from the following process smell: "not being reactive enough to issues" [5].

Acknowledgements

This work was supported by the Discovery Grant no. 341511-07 from the Natural Sciences and Engineering Research Council of Canada (NSERC). The author acknowledges the access he was granted to the SourceForge Research Data Archive (SRDA) [11]. The author would also like to thank Jonathan Sillito, Guenther Ruhe and Dietmar Pfahl for their valuable comments and suggestions on the early drafts of this article.

References

1. Mayer-Schönberger, V., Lazer, D.: Governance and Information Technology: From Electronic Government to Information Government. MIT Press, Cambridge (2007)
2. `http://alexandria.wiki.sourceforge.net` (last accessed, August 2008)
3. Xie, T., Pei, J., Hassan, A.E.: Mining Software Engineering Data. In: Companion of the International Conference on Software Engineering, pp. 172–173 (2007)
4. Basili, V., Caldeira, G., Rombach, H.D.: The Goal Question Metric Approach. In: Marciniak, J. (ed.) Encyclopedia of Software Engineering. Wiley, Chichester (1994)
5. Ambler, S.: Model Reviews: Best Practice or Process Smell?, `http://www.agilemodeling.com/essays/modelReviews.htm` (last accessed, August 2008)
6. Crowston, K., et al.: Towards a Portfolio of FLOSS Project Success Measures. In: Workshop on Open Source Software Engineering, International Conference on Software Engineering, pp. 29–33 (2004)
7. Crowston, K., Scozzi, B.: Coordination Practices for Bug Fixing within FLOSS Development Teams. In: Proc. of the Int. Workshop on Computer-Supported Activity Coordination (2004)
8. Jensen, C., Scacchi, W.: Data Mining for Software Process Discovery in Open Source Software Development Communities. In: Proc. Workshop on Mining Software Repositories, pp. 96–100 (2004)
9. Raymond, E.S.: The Cathedral and the Bazaar. First Monday Journal 3(3) (1998)
10. v. d. Aalst, W.M.P., Weijters, A.J.M.M.: Process mining- a research agenda. Computers in Industry, Special issue on Process / Workflow Mining 53(3), 231–244 (2004)
11. Madey, G. (ed.): The SourceForge Research Data Archive (SRDA). University of Notre Dame (2008), `http://zerlot.cse.nd.edu` (last accessed: September 2008)

Process Aspect: Handling Crosscutting Concerns during Software Process Improvement

Jia-kuan Ma, Lei Shi, Ya-sha Wang*, and Hong Mei

Key Laboratory of High Confidence Software Technologies, Ministry of Education
School of Electronics Engineering and Computer Science, Peking University, Beijing, 100871, China
{majk06,shilei07,wangys}@sei.pku.edu.cn, meih@pku.edu.cn

Abstract. A frequently emerging situation in process improvement is adding new concerns into existing processes. Implementing these concerns calls for changes over a series of tasks, roles, work products and tools, which usually crosscut different modules of existing process models. Lacking systematic modeling of these crosscutting concerns may raise difficulties in understanding, managing, and reusing their implementations. Aiming at such problems, in this paper we propose leveraging Process Aspect to handle these crosscutting concerns. Modeling and weaving process aspects into SPEM2.0-based processes are presented. Finally, an example is provided as a case study.

Keywords: Software Process Improvement, Process Aspect, Crosscutting Concern, SPEM 2.0.

1 Introduction

According to ISO/IEC 15504 [2], software process improvement is defined as "actions taken to change an organization's processes, so that they meet the organization's business needs and achieve its business goals more effectively". To advance with continuous evolving business needs and goals, adding new concerns into existing processes becomes a frequently emerging situation during process improvement.

On the other hand, to better understand and manage the rich information of software process, commonly a process model is organized as a hierarchy of modules, following the "Separation of Concerns" principle. For example, OpenUP [3] divides its content into 6 disciplines: Requirements, Architecture, Development, Test, Project Management, and Configuration Management. Each discipline further contains a set of tasks, roles, work products and tools.

Some new concerns during process improvement focus on only one discipline. For example, reduce code defect rate via improving test discipline. Still, there are numbers of concerns that crosscut many tasks, roles, work products and tools from different disciplines. In [4], security, accounting, resource management are proposed as examples of crosscutting concerns. Moreover, intellectual property, customer

[*] Contract Author.

Q. Wang et al. (Eds.): ICSP 2009, LNCS 5543, pp. 124–135, 2009.

involvement, systematic reuse are also common crosscutting concerns emerging in software process improvement.

As an example, let's consider the Intellectual Property (IP) concern. With reuse being widely accepted, more and more developers begin to use third-party code in their projects. However, many third-party codes are published under certain licenses, e.g. GNU, MIT. Improper usage of such codes can lead to severe technical and commercial risks. To avoid this, IP as a new concern should be incorporated into existing process models. According to our study in 10 Chinese software companies over Beijing, Qing-Dao and Kun-Ming, implementation of the IP concern will at least crosscut two different disciplines (taking OpenUP as an example): project management and development, involving a number of activities, roles, work products and tools, as illustrated in Fig. 1.

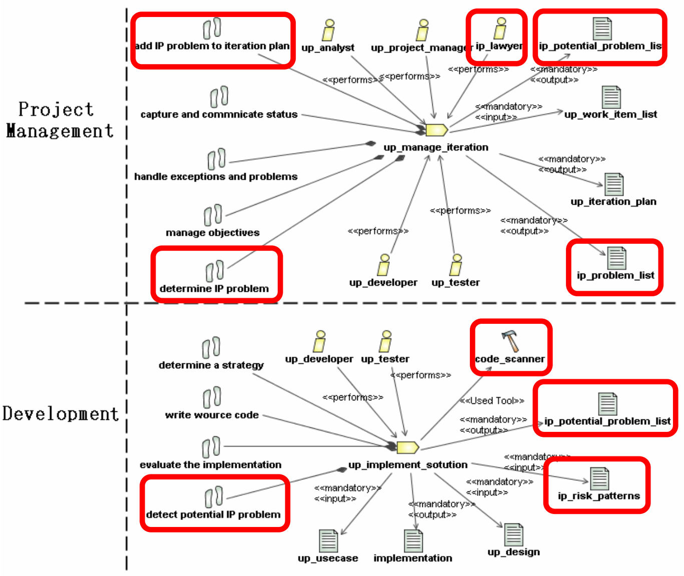


Fig. 1. A schematic implementation of the IP concern in OpenUP. Elements inside the red rectangles are new tasks, roles, work products and tools added for IP.

In order to provide support for IP management, a series of elements (tasks, roles, work products and tools) should be added. These "add" operations aim at solving the same concern, but spread over different disciplines. Without systematic modeling these operations as a whole, several problems may arise:

1. It is by no means easy to understand a crosscutting concern and its implementation. The collection of change operations actually constitutes certain knowledge on dealing with a concern. If there is no place where all the changes are formally modeled, such knowledge will scatter over the improved process, making it hard to understand.

2. With further process improvement, implementation of a crosscutting concern may also be placed in continuous evolution. Heavy workload is called to modify an implementation which has already spread over process model.

3. An organization usually has several similar processes in use (e.g. variants of OpenUP), forming the so called process family [5]. For many crosscutting concerns deriving from organizational consideration, they are intended to apply in several similar processes. Without systematic modeling, implementations of such concerns can not be reused among these processes.

Aiming at these problems, in this paper we introduce Process Aspect to handle crosscutting concerns during process improvement. Applying process aspect can be divided into two steps. First, model the implementation of a crosscutting concern as a process aspect. Second, weave the process aspect into existing process model.

The rest of this paper is structured as follows: section 2 provides the process aspect model and a corresponding XML schema for describing process aspect; section 3 presents the mechanism for weaving a process aspect into SPEM2.0-based[1] processes; section 4 is an example of applying process aspect on the IP concern. Finally, we discuss related work in Section 5 and conclude in Section 6.

2 Process Aspect

Similar to the case in software aspect (e.g. AspectJ [6]), definition model and weaving mechanism of process aspect are greatly influenced by the target process language (e.g. Little/JIL [7], SPEM2.0). In this paper, we choose SPEM2.0 as the target process language. SPEM2.0 is a software process engineering meta-model standard, released by OMG in April 2008. Currently, SPEM 2.0 is supported by most mainstream process modeling tools such as EPF [8], Fujitsu DMR Macroscope [9], etc.

Two major features of SPEM2.0 are:

a) Explicitly distinguish reusable method content from its application in process. For example, a role's definition and its multiple usages in different contexts are respectively separated into method content and process use. Since our work can apply both on method content and process use, we do not distinguish the naming difference between definition and usage. For example, we use Role instead of RoleDefinition (in method content) or RoleUse (in process use).

b) Provide flexible variability and expansibility via method plugin package. SPEM2.0 provides comprehensive semantic specifications for method plugin elements. Tools supporting SPEM2.0 with 'SPEM Complete' compliance point all implement such semantics. Later in section 3, we will leverage method plugin package to implement the weaving of process aspect. The result is a method plugin package, which can be further integrated with existing process models by tools like EPF, Fujitsu DMR Microscope, etc.

2.1 Process Aspect Model

Just like it was in software aspect, we start with defining the join point, and then the two parts of an aspect: pointcut and advice.

Join Point. A join point specifies one possibly changed element in an existing process model when applying a process aspect. In this paper, every task, role, work product and tool in SPEM2.0 process models can be a join point. This is quite different from software aspect, where points in program execution, e.g. calling a function, setting a field, etc. We define join points as elements in software process models, instead of certain time points in process execution. The reason is that our work aims at applying process aspect to cast a series of changes onto existing process models, so that they can be transformed to new models, meeting the given crosscutting concern.

Pointcut. Pointcut is simply a set of join points, on which a certain advice should apply. A pointcut results from certain filtering among join points, therefore it's important to define the filtering mechanisms, namely pointcut designators.

As our work is based on SPEM2.0 process models, we provide 4 different pointcut designators corresponding to the 4 types of join points: f*ind_tasks, find_roles, find_workproducts,* and *find_tools*.

Distinguishing pointcut designators by types means in our model, a pointcut actually consists of join points with the same type. This is meaningful because it's generally very rare that join points with different types can share one same advice, which will be further discussed when introducing Advice. Also, for sake of later discussion, we define *pointcut type* to be *the type of join points it consist of.* Thus, we have 4 different pointcut types, namely *Task, Role, WorkProduct, Tool.*

A designator returns all the elements that satisfy the given criteria. Criteria for locating different types of pointcut vary from one to another, as elements with a type generally have their own features according to SPEM2.0. For example, we can specify "find all tasks taking x as an input work product". But specify "find all roles taking x as an input work product" is meaningless, which should be "find all roles responsible for work product x".

Therefore, formally we define each designator as a function, which accepts given criteria as input parameters, and returns a set of join points which satisfy the input criteria as the result pointcut. Criteria are represented by a group of two-tuple (f,v). For every (f,v), f denotes one certain field of a join point, and v specifies the target value to match. Fields of a join point are specified according to corresponding definition in SPEM2.0. We also constrain the type of v by presenting the set of (f, v.type) in each criteria definition.

tasks_ designators: tasks_criteria → {e | e.type = *Task*}

tasks_criteria = {(f,v) | (f, v.type) ∈ {*(name, string), (description, string), (step, Step), (performer, Role), (input, WorkProduct), (output, WorkProduct), (used tool, Tool)*}}

roles_designators: roles_criteria → {e | e.type = *Role*}

roles_criteria = {(f,v) | (f, v.type) ∈{*(name, string), (description, string), (skill, string), (perform, Task), (responsible for, WorkProduct)*}}

workproducts_designators: workproducts_criteria → {e | e.type = *WorkProduct*}

workproducts_criteria = {(f,v) | (f, v.type) ∈{*(name, string), (description, string), (author, Role), (source, WorkProduct) ,(target, WorkProduct)*}} tools_designators: tools_criteria → {e | e.type = *WorkProduct*}

tools_criteria = {(f,v) | (f, v.type) ∈ {*(name, string), (description, string)*}}

For the locating logic, we define:

For ∀d ∈ {tasks_designators, roles_designators, workproducts_designators, tools_designators },

$\forall e \in d((f_1, v_1), (f_2, v_2), ...(f_n, v_n))$, we have $e.f_i = v_i$, $i \in [1, n]$

For instance, find_tasks(*(performer,OpenUP::developer), (input,OpenUP::usecase)*) returns all the tasks performed by the role developer and leverage usecase as an input work product.

Advice. Advice defines which operations should be taken. An operation is represented as (a,p), where a denotes the action to be taken, and p denotes the parameter for the action. Formally, we can define Advice as:

Advice = {(a,p) | (a, p.type) ∈ *{(add_attribute, string), (add_role, Role), (add_workproduct, WorkProduct), (add_tool, Tool), (add_task_before, Task), (add_task_after, Task), (add_unordered_task,Task)}*}

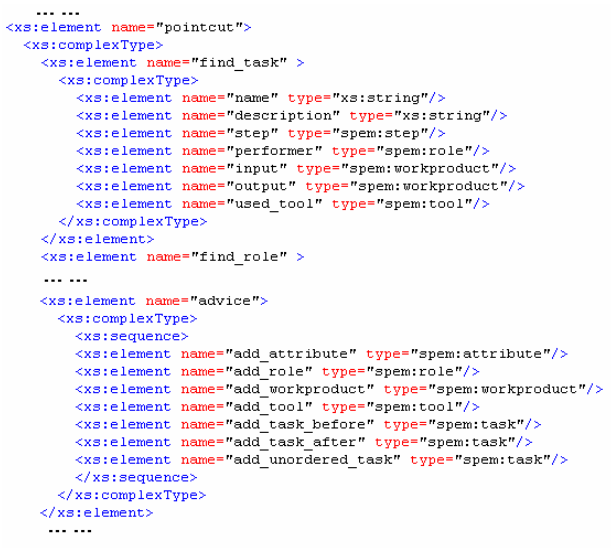
It is notable that each operation has its corresponding types of join points on which it can work. For example, casting *add_workproduct on a task* is reasonable, while casting *add_workproduct on a tool* is meaningless in SPEM2.0. Such an observation suggests that there exist inherent correspondence constrains between a pointcut and its advice. We summarize the 19 meaningful correspondence cases in Table 1.

Table 1. Meaningful correspondence cases between a pointcut type and operations in its advice

Pointcut Type	Operation	Meaning
Task	(add_attribute,x)	Add a new attribute x for an existing task
Task	(add_role,x)	Relate a new role x with anexisting task
Task	(add_workproduct, x)	Relate a new work product x with an existing task
Task	(add_tool, x)	Relate a new tool x with an existing task
Task	(add_task_before, x)	Add a new task x before an existing task
Task	(add_task_after, x)	Add a new task x after an existing task
Task	(add_unordered_task, x)	Add a new unordered task x into an existing task
Role	(add_attribute,x)	Add a new attribute x for an existing role
Role	(add_task, x)	Relate a new role x with an existing role
Role	(add_workproduct, x)	Relate a new work product x with an existing role
WorkProduct	(add_attribute,x)	Add a new attribute x for an existing work product
WorkProduct	(add_task, x)	Relate a new task x with an existing work product
WorkProduct	(add_role, x)	Relate a new role x with existing work product
WorkProduct	(add_ workproduct, x)	Relate a new work product x with an existing work product
Tool	(add_attribute,x)	Add a new attribute x for an existing tool
Tool	(add_task, x)	Relate a new task x with an existing tool

2.2 Describing a Process Aspect

Until now, we have process aspect formally defined. However, in practice, users need an easy and clear way of describing a process aspect. Therefore, we provide a XML Schema for process aspect defined above. Users can use labels defined in our schema to describe a process aspect. We present part of the schema in Fig. 2. (A complete schema is available at http://process.seforge.org/aspect/schema.xml)



```
    ... ...
<xs:element name="pointcut">
  <xs:complexType>
    <xs:element name="find_task" >
      <xs:complexType>
        <xs:element name="name" type="xs:string"/>
        <xs:element name="description" type="xs:string"/>
        <xs:element name="step" type="spem:step"/>
        <xs:element name="performer" type="spem:role"/>
        <xs:element name="input" type="spem:workproduct"/>
        <xs:element name="output" type="spem:workproduct"/>
        <xs:element name="used_tool" type="spem:tool"/>
      </xs:complexType>
    </xs:element>
    <xs:element name="find_role" >
     ... ...
    <xs:element name="advice">
      <xs:complexType>
        <xs:sequence>
        <xs:element name="add_attribute" type="spem:attribute"/>
        <xs:element name="add_role" type="spem:role"/>
        <xs:element name="add_workproduct" type="spem:workproduct"/>
        <xs:element name="add_tool" type="spem:tool"/>
        <xs:element name="add_task_before" type="spem:task"/>
        <xs:element name="add_task_after" type="spem:task"/>
        <xs:element name="add_unordered_task" type="spem:task"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
     ... ...
```

Fig. 2. An abstract object diagram for mapping pattern P1. Note *plugin element A'* has an extra *attributeX* compared to *based process element A*.

3 Weaving Process Aspect into SPEM-Based Processes

We choose to leverage the VariabilityElement class (in the method plugin package of SPEM2.0) to serve as the infrastructure for weaving process aspect. In Fig.3., *based process element A* and *plugin element A'* are both objects of class VariabilityElement. The link from *plugin element A'* to *based process element A* is established by the attribute *varialibilityBasedOnElement* of *plugin element A'*, the attribute *varialibilityType* of *plugin element A'* specifies using the <<contribute>> semantic to cast appending changes on *based process element A*. In such a relation, *based process element A* corresponds to a pointcut in process aspect, while for *plugin element A', its* extra attributes and relations to other elements in the same method plugin package constitute the advice.

In particular, in order to map the 19 meaningful correspondence cases (summarized in section 2) to SPEM2.0's method plugin mechanism, we classify them into two categories, and provide each category with a mapping pattern.

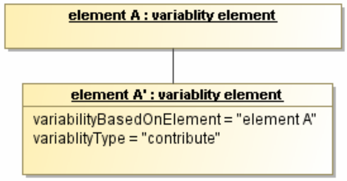

element A : variablity element
element A' : variablity element
variabilityBasedOnElement = "element A"
variablityType = "contribute"

Fig. 3. Leverage VariabilityElement in SPEM2.0 to express the weaving semantics

Mapping Pattern 1 (MP1): adding new attribute x for a task, role, work product or tool in the existing process. Implementing MP1 via SPEM2.0 Method Plugin mechanism can be done as follows:

1) Create an element A' in the method-plugin package for advice, A' have an extra attributeX, A and A' have the same type.

2) Create a <<contribute>> relation from A' to A.

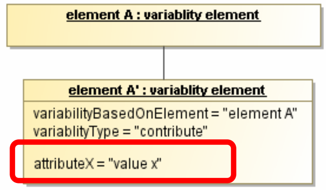

element A : variablity element
element A' : variablity element
variabilityBasedOnElement = "element A"
variablityType = "contribute"
attributeX = "value x"

Fig. 4. An abstract object diagram for MP1. Note *plugin element A'* has an extra *attributeX* compared to *based process element A*.

MP1 can be used to implement 4 correspondence cases in Table 1, depicted as follows:

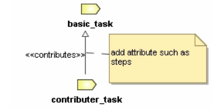
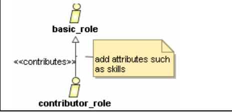
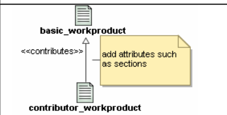
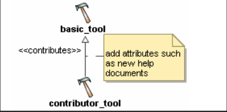
Add new attribute for existing task	Add new attribute for existing role	Add new attribute for existing work product	Add new attribute for existing tool
			

Fig. 5. Map 4 correspondence cases to SPEM2.0 by pattern MP1

Mapping Pattern (MP2): relate new element B with an existing element A in the existing process. Implementing MP2 via SPEM2.0 Method Plugin mechanism can be done as follows:

1) Create element A' and element B in the method-plugin package for advice, A and A' have the same type.

2) Relate A' and B.

3) Create a <<contribute>> relation from A' to A.

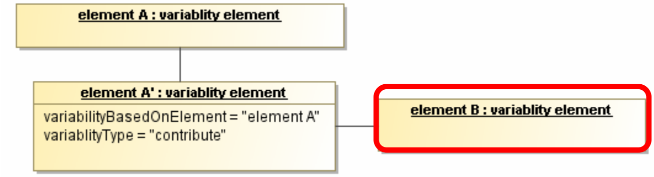

element A : variablity element
element A' : variablity element
variabilityBasedOnElement = "element A"
variablityType = "contribute"
element B : variablity element

Fig. 6. An abstract object diagram for MP2. Note *plugin element A'* has no extra attribute but an extra *relation* to *element B* compared to *based process element A*.

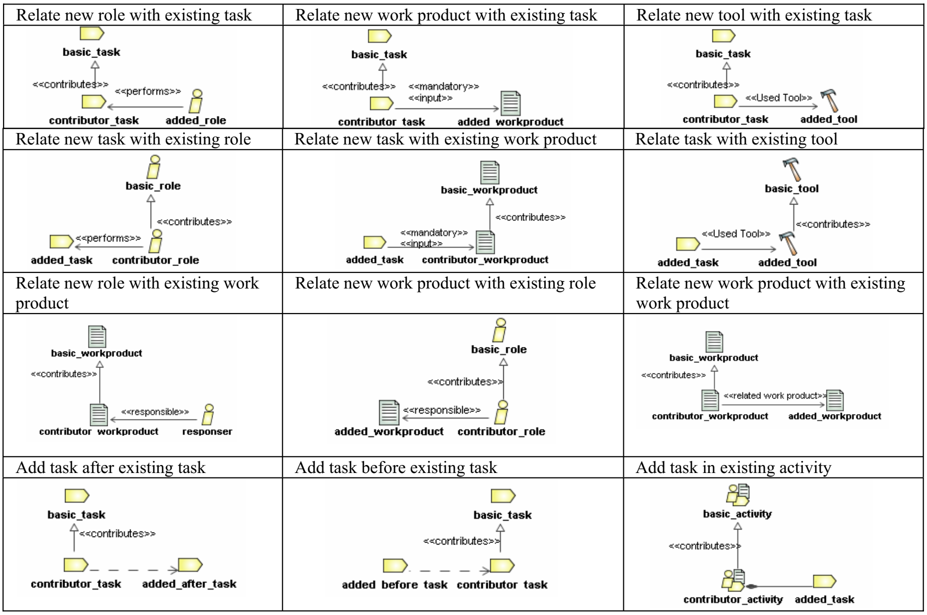


Fig. 7. Map 12 correspondence cases to SPEM2.0 by MP2

Following MP1 and MP2, we can map actions in an advice to standard SPEM2.0 method-plugin infrastructure, which can be understood and interpreted by tools supporting SPEM2.0. In such a way, an aspect can be weaved into existing processes.

However, the resulting model is redundant. We can see a simple example as follows:

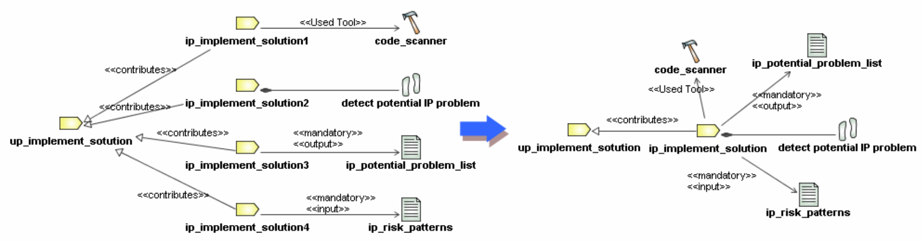


Fig. 8. The model to the left results from applying mapping pattern MP2 for 4 different actions in an advice. It can be simplified as the model to the right, with the same semantics.

Such redundancies cause unnecessary complexity in the result model, making it hard to understand, especially when the content of an aspect is relatively complex. Therefore, we further simplify the implementation result by merging redundant elements. The merge algorithm is presented as follow:

Merge Algorithm

Let `E`: **list of elements in the** `aspect-method-plugin-package`
Let `e.contributee`: **the element pointed by** `e` **with a** `'contribute'` **relation**
Let `e.addProperty(p)`: **adding property** `p` **into** `e`
Let `relation(e1,e2)`: **relation between** `e1` **and** `e2`
Let `e1.addRelatedElement(e2,r)`: **adding relation** `r` **between** `e1` **and** `e2`
Let `E.deleteElement(e)`: **deleting element** `e` **itself and all the links from** `e` **or to** `e`

```
1  For each e1 of E
2     For each e2 of E
3        If (e1 ≠ e2) and (e1.contributee = e2.contributee)
4           For each p of e2.properties
5              e1.addProperty(p)
6           For each re of e2.relatedElements
7              e1.addRelatedElement(re, relation(e2,re))
8           E.deleteElement(e2)
```

Let E': list of elements in the aspect-method-plugin-package after Merge.

Proof 1: Semantics of E equal Semantics of E'

For ∀ property p contributed from E to M, let p □ x, x is an element of E.During Merge,

(a) If x acts as e1 in s3, obviously s4 to s8 do not touch p;

(b) If x acts as e2 in s3, s5 will add p into e1 and e1.contributee M, p is still contributed from E' to M;

(c) If x does not satisfy s3, obviously there is no other place in Merge that changes p

So, p is still contributed from E' to M

In the same way, we can have:

for ∀ related element re contributed from E to M, re is still contributed from E' to M

Meanwhile, E′ does not introduce any new property or related element that does not belong to E

As a result, Semantics of E equal Semantics of E'

There are two types of elements in E: some that directly contribute to the original process (let they be contributors), and others that act as related elements of contributors (let they be related_elements). If we define the simplicity of E as number of contributors + number of related elements, we can proof that E' has the least simplicity.

Proof 2: E' has the least simplicity

Proof 2: E' has the least simplicity

In E', the number of contributor = the number of contributee, which is the possible least. Otherwise there will exist one contribute without any contributor, which will change the semantics of the whole Aspect.

Meanwhile, s7 and s8 do not change the number of related_elements, so the number of related_elements stays unchanged after Merge.

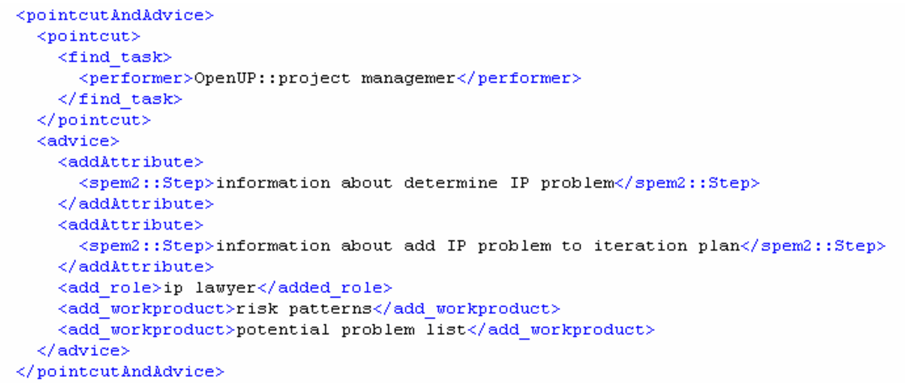
Therefore, we can get that the sum of contributor number and related_elements number, which is the simplicity defined above, has the least possible value in E'.

That is to say, E' has the least simplicity

4 An example

In this section, we take IP concern which has been discussed in section 1, as an example to illustrate how to model IP concern as an aspect, and weave it into OpenUP.

First, consider and extract changes introduced by adding IP concern. As is depicted in section 1, we group changes as follows:



```
<pointcutAndAdvice>
  <pointcut>
    <find_task>
      <performer>OpenUP::project managemer</performer>
    </find_task>
  </pointcut>
  <advice>
    <addAttribute>
      <spem2::Step>information about determine IP problem</spem2::Step>
    </addAttribute>
    <addAttribute>
      <spem2::Step>information about add IP problem to iteration plan</spem2::Step>
    </addAttribute>
    <add_role>ip lawyer</added_role>
    <add_workproduct>risk patterns</add_workproduct>
    <add_workproduct>potential problem list</add_workproduct>
  </advice>
</pointcutAndAdvice>
```

Fig. 9. A snippet of resulting model, illustrating change 1. The complete version is available at *http://process.seforge.org/aspect/ip.xml*

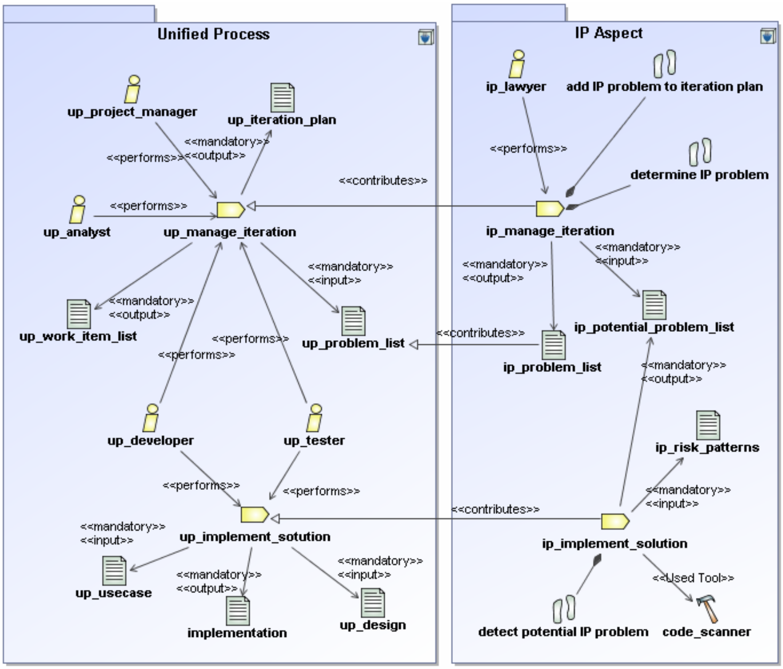


Fig. 10. Final implementation of IP aspect

a) Add two new steps *add IP problems to iteration plan* and *determine IP problem*, two new work products *problem_list* and *potential_problem_list* and a new role *lawye*r to corresponding tasks performed by existing role *project_manager*.

b) Add a new step *detect potential IP problems*, two new work products *risk_pattern* and *potential_problem_list* and a new tool *code_scanner* to corresponding tasks producing existing work product *implementation*.

c) Add a new section *ip_problem_list* into the existing work product *problem_list.*

Then, use aspect to model these changes. A snippet of resulting model is presented in Fig, 9.

Finally, after leveraging the two mapping patterns and merge redundant elements, we can get final implementation the for IP aspect, as depicted in Fig. 10. Now, the IP concern has been handled.

5 Related Work

There have been efforts on applying aspect in software processes. In [3], a general analysis on aspect-orientation's implications for software process is presented. Our work can be seen as a concrete step towards the direction proposed by [3].

There are also other concrete steps aiming at applying aspect to better support software processes [10] [11]. In [10], certain policies are defined as aspects, and weaved into a process-centered software engineering environment (PCE), so that conformance of these policies is ensured during execution of a process. [11] treats the 12 practices proposed by XP as aspects, and weave them into Eclipse to embody XP during development. Unlike these works, we focus on using aspect to handle concerns crosscutting different modules of process models, as a way of better structuring the rich content in process models.

Aspects are applied in business process as well. In [12] and [13], business rules are modeled as aspects. [14] and [15] propose aspect-oriented modeling of business processes. Moreover, an aspect-oriented workflow language, AO4BEPL, is defined in [16].

6 Conclusion

In this paper, we propose introducing process aspect to handle crosscutting concerns during process improvement. Applying process aspect facilitates understanding, managing, and reusing implementations of crosscutting concerns in process model. Then, we propose an aspect model and its weaving method in SPEM-based processes. The model together with weaving method, bridge the gap from concerns to SPEM infrastructures.

Our next step will be providing strong tool support for process aspect, allowing users to define an aspect in a visual way and automatically weave into SPEM-based processes. Currently, the development is ongoing, and we plan to contribute the tool as an open source plug-in for Eclipse Process Framework.

We have been collecting and organizing crosscutting concerns and their implementations for a long time, aiming to set up a process aspect library. According to process aspects in hand, most of them ask for adding operation, while some requires deleting and replacing elements in existing process models, which may cause confliction among several concerns. We will study these open issues in the future.

Acknowledgement

This work is funded by the National Basic Research Program of China (973) under Grant No. 2009CB320703, the High-Tech Research and Development Program of China under Grant No. 2007AA010301 and the Science Fund for Creative Research Groups of China under Grant No. 60821003.

References

1. Object Management Group, Inc. Software Process Engineering Metamodel (SPEM) 2.0 (April 2008)
2. ISO/IEC TR 15504–2:1998 Information technology – Software process assessment– Part2: A reference model for processes and process capability
3. `http://epf.eclipse.org/wikis/openup/`
4. Sutton Jr., S.M.: Aspect-Oriented Software Development and Software Process. In: ISPW 2005, pp. 177–191 (2005)
5. Sutton, O.L.J.: Product families and process families. In: SPW 1996 (1996)
6. Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., Griswold, W.G.: An overview of aspectJ. In: Knudsen, J.L. (ed.) ECOOP 2001. LNCS, vol. 2072, p. 327. Springer, Heidelberg (2001)
7. Cass, A.G., Lerner, B.S., Sutton Jr., S.M., McCall, E.K., Wise, A., Osterweil, L.J.: Little-JIL/Juliette: a process definition language and interpreter. In: ICSE 2000, Limerick, Ireland (2000)
8. `http://www.eclipse.org/epf/`
9. `http://www.dmrconseil.ca/en/Products/Macroscope/`
10. Reis, R.Q., Lima Reis, C.A.: Towards an Aspect-Oriented Approach to Improve the Reusability of Software Process Models. In: Proceedings of the IWEA, New York
11. Mishali, O., Katz, S.: Using aspects to support the software process: XP over Eclipse. In: Proceedings of the 5th international conference on Aspect-oriented software development, March 20-24, 2006, Bonn, Germany (2006)
12. Tarr, P., Ossher, H., Sutton Jr., S.: Hyper/J: Multi-Dimensional Separation of Concerns for Java, Tutorial (2001)
13. AOP for Business Rules (2003), `http://ssel.vub.ac.be/br/index.php`
14. Odgers, B., Thompson, S.: Aspect-Oriented Process Engineering (ASOPE). In: Moreira, M.D, Demeyer, S. (eds.) ECOOP 1999 Workshops. LNCS, vol. 1743. Springer, Heidelberg (1999)
15. Zhu, J.: Personnel communication. IBM Research (2005)
16. Charfi, A., Mezini, M.: Aspect-oriented workflow languages. In: Meersman, R., Tari, Z. (eds.) OTM 2006. LNCS, vol. 4275, pp. 183–200. Springer, Heidelberg (2006)

Stochastic Process Algebra Based Software Process Simulation Modeling*

Jian Zhai[1,3], Qiusong Yang[1], Feng Su[1,3], Junchao Xiao[1],
Qing Wang[1], and Mingshu Li[1,2]

[1] Laboratory for Internet Software Technologies, Institute of Software,
The Chinese Academy of Sciences, Beijing, China, 100190
[2] Key Laboratory for Computer Science, The Chinese Academy of Sciences,
Beijing, China, 100190
[3] Graduate University of Chinese Academy of Science,
Beijing, China, 100049
{zhaijian,qiusong_yang,sufeng,xiaojunchao,wq,
mingshu}@itechs.iscas.ac.cn

Abstract. In recent years, simulation techniques that have been widely used in many other disciplines are being increasingly used in analyzing software processes. However, researchers from software process simulation community tend to build a separate new model with various technologies from traditional software models. This is partially because that software process simulation might take a completely different approach to describe a process under certain circumstances, for instance, a process being modeled as an overall system. Another reason is that traditional software process modeling methods can not provide simulation functions. The gap between traditional software process modeling and software process simulation modeling confined a wider application of simulation approach in the software engineering community. In this paper, we show the possibility of a simulation model being automatically derived from a traditional descriptive process model and thus one does not necessarily need to build a separate simulation model. By doing so, all information in the descriptive models can be reused.

1 Introduction

Over the past few decades, software organizations have always been facing the problem that the cost and schedule of software projects are often overrun and the quality of software product does not meet consumers' expectation. A lot of research has been conducted by people from both the industry and academia to solve the problem. Among those attempts, the research based on software process techniques, such as software process modeling and software process simulation, is one of the most viable approaches.

The principle that the quality of a software product highly depends on the quality of the software development process has been widely accepted. Software process is

* Supported by the National Natural Science Foundation of China under grant Nos. 90718042, the Hi-Tech Research and Development Program (863 Program) of China under grant No.2007AA010303, 2007AA01Z186, as well as the National Basic Research Program (973 program) under grant No. 2007CB310802.

Q. Wang et al. (Eds.): ICSP 2009, LNCS 5543, pp. 136–147, 2009.

defined as a set of partially ordered process steps, with sets of related artifacts, human and computerized resources, organizational structures and constraints, intended to produce and maintain the requested software deliverables [1]. In order to enhance understanding on performed processes and provide direct guidance on actual software development, software process modeling has been focused on the representation, analysis and automatic execution of processes, which has become an active research area of the academia.

Software process modeling is an extensive and complex domain and many languages and technologies have their roots in computer science and related domains [2]. For instance, dozens of modeling methods have been developed based on various technologies, ranging from software design notations and methodologies, formal methods or languages, to multi-agents or rules-based intelligent systems [3,4]. Besides documenting processes, another goal of software process modeling is to provide process designers with a leverage to analyze process models and thus enhance understanding of such process being modeled.

In recent years, simulation techniques that have been widely used in many other disciplines are being increasingly used in analyzing software processes. It is mainly because the validation of a research initiative on software process in real context can hardly be realized, due to the ever changing external environment and cost concerns. Compared with traditional analysis approaches, both qualitative and quantitative results can be achieved by simulation techniques. In addition, a process designer or project manager can see the impact of different alternatives. As a result, simulation techniques have been used to address a variety of issues in software process modeling ranging from the strategic management of software development, supporting process improvements to software project management training [5]. A summary of the state-of-the-art of software process simulation modeling can be found in [6].

However, researchers from software process simulation community tend to build a totally new model based on various technologies such as system dynamics (SD) and discrete event simulation (DES), rather than reusing traditional descriptive models, i.e. stating how a software should be developed. It is partially because that software process simulation might need to take a different approach to describe the software process under certain circumstances. For instance, a process is described as an overall system and its behaviors are described by a set of external parameters that continuously vary over time [6]. Another dimension of the problem is that traditional software process modeling methods can not provide analysis on software process simulation. The gap between traditional software process modeling and software process simulation modeling confined a wider application of the modeling and simulation approach in the software engineering community.

In this paper, we demonstrate the possibility that a simulation model can be automatically derived from a traditional descriptive process model. The strength of the approach is that one does not need to build a separate simulation model and all the information in the descriptive models can be reused. The basic approach is to introduce random variables into the language TRISO/ML [7] to describe the uncertainties of a software development project. The extended language s-TRISO/ML is a graphical process modeling language with rigorous semantics in π-calculus [8]. Thus a simulation model in stochastic process algebras can be produced from a graphical process model in TRISO/ML. Various kinds of analysis can then be conducted on the simulation model.

The rest of this paper is organized as follows: Section 2 introduces s-TRISO/ML as well as a general introduction of stochastic polyadic π-calculus. Section 3 provides an experiment on the modeling and the simulation in s-TRISO/ML. Section 4 and Section 5 review the related work and make a conclusion respectively.

2 s-TRISO/ML: A Modeling Language with Stochastic Information

To describe the stochastic attributes of a software process and simulate the performance of a process model, a new software process simulation modeling language named s-TRISO/ML is used. The language includes two main parts: the definition and the mapping rules. The foundation of s-TRISO/ML is stochastic polyadic π-calculus. Software process modeled by s-TRISO/ML can be used to govern as well as to simulate the actual software process execution.

2.1 Stochastic Polyadic π-Calculus

Stochastic process algebra (SPA) was first proposed as a tool for performance and dependability modeling in 1990 [9]. It was introduced as an extension of classical process algebra, such as CCS [10], CSP [11] or π-calculus [8], with timing information aimed at facilitating the integration of functional design and quantitative analysis of computer systems. In fact, stochastic π-calculus is popular in biological or biochemical system modeling and simulation [12,13].

Definition 1 (Stochastic Polyadic π-calculus). *The syntax of the stochastic polyadic π-calculus is given in the following BNF equations:*

$$\begin{aligned} P &:= M \mid P|P' \mid (\nu z)P \mid !P \\ M &:= \mathbf{0} \mid \pi.P \mid M + M' \\ \pi &:= \overline{x}\langle\widetilde{y}\rangle \mid x(\widetilde{z}) \mid (\tau, r) \mid [x = y]\pi \end{aligned}$$

Briefly, $\mathbf{0}$ is *inaction* that represents a process which can do nothing; the *prefix* $\pi.P$ can perform the output, input, unobserved action or match action, thereby evolving into P; the *sum* $M + M'$ offers the *choice* of M or M'; the composition $P|P'$ – "P par Q" – places the two processes together and they will be concurrently activated and act independently, but they can also communicate; the $(\nu x)P$ – "new x in P"– restricts the use of the name x to P and it declares a new unique name x, distinct from all external names, for use in P. As for the output and input prefixes, the intended interpretations of them are that $\overline{x}\langle\widetilde{y}\rangle.P$ can send the tuple $\widetilde{y}$ via the co-name of x and continue as P, and $x(\widetilde{z}).Q$ can receive a tuple $\widetilde{y}$ via the name x and continue as $Q\{\widetilde{y}/\widetilde{z}\}$. The *unobserved prefix* $(\tau, r).P$ can evolve invisibly to P. In (τ, r), τ is the internal action of a process and invisible to the external viewer; $r \in (0, +\infty)$ is a parameter of the negative exponential distribution governing its duration, and it specifies how long it will take to complete the action. $+\infty$ can be expressed by the notation $\top$, which means the activity can be finished instantaneously, and 0 means that it would be finished in a large enough time, or it cannot be finished. The *match prefix* $[x = y]\pi.P$ can evolve as $\pi.P$ if x and y have the same name, otherwise the process will act as $\mathbf{0}$.

2.2 s-TRISO/ML

The software process simulation model s-TRISO/ML (stochastic-TRidimensional Integrated SOftware development model/Modelling Language) is an extension of TRISO/ ML [7], which is a polyadic π-calculus based graphical software process modeling language, and is proposed for supporting the TRISO Model advocated in [14]. The primary element of s-TRISO/ML is the process activity, connected by the temporal relation operators. The language describes a process as an activity hierarchy and it provides powerful abstractions of control flow, data dependency, and resource usage in software processes. More importantly, this language provides a group of mapping rules in order to transform every graphical process model in s-TRISO/ML to a series of stochastic polyadic π-calculus expressions in a mechanical way. The mapping rules ensure that the semantics of the graphical model and the stochastic polyadic π-calculus expressions be consistent. The definition of a software process in s-TRISO/ML is the same as that in TRISO/ML, which can be found in [7].

In s-TRISO/ML, non-terminal activities, which are divided into sub-activities in a model, and terminal activities, which are leaf nodes in a model, are separated strictly. In a s-TRISO/ML model, only terminal activities are actual activities executed by human resource or other agents, and they are operated in a certain duration. Terminal activities can be finished on scheduled time or can be delayed. These activities need to be expressed in a stochastic way. In opposite, the non-terminal activities in the model does not happen in actual software processes. The function of these activities is to indicate the execution sequence or the relationship among the actual activities or terminal activities.

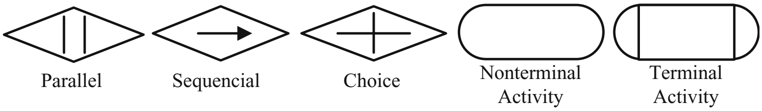


Fig. 1. Graphical elements of s-TRISO/ML

Fig. 1 is the graphical elements of s-TRISO/ML. The diamonds stand for the execution sequence of activities, which include sequential, parallel and choice. The ellipse stands for the non-terminal activities. The ellipse with rectangle inside stands for the terminal activities. The caption of an activity can be labeled inside the frame. For a non-terminal activity, the caption should be its identifier, and for a terminal activity, its caption should be a couple (ID,r), where ID is the identifier of the activity and r implies for the possible time that the activity may last. The graphical elements in a model connect each other by links. Attributes can be labeled besides the graphical elements, including nodes and links.

2.3 Mapping Rules

In this section, the rules for mapping a software process modeled in s-TRISO/ML onto the stochastic polyadic π-calculus processes are provided. All elements of s-TRISO/ML defined above are covered by the mapping rules. With these rules, the interpreting procedure becomes rather straightforward and mechanical. Furthermore, the transformed s-TRISO/ML can be simulated in stochastic process algebra tools as the input.

Rule 1. For an actor with the unique identifier ac, the stochastic polyadic π-calculus process for it has the following form:

$$A_{ac} \overset{def}{=} assign_{ac}(start, end).\overline{start}.end.A_{ac}|A_{ac}$$

The process A_{ac} waits on channel $assign_{ac}$ for channels $start$ and end. When the actor wants to begin to perform the activity, the process will send an empty message through the received $start$ channel. The actor will receive an empty message from the end channel when all the sub-activities of the assigned activity have been finished. Having accomplished an activity, the actor will be ready for another task.

Rule 2. For an activity $a \in \mathcal{A}$, it receives $\{b_{11}, \cdots, b_{1m}\}, \cdots, \{b_{l1}, \cdots, b_{ln}\}$ from the channels $\{chi_1, \cdots, chi_l\}$ and $\{p_1, \cdots, p_u\}$ from the channel $ex_{a_p\_a}$, sends $\{c_{11}, \cdots, c_{1s}\}, \cdots, \{c_{r1}, \cdots, c_{rt}\}$ through the channels $\{cho_1, \cdots, cho_r\}$, and returns $\{q_1, \cdots, q_v\}$ to its parent a_p through the channel $ex_{a\_a_p}$. Then, the stochastic polyadic π-calculus process A_a for the activity is:

$$\begin{aligned}
A_a &\overset{def}{=} (\nu\ i_1, \cdots, i_l, i_o)(I_{a_s}\langle i_1, \cdots, i_l\rangle\ |E_a\langle i_1, \cdots, i_l, i_o\rangle \mid O_a\langle i_o\rangle) \\
I_a &\overset{def}{=} (i_1, \cdots, i_l).chi_1(b_{11}, \cdots, b_{1m}).\bar{i}_1\langle b_{11}, \cdots, b_{1m}\rangle| \cdots | \\
&\quad chi_l(b_{l1}, \cdots, b_{ln}).\bar{i}_l\langle b_{l1}, \cdots, b_{ln}\rangle.ex_{a_p\_a}(p_1, \cdots, p_v) \\
O_a &\overset{def}{=} (i_o).i_o(c_{11}, \cdots, c_{1s}, \cdots, c_{r1}, \cdots, c_{rt}, q_1, \cdots, q_v).\overline{cho_1}\langle c_{r1}, \cdots, c_{rs}\rangle. \\
&\quad \cdots .\overline{cho_r}\langle c_{r1}, \cdots, c_{rt}\rangle.\overline{ex_{a\_a_p}}\langle q_1, \cdots, q_v\rangle
\end{aligned}$$

The process A_a is the concurrent combination of I_a, E_a, and O_a. The process I_a receives data from prescribed channels and the channel connecting to its parent activity then sends the received data to the process E_a through private channels. Acting as a relay station, I_a ensures that the communication on any input channel can be carried out immediately and deadlocks will not arise as a result of the mismatch between the order of input and the order of manipulation. When an activity and its sub-activities are completed, it will output data to other activities and its parent activity, as shown by the process O_a. The execution of the activity is modeled by the process E_a, whose definition is given by the following rules.

Rule 3. For a non-terminal activity $a \in \mathcal{A}$, it is refined to w sequential activities, $a_1, \cdots, a_w$. Each sub-activity may specify the information exchanges with its parent. For example, the wth sub-activity will receive $\{p_{w1}, \cdots, p_{wj}\}$ from the activity a and returns $\{q_{w1}, \cdots, q_{wk}\}$. The activity will be assigned to the actor with the unique identifier ac. Then the E_a process for the activity a is:

$$\begin{aligned} E_a = {} & (i_1, \cdots, i_l, i_o).i_1(b_{11}, \cdots, b_{1m}). \cdots .i_l(b_{l1}, \cdots, b_{ln}).ex_{a_p\_a}(p_1, \cdots, p_u). \\ & trigger_a.\overline{assign_{ac}}\langle start_a, end_a\rangle.start_a.\overline{ex_{a\_a1}}\langle p_{11}, \cdots, p_{1h}\rangle.\overline{trigger_{a1}}. \\ & ex_{a1\_a}(q_{11}, \cdots, q_{1i}).triggered_{a1}. \cdots .\overline{ex_{a\_aw}}\langle p_{w1}, \cdots, p_{wj}\rangle.\overline{trigger_{aw}}. \\ & ex_{aw\_a}(q_{w1}, \cdots, q_{wk}).triggered_{aw}.\overline{triggered_a}.\overline{end_a}. \\ & \overline{i_o}\langle c_{11}, \cdots, c_{1s}, \cdots, c_{r1}, \cdots, c_{rt}, q_1, \cdots, q_v\rangle \end{aligned}$$

In this rule, each output variable must be bounded by certain input prefix. As an example, for $\forall t,\ 1 \leq t \leq w$: $\{p_{t1}, \cdots, p_{t.}\}$ is a subset of $\{b_{11}, \cdots, b_{1m}\} \cup \cdots \cup \{b_{l1}, \cdots, b_{ln}\} \cup \{p_1, \cdots, p_u\} \cup \{q_{11}, \cdots, q_{1i}\} \cup \cdots \cup \{q_{(t-1)1}, \cdots, q_{(t-1).}\}$. All of the following rules are also subject to this constraint. Firstly, the process E_a withdraws the relayed input from the process I_a. Then, the activity a is assigned to the prescribed actor when the activity is triggered by its parent activity. The actual execution of the activity will not begin until the actor decides to do so. As the activity a is a non-terminal node, the process E_a then sequentially triggers the execution of its sub-activities. When the activity is completed, it will notify its parent and release the assigned actor. At last, the obtained data will be sent to the process O_a for output.

Rule 4. For a non-terminal activity $a \in \mathcal{A}$, it is decomposed into w concurrently combined activities. Then the E_a process for the activity a is:

$$\begin{aligned} E_a = {} & (i, q, i_1, \cdots, i_l, i_o).(\nu k_{a1}, \cdots, k_{aw})i_1(b_{11}, \cdots, b_{1m}). \cdots .i_l(b_{l1}, \cdots, b_{ln}). \\ & ex_{a_p\_a}(p_1, \cdots, p_u).trigger_a.\overline{assign_{ac}}\langle start_a, end_a\rangle.start_a.(E_1|E_2) \\ E_1 = {} & (\overline{ex_{a\_a1}}\langle p_{11}, \cdots, p_{1h}\rangle.\overline{trigger_{a1}}.ex_{a1\_a}(q_{11}, \cdots, q_{1i}).triggered_{a1}.\overline{k_{a1}}. \\ & \overline{k_{a1}}\langle q_{11}, \cdots, q_{1i}\rangle) \,|\cdots|(\overline{ex_{a\_aw}}\langle p_{w1}, \cdots, p_{wj}\rangle.\overline{trigger_{aw}}.ex_{aw\_a}(q_{w1}, \cdots, q_{wk}). \\ & triggered_{aw}.\overline{k_{aw}}.\overline{k_{aw}}\langle q_{w1}, \cdots, q_{wk}\rangle) \\ E_2 = {} & k_{a1}.k_{a1}(q_{11}, \cdots, q_{1i}). \cdots .k_{a1}.k_{wa}(q_{w1}, \cdots, q_{wk}).\overline{triggered_a}.\overline{end_a}. \\ & \overline{i_o}\langle c_{11}, \cdots, c_{1s}, \cdots, c_{r1}, \cdots, c_{rt}, q_1, \cdots, q_v\rangle \end{aligned}$$

The process E_1 triggers the sub-activities concurrently and the E_2 process collects results from sub-activities and output them to the process O_a. The enforced synchronization on channels $k_{a1}, \cdots, k_{aw}$ ensures that the process E_2 be executed after the process E_1 even under the condition that there is no activity passing data back to the activity a.

Rule 5. For a non-terminal activity $a \in \mathcal{A}$, it is decomposed into w sub-activities, which are combined together through the choice operator. Then the E_a process for the activity a is:

$$\begin{aligned} E_a = {} & (i, q, i_1, \cdots, i_l, i_o).(\nu k)i_1(b_{11}, \cdots, b_{1m}). \cdots .i_l(b_{l1}, \cdots, b_{ln}). \\ & ex_{a_p\_a}(p_1, \cdots, p_u).trigger_a.\overline{assign_{ac}}\langle start_a, end_a\rangle.start_a.(E_1|E_2) \\ E_1 = {} & (\overline{ex_{a\_a1}}\langle p_1, \cdots, p_h\rangle.\overline{trigger_{a1}}.ex_{a1\_a}(q_1, \cdots, q_j).triggered_{a1}.\overline{k}.\overline{k}\langle q_1, \cdots, q_j\rangle) \\ & + \cdots + (\overline{ex_{a\_aw}}\langle p_1, \cdots, p_h\rangle.\overline{trigger_{aw}}.ex_{aw\_a}(q_1, \cdots, q_j).triggered_{aw}.\overline{k}. \end{aligned}$$

$$\overline{k}\langle q_1, \cdots, q_j\rangle)$$
$$E_2 = k.k\langle q_1, \cdots, q_j\rangle.\overline{triggered_a}.\overline{end_a}.\overline{i_o}\langle c_{11}, \cdots, c_{1s}, \cdots, c_{r1}, \cdots, c_{rt}, q_1, \cdots, q_v\rangle$$

Rule 6. For a terminal activity $a \in \mathcal{A}$, it is not decomposed further. Then the process E_a for the activity a is:

$$\begin{aligned}E_a = {} & (i, q, i_1, \cdots, i_l, i_o).i_1(x_{11}, \cdots, x_{1m}).\cdots i_l(x_{l1}, \cdots, x_{ln}).ex_{a_p\_a}(p_1, \cdots, p_u).\\ & trigger_a.\overline{assign_{ac}}\langle start_a, end_a\rangle.start_a.(\tau, r).\overline{ex_{a\_ap}}\langle p_1, \cdots, p_h\rangle.\overline{triggered_a}.\\ & \overline{end_a}.\overline{i_o}\langle c_{11}, \cdots, c_{1s}, \cdots, c_{r1}, \cdots, c_{rt}, q_1, \cdots, q_v\rangle\end{aligned}$$

In the expression above, the invisible behavior (τ, r) is extremely important for this modeling approach. τ stands for the invisible behavior itself and r implies its possible operation time. This element reflects the stochastic attribute of a software process model. It's worth mentioning that there is no stochastic behavior in the mapping rules that refers to non-terminal activities, because only terminal activities are real activities in actual software process. In opposite, the non-terminal activities are all fictitious activities for indicating the execution sequence of real activities. We assume all of the non-terminal activities can be definitely finished instantaneously.

Rule 7. The software process is defined as the concurrent combination of activities and actors:

$$SP = A_{a1} \mid \cdots \mid A_{am} \mid A_{ac1} \mid \cdots \mid A_{acn}$$

To analyze or simulate the software process, sometimes an additional process modelling the environment is needed to make the system closed. The process is named Env and it is concurrently combined with the process SP. It can be simply defined as:

$$Env = \overline{trigger_{root}}.triggered_{root}$$

where $root$ denotes the root activity of a software process.

3 Experiment

In a model defined by s-TRISO/ML, we give r-value to each terminal activity, and transform the model into stochastic polyadic π-calculus expressions. Then, the expressions are inputted to PEPA [15], a stochastic process algebra simulation system, to be simulated. Different r-value of terminal activities would lead to dramatically different simulation results. If all of the r-values of terminal activities are $\top$, which means all of the activities are finished instantaneously, the model would degenerate back into a primary TRISO/ML model without any uncertainty. In software engineering domain, a r-value will be affected by many practical factors, such as the activity scale, complexity, the productivity of the operator and so on. Different software organizations may use different practical factors to compute r-value. In this experiment, the following formula is used to generate r-value of activities:

$$r = P \times C^{-1} \times S^{-1}$$

where, P stands for the operator's productivity, which indicates how much work can be done in a day by the operator, and S stands for the scale of the activity and C stands for the complexity of the activity. For example, if the scale of a coding activity is 5KLOC and the productivity of the operator who is going to be assigned is 0.5KLOC per day, and if the complexity of the activity is 1, then the corresponding r-value should be 0.1. In practice, the factors would be calculated based on historical project experiences or expert knowledge.

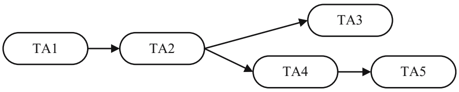


Fig. 2. A simple software process in workflow graph

Fig. 2 is a simple software process in a generic workflow graph form. In this graph, activities TA1, TA2 are sequentially executed, and so do activities TA4, TA5. Activity TA3 is parallelly executed with the branch of TA4 and TA5. TA3 and TA4 are all executed after TA2. In fact, the graph illustrates a specific waterfall development process. TA1, TA2, TA4 and TA5 are requirement, design, coding, and testing activity respectively, and they are sequentially executed. Meanwhile, TA3 is an combined activity with coding and testing activities, and it can be parallelly executed with the other coding and testing activities.

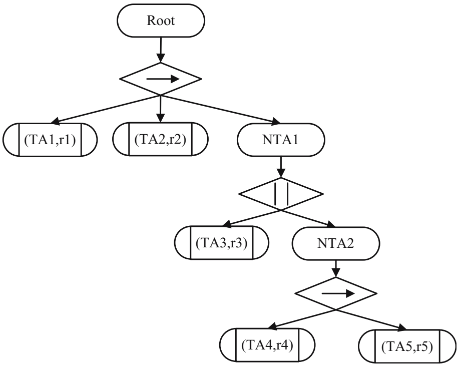


Fig. 3. The process model in s-TRISO/ML corresponding to Fig. 2

Further, the process in Fig. 2 can be expressed by an s-TRISO/ML process model, which is shown in Fig. 3. The process model starts with a non-terminal activity *Root*, and all the activities in Fig. 2 are transformed into terminal activities in the model. Along with the terminal activities, there are two non-terminal activities, NTA1 and NTA2, added in the model, and they imply the execution relationship among the terminal activities.

To simulate the process, the factors of each activity and its r-value should be instantiated. The factors used in this paper are shown in Table 1. The scale is decided by the

Table 1. The factors of each activity and the r-value of them

Activity	Type	Scale	Complexity	Productivity	r-value
TA1	requirement	20 pages	1	2 pages	0.1
TA2	design	5 pages	2	1 pages	0.1
TA3	coding/testing	10 KLOC	0.5	0.5 KLOC	0.1
TA4	coding	2 KLOC	0.5	0.5 KLOC	0.5
TA5	testing	1 KLOC	1	1 KLOC	1

activity itself, and the productivity depends on the assigned operator's ability. The type of an activity is considered in order to adjust the complexity.

This model is transformed into stochastic π-calculus expressions and simulated by PEPA for 110 times. Fig. 4 shows the operational duration of each activity in the simulation. The horizontal coordinate axis indicates the simulation times, which is from 0 to 110, and the vertical coordinate axis indicates the lasting duration of each activity. Each curve in the figure indicates an activity, and one activity's lasting duration in each simulation can be found in the figure. To average the value of each activity's duration during the simulation, the expected operational duration under the assigned r-value can be generated. Further, the result of the average value can be transformed into a Gantt Chart as an expected schedule of the process. Fig. 5 is the expected Gantt Chart, and it shows the operational duration of the whole process would be 31 days.

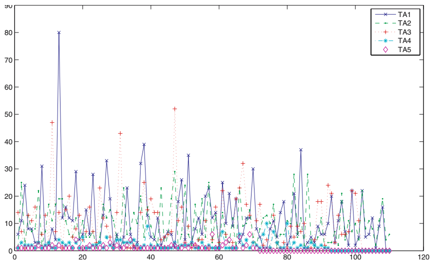


Fig. 4. The lasting duration of each activity

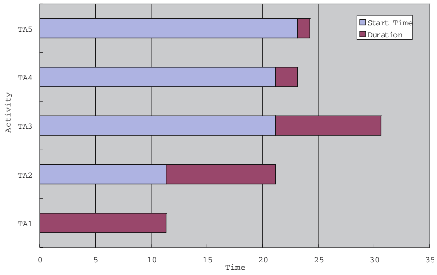


Fig. 5. The expected Gantt chart

Then, we assume that the assigned operator of TA2 is substituted by a new staff, whose productivity is half of the primary one. So the r-value of TA2 should be adjusted according to the new factor of productivity, and the new value would be 0.05. In this case, the model should be simulated again. Since the total simulation duration is fixed, the simulation time may be changed. In this new simulation, the simulation time is reduced to 88. The result is shown by Fig. 6, and the expected Gantt Chart is shown by Fig. 7. The two figures above share the same meaning with Fig 4 and Fig 5. From the Gantt Chart, it can be found that the lasting duration of TA2 improves to 24, and that in Fig. 5 is only 10. It is clear that the simulation result is sensitive to the assigned r-value.

From the simulation results, the delay probability of software process in a certain executor assignment can be detected. Fig. 8 and Fig. 9 show the total process execution duration of each simulation. The data in these figures are re-arranged to be a monotone

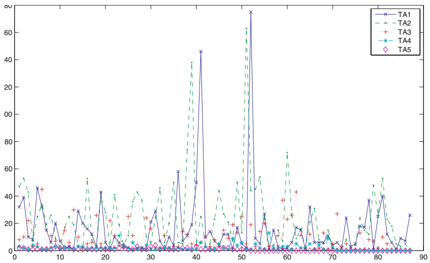


Fig. 6. The lasting duration of each activity when TA2 is reassigned

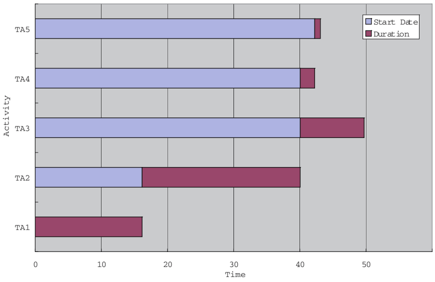


Fig. 7. The expected Gantt chart of the process when TA2 is reassigned

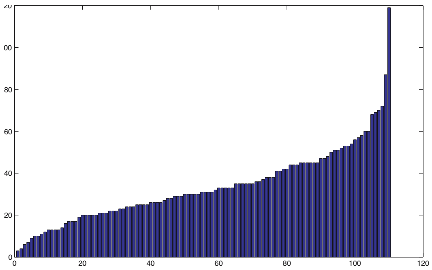


Fig. 8. The duration of each times of simulation corresponding to Fig. 4

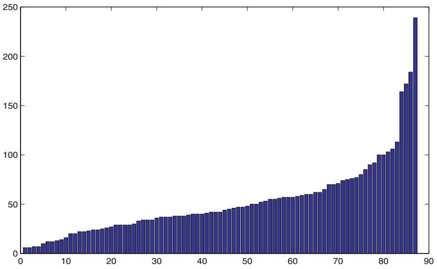


Fig. 9. The duration of each times of simulation corresponding to Fig. 6

increasing sequence. The figures are based on the two groups of data indicated by Fig. 4 and Fig. 6 respectively. If the requirement is to finish the project in 40 days, then the delay probability of the process in the two assignments are 22% and 64%. To synthesize all of the results, an interesting finding can be uncovered: in the first assignment, the expected finishing time would be 31 days, but there is still 22% probability for the process to be delayed; by contrast, in the second assignment, though the expected finishing time would be 50 days, there is still 36% probability for the process to be finished on the scheduled time.

4 Related Work

Software process simulation model is first introduced by Abdel-Hamid [16]. In 1998, a famous paper was published in the domain of software process simulation model [5], which is widely considered as a milestone in this field. Kellner et al. systemically discussed the foundational problems in software process simulation model. In the last ten years, software process simulation model receives wide attention. More than 80% researches use system dynamics approach or discrete-event simulation approach to simulate software process. System dynamics integrates quantitative dynamic models with quantitative and qualitative static models in a natural way. In contrast, discrete event models easily represent queues and capture the variation in several attributes, which can be assigned to each entity individually.

IMMoS is one of the most famous system dynamics simulation approaches, introduced by Pfahl et al. [17]. IMMoS enhances the traditional system dynamic approach by adding a component that enforces goal-orientation, and by providing a refined model. Unfortunately, the useful information learned from empirical studies is hard to be combined into the simulation result generated by this approach. In addition, it is hard for this approach to build a system dynamic model in practice. As for discrete-event simulation approach, Schriber et al. gave a general introduction [18], and Raffo et al. made further research on this approach [19]. However these researches failed to accurately capture the continuously changing variables or to elegantly represent simultaneous activities. To combine the positive aspects of the two approaches, some hybrid approaches are also proposed [20].

Stochastic process algebra based simulation is widely used in biological or chemical process simulation. Clark et al. introduced the stochastic process algebras and the well-known simulation tools PEPA [15]. As applications of this simulation approach, Bradley et al. used it to model a complex electronic voting process [21], and Clark et al. showed how the approach works in an interacted battle process [15]. It seems there is still no influential research on this simulation approach in software process modeling and predicting area.

5 Conclusion

Both of the software process modeling and the software process simulation are key techniques in software process research. Proper use of the techniques may well guide an actual software process and predict the possible performance and outcome of a software process. Unfortunately, the software process model and software process simulation model are totally separated in most situations. Software organization has to build two models for different aims in practice, which may cost extra resource. In this paper, a new software process simulation model is proposed. The model uniforms the software process model and the software process simulation model. Only one process model will be built in order to govern the actual software process execution and to simulate the software process simultaneously.

A new software process simulation model named s-TRISO/ML is introduced in the paper. Since the modeling language is a graphical language, it is easy to use for common process modeler. At same time, it has a strict formal foundation. Various process algebra simulation tools, such as PEPA in this paper, can be used to simulate the modeled process. The experiment shows that the approach may successfully simulate a modeled process and give the expected simulation result.

For the future work, an appropriate simulation tool is needed for the approach to substitute PEPA, which is not optimized for software process simulation and cannot afford the complex process model. Besides, the empirical research on the r-value in the model is needed. Many factors may affect the value. To analyze the effect of the factors and integrate them into the formula would greatly benefit the accuracy of the simulation approach.

References

1. Lonchamp, J.: A structured conceptual and terminological framework for software process engineering. In: Proceedings of the Second International Conference on the Software Process, pp. 41–53. IEEE Computer Society Press, Los Alamitos (1993)
2. Conradi, R., Jaccheri, M.L.: Process modelling languages. In: Derniame, J.C., Kaba, B.A., Wastell, D.G. (eds.) Promoter-2 1998. LNCS, vol. 1500, pp. 27–52. Springer, Heidelberg (1999)
3. Zamli, K.Z., Lee, P.A.: Taxonomy of process modeling languages. In: AICCSA 2001: Proceedings of the ACS/IEEE International Conference on Computer Systems and Applications, Washington, DC, USA, p. 435. IEEE Computer Society Press, Los Alamitos (2001)
4. Arbaoui, S., Derniame, J.-C., Oquendo, F., Verjus, H.: A comparative review of process-centered software engineering environments. Ann. Softw. Eng. 14(1-4), 311–340 (2002)
5. Kellner, M.I., Madachy, R.J., Raffo, D.M.: Software process simulation modeling: Why? what? how? Journal of Systems and Software 46(2), 91–105 (1999)
6. Zhang, H., Kitchenham, B.A., Pfahl, D.: Reflections on 10 years of software process simulation modeling: A systematic review. In: Wang, Q., Pfahl, D., Raffo, D.M. (eds.) ICSP 2008. LNCS, vol. 5007, pp. 345–356. Springer, Heidelberg (2008)
7. Yang, Q., Li, M., Wang, Q., Yang, G., Zhai, J., Li, J., Hou, L., Yang, Y.: An Algebraic Approach for Managing Inconsistencies in Software Processes. In: Wang, Q., Pfahl, D., Raffo, D.M. (eds.) ICSP 2007. LNCS, vol. 4470, pp. 121–133. Springer, Heidelberg (2007)
8. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes – part I and II. Journal of Information and Computation 100, 1–77 (1992)
9. Herzog, U.: Formal description, time and performance analysis: A framework. Technical Report 15/90, IMMD VII, Friedrich-Alexander-Universität (1990)
10. Milner, R.: A Calculus of Communicating Systems. Springer, Heidelberg (1980)
11. Hoare, C.A.R.: Communicating sequential processes. Commun. ACM 21(8), 666–677 (1978)
12. Lecca, P., Priami, C.: Cell cycle control in eukaryotes: A biospi model. Electron. Notes Theor. Comput. Sci. 180(3), 51–63 (2007)
13. Priami, C., Regev, A., Shapiro, E., Silverman, W.: Application of a stochastic name-passing calculus to representation and simulation of molecular processes. Information Processing Letters 80(1), 25–31 (2001)
14. Li, M.: Expanding the horizons of software development processes: A 3-D integrated methodology. In: Li, M., Boehm, B., Osterweil, L.J. (eds.) ISPW 2005. LNCS, vol. 3840, pp. 54–67. Springer, Heidelberg (2006)
15. Clark, A., Gilmore, S., Hillston, J., Tribastone, M.: Stochastic process algebras. In: Bernardo, M., Hillston, J. (eds.) SFM 2007. LNCS, vol. 4486, pp. 132–179. Springer, Heidelberg (2007)
16. Abdel-Hamid, T., Madnick, S.E.: Software project dynamics: an integrated approach. Prentice-Hall, Inc., Upper Saddle River (1991)
17. Pfahl, D., Ruhe, G.: Immos. a methodology for integrated measurement, modelling, and simulation (2003)
18. Schriber, T.J., Brunner, D.T.: Inside discrete-event simulation software: how it works and why it matters. In: WSC 2005: Proceedings of the 37th conference on Winter simulation, Winter Simulation Conference, pp. 167–177 (2005)
19. Raffo, D.: Combining process feedback with discrete event simulation models to support. In: Software Project Management. International Software Process Simulation Modeling Workshop (ProSim 2004), pp. 24–25 (2004)
20. Choi, K., Bae, D.-H., Kim, T.: An approach to a hybrid software process simulation using the devs formalism. Software Process: Improvement and Practice 11(4), 373–383 (2006)
21. Bradley, J.T., Gilmore, S.T.: Stochastic simulation methods applied to a secure electronic voting model. Electronic Notes in Theoretical Computer Science 151, 5–25 (2006)

Combining Aspect and Model-Driven Engineering Approaches for Software Process Modeling and Execution*

Reda Bendraou[1], Jean-Marc Jezéquél[2,3], and Franck Fleurey[4]

[1] University Pierre & Marie Curie
4, Place Jussieu, Paris F-75005, France
{firstname.lastname@lip6.fr}
[2] INRIA-Rennes Bretagne Atlantique, Campus de Beaulieu
F-35042 Rennes Cedex, France
{firstname.lastname@inria.fr}
[3] IRISA, Université Rennes 1
Campus de Beaulieu
F-35042 Rennes Cedex, France
[4] SINTEF, Oslo Franck
Fleurey@Sintef.no

Abstract. One major advantage of executable software process models is that once defined, they can be simulated, checked and validated in short incremental and iterative cycles. This also makes them a powerful asset for important process improvement decisions such as resource allocation, deadlock identification and process management. In this paper, we propose a framework that combines Aspect and Model-Driven Engineering approaches in order to ensure process modeling, simulation and execution. This framework is based upon UML4SPM, a UML2.0-based language for Software Process Modeling and Kermeta, an executable metaprogramming language.

Keywords: Executable models, process modeling and execution, UML.

1 Introduction

Executable process models are process models that can be used not only for documenting processes and methods but also for the support of their execution. Indeed, executable process models can be used to coordinate between agents, to enforce artifacts routing between process's steps, to ensure rules and constraints integrity and process deadlines. They can also be of an effective aid since they can be used for simulation and testing. Simulation results can be used as a basis for important improvement decisions such as resource allocation, deadlock identification, estimation of the project duration and many other aspects that have a direct impact on the process and thus on the quality of the delivered software.

* This work is supported in part by the IST European project "MODELPLEX" (contract no IST-3408).

Q. Wang et al. (Eds.): ICSP 2009, LNCS 5543, pp. 148–160, 2009.

During the last two decades, the need for executable Software Process Modeling Languages (SPML) has been widely recognized. Osterweil opened the way with its seminal work *"Software Processes are Software Too"* [12]. He introduced the notion of *Process Programming*, which consisted in representing software processes in terms of computer-readable programs. The main goal behind this was to ensure agent coordination and the automation of process's repetitive and non-interactive tasks through the execution of *process programs*. The process programming trend stimulated many research works and had as an impact, the emergence of a multitude of SPMLs. These SPMLs were based on some well-known programming languages (e.g., Ada, LISP) or formal formalisms such as Petri Nets and put a strong emphasis on the executability aspect.

One of the lessons learned from these first-generation languages is that comprehensibility and communication of process's agents around process models is at least as important as their degree of formality [4]. The use of low-level formalisms by some process description languages, the lack of flexibility and the impossibility for non-programmers to use them, were among the main causes of their limited adoption.

Another fact that became manifest to the software process modeling community was the critical need of having a standard formalism for representing and exchanging software processes. Instead of reinventing the wheel, many industrial and research teams were attracted by the success of UML (Unified Modeling Language) and explored the possibility of using it as a process modeling language [2] [3] [10] [15]. UML is standard, provides a rich set of notations and diagrams, extension mechanisms and whatever its advantages and drawbacks, it is undeniably one of the most adopted modeling languages of this decade. Experiences with UML were not restricted to the software process community but covered other areas such as the business process and the workflow domains [9]. However, these experiences faced in their turn a major barrier. Despite the expressiveness of the language, UML models are not executable. Process models were used as contemplative rather than productive assets. An example of such propositions in the industry is the OMG's SPEM standard (Software Process Engineering Metamodel) [10]. While execution was out of the scope of the first version of SPEM (i.e. SPEM1.1), it has been established as a mandatory requirement in its second revision (i.e. SPEM2.0). Unfortunately, the recently adopted standard fails in ensuring this requirement.

In this paper we propose to deal with the executability issue in the context of UML-based process modeling languages. At this aim, we propose a framework and an approach for modeling and executing software processes. The proposed framework is based on our dedicated language for software process modeling called UML4SPM (UML-based Language for Software Process Modeling) [1] and a metaprogramming language called Kermeta [7]. UML4SPM comes in form of a MOF (Meta Object Facility)-compliant metamodel [11], a notation and semantics that extend the UML2.0 standard. To make UML4SPM process models executable, the semantics of the metamodel is implemented in terms of operations and instructions using Kermeta. This implementation is then woven into the UML4SPM metamodel using aspect techniques. It is worth noting that the approach described in this paper for building an executable environment for UML4SPM models can be generalised to any other MOF-instance language and is not restricted to UML-based languages.

The paper is organized as follows. Section 2 discusses how UML 2.0 *Activities* can be extended to build a software process modeling language and details the

UML4SPM language. Section 3 presents the executable semantics of UML4SPM and shows how it is implemented using Kermeta. An example is used to illustrate our approach. Related work is addressed in Section 4. Finally, in section 5 we discuss this work and we conclude the paper.

2 UML as a Basis for Software Process Modeling

In UML2.0, *Activities* have changed radically from UML1.x. Indeed, in the last version of the standard, *Activities* are not only suitable for modeling processes; they also have some features to support their automation. This is made possible thanks to *Action* packages, which now allow expressing the semantics of most executable instructions that one can find in common programming languages.

UML2.0 *Activities* also provide coordination mechanisms in order to ensure *proactive control*[1] and *reactive control*[2]. The first kind of coordination mechanism is ensured using concepts such as *Control Flow*, *Object Flow* and *Invocation Actions* (e.g., *CallBehaviorAction*). Reactive control is ensured thanks to the use of UML2.0 *Events*, *AcceptEventAction* and *SendSignalAction* constructs. For more sophisticated coordination mechanisms like concurrency, synchronization, merge, etc., *Control Nodes* can be employed. For instance, a *Fork Node* combined with a *CallBehaviorAction* can be used for modeling multiple and parallel activity calls. Furthermore, some experiences have been realised in order to evaluate the ability of UML2.0 *Activities* to support some well-known and complex Workflow patterns [13]. These experiences revealed that UML2.0 supports more than thirty control flow patterns of forty-three, which makes it more expressive than most business process formalisms such as BPEL (Business Process Execution Language) [14]. UML2.0 also offers some advanced constructs such as *Loop*, *Conditional Nodes*, and concepts to deal with exception handling, which is lacking in most current SPML propositions. All these facilities offered by UML2.0 added to the fact that it is a standard, that many people are already familiar with its notation and diagrams, and that a wide bunch of tooling support is provided, make UML a good candidate as a software process modeling language [1]. However, apart from the notion of *Activity*, UML lacks of some primary process elements, which constitute the vocabulary necessary for modeling software processes. This set of concepts was identified by many initiatives in the literature and regroups elements such as *Role*, *WorkProduct*, *Agent*, *Tool*, *Guidance*, *Team*, etc. [6].

In our proposition UML4SPM, we propose to deal with this issue by introducing these primary process elements into UML2.0. This is obtained by extending the UML metamodel and more precisely, the *Activity* and *Artifact* metaclasses. This extension comes in form of a MOF-compliant metamodel and is presented in fig. 1. White boxes represent the UML metaclasses we extended, i.e. UML2.0 *Activity* and *Artifact* metaclasses.

The UML4SPM metamodel aims at defining the minimal subset of concepts for software process modeling while relying on the advanced constructs and activity coordination mechanisms offered by UML2.0. Since the aim of this paper is to

[1] An imperative specification of the order in which activities (actions) are to be executed - direct invocation.

[2] A reactive specification of the conditions or events in response to which activities (actions) are to be executed - indirect invocation.

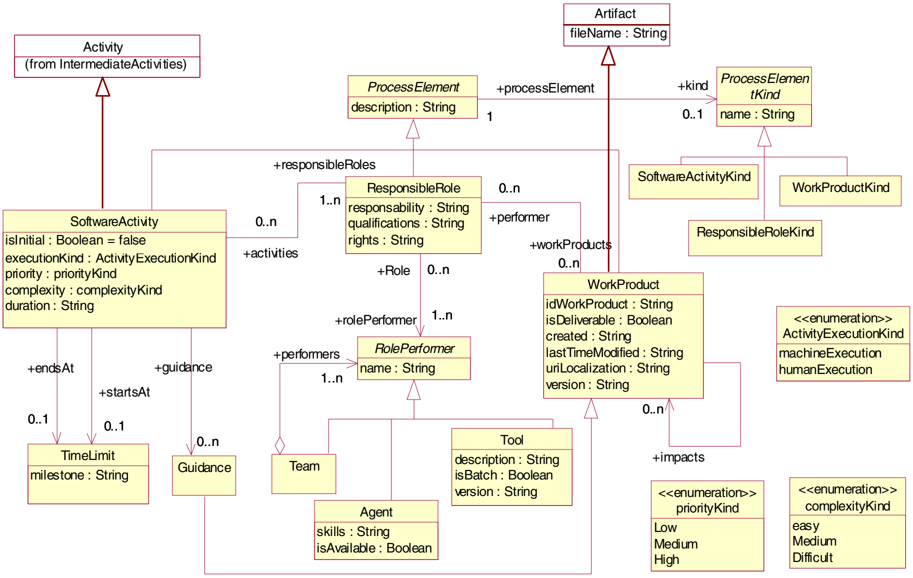


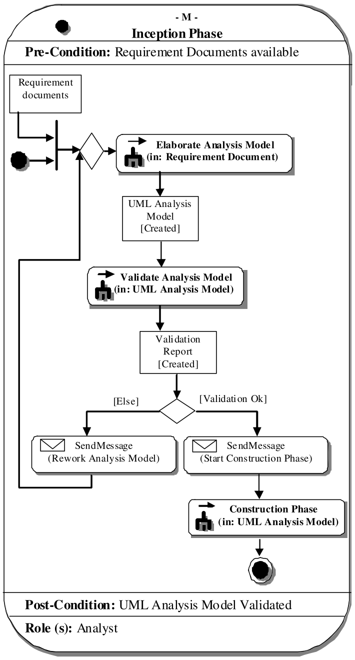
Fig. 1. UML4SPM Metamodel

present the executability aspect of UML4SPM and not the language itself, the interested reader can refer to [1] for more details on the metamodel.

By making UML4SPM *Software Activity* extending the UML2.0 *Activity* metaclass, we take advantage of all its properties and associations. Thus, a *Software Activity* can be composed of other *Software Activities* and may contain *Actions*. An UML2.0 *Activity* being indirectly a *Classifier*, the ability to specify new *properties* and new *operations*, as well as *pre* and *post conditions* on the execution of a *Software Activity* is also made possible.

The UML4SPM *WorkProduct* element extends UML2.0 *Artifact*. It represents any physical piece of information consumed, produced or modified during the software development process. An *Artifact* being a *Classifier*, *WorkProducts* can be defined as *InputPins* and *OutputPins* of *Software Activities* and *Actions*. It is also possible to specify composite *WorkProducts* thanks to the reflexive "nested artifact" association (not presented in the figure).

We also enriched the UML2.0 activity diagram notations in order to take into account some new properties and aspects specific to software process modeling that we introduced by our extension. It is important to note that this extension do not affect neither the comprehensibility of people already familiar with the UML2.0 Activity constructs nor their semantics. One that makes use of Activity diagrams can easily use the UML4SPM notations. This notation is given in fig. 2. Looking to the figure, one can identify the activity's name, its input and output parameters (and possibly their current state), its priority in the process, its duration, the assigned roles, the tools used for performing the activity, accepted and triggered events, if it's machine or human-oriented, etc. Post and pre conditions can be expressed using OCL2.0 constraints (Object Constraint Language). These constraints have to be expressed upon process's constituents (i.e., properties and states of WorkProducts, activities, roles, etc.). Of


- M -
Inception Phase
Pre-Condition: Requirement Documents available
Requirement documents
Elaborate Analysis Model (in: Requirement Document)
UML Analysis Model [Created]
Validate Analysis Model (in: UML Analysis Model)
Validation Report [Created]
[Else]
[Validation Ok]
SendMessage (Rework Analysis Model)
SendMessage (Start Construction Phase)
Construction Phase (in: UML Analysis Model)
Post-Condition: UML Analysis Model Validated
Role (s): Analyst

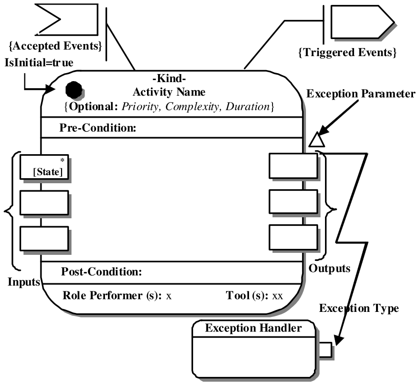


Fig. 2. The UML4SPM Software Activity Notation

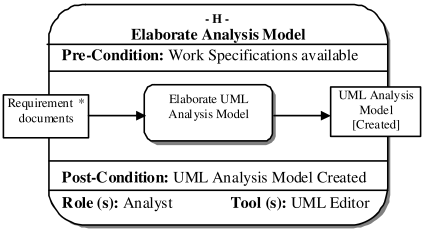

- H -
Elaborate Analysis Model
Pre-Condition: Work Specifications available
Requirement * documents
Elaborate UML Analysis Model
UML Analysis Model [Created]
Post-Condition: UML Analysis Model Created
Role (s): Analyst
Tool (s): UML Editor

Fig. 3. Software Process Example

course, it is not mandatory that all these features appear on the activity representation. Fig. 3, gives a simple yet representative example of a portion of a software process modelled using the UML4SPM notation. This process example was provided by our industrial partners within the IST European Project MODELPLEX[3]. We will use it throughout the paper to demonstrate our approach.

The "Inception Phase" activity represents the context of this process (i.e., container for all process's activities). This is indicated by the start-blob in the top-left corner. It is used to coordinate between different process's activities and WorkProducts. The "M" letter is to indicate that the activity is machine-executable (H for Human execution). One important aspect is the use of *CallBehaviorActions* in order to initiate/call process's activities (e.g., "Elaborate Analysis Model" call). In the call, one has to precise 1) whether the call is synchronous (use of a complete arrow in the top-left corner) or asynchronous (half arrow, e.g., "Construction Phase" call); 2) the parameters of the call, which represent WorkProducts inputs/outputs of the activity. Another aspect is the use of *Decision* and *Merge* nodes. The decision node allows for the expression of a choice of actions to perform depending on a condition (in this case, if the analysis model is valid or not). Conditions have to be expressed on activity edges (i.e., object flows) and will be evaluated at runtime. The merge node here is used to express that the "Elaborate Analysis model" activity may be triggered by one of the

[3] Modelplex, IST European Project contract IST-3408, at http://www.modelplex-ist.org/

two possibilities. The first one is when the "Inception Phase" activity is launched. The second one is when the analysis model validation fails.

At this level, UML4SPM is used only for modeling purposes. Since it is UML-based, there is no direct support for executing UML models. Even if UML2.0 provides execution semantics for each activity's constructs and actions, no implementation or virtual machine is provided. In the next section, we will see how to deal with this issue by introducing what we call *Execution Model*. That latter specifies the operational semantics of each element of the UML4SPM metamodel and particularly of UML2.0 *Activity* and *Action* elements. The *Execution Model* is then implemented using Kermeta, our metaprogramming language. The running example described above will be used to explain the approach.

3 Weaving Executability into Metamodels

The approach we propose for defining executable models requires two main steps. The first one consists in defining the *Execution Model*, which aims at specifying the operational semantics of the metamodel. It defines how each element of the metamodel should react at runtime and the set of operations it has to perform. In the context of UML4SPM for instance, this means to specify how the activity starts its execution, how roles are assigned to activities, how WorkProducts are automatically routed between activity's actions, how activities react to events, etc.

The second step is to formalise this semantics at the metamodel level. In UML4SPM, the operational semantics was implemented using Kermeta and integrated to the metamodel. The following sub-sections present the UML4SPM *Execution Model* and its implementation using Kermeta.

3.1 Definition of the Execution Model

The idea of the *Execution Model* is inspired from the RFP (Request For Proposal) issued by the OMG called: *Executable UML Foundation* [8]. The objective of this initiative is the definition of a compact subset of UML 2.0 to be known as “Executable UML Foundation”, along with a full definition of its execution semantics. Since that the building blocks of UML4SPM are UML2.0 *Activity* and *Action* packages, we found it interesting to take advantage of this specification, while focusing on UML2.0 elements we reused in our SPML. In UML4SPM, *Activity* and *Action* elements are used for sequencing the process's flow of work and data, for expressing actions, events, decisions, concurrency, exceptions, and so on. Thus, the implementation of the execution behavior of these concepts will be used as the core engine of UML4SPM.

The UML4SPM *Execution Model* introduces the execution model in form of class diagrams; each class represents the executable class of a UML4SPM element. An executable class is a class having a set of operations aiming at describing the execution behavior of the UML4SPM element at runtime. If the element is an UML element reused by UML4SPM, then its semantics is implemented according to the one given by the UML2.0 standard in natural language. The implementation of the UML *Execution Model* was restricted to *Activity* and *Action* elements that we reused within UML4SPM, and which respects the UML2.0 semantics.

Fig. 4. gives an example of the operations and features required for an *Activity Node* to execute. In UML, *Activity Nodes* regroup *Actions, Object Nodes (pin*s), and *Control Nodes* metaclasses. The execution semantics adopted by UML2.0 activities is quite similar to Petri Nets one and is based on offering and consuming tokens between the different activity's constituents (i.e., *Activity Nodes* and *Activity Edges*).

To illustrate this, let's go back to the example we defined in figure 3. When the "Elaborate Analysis Model" action ends, it produces an output, which is the "UML Analysis Model" document. This document is placed in the action's OutputPin. In UML, an OutputPin represents a container that holds action's output values (i.e., Tokens). An action has an OutputPin for each type of output it produces. The same applies for InputPin. This output has then to be consumed by the "Validate Analysis Model" action. Prior to this, the output has to be first put in the action's OutputPin, offered by the OutputPin to all its out coming edges, checked against guards or conditions, if any, which may be specified between the first action's outputpin and the second action's inputpin. In the example, we can figure out a guard specifying that the "UML Analysis Model" document's state should be set at "created" when passing from the source action into the target action, otherwise, the target action will not start. If the guard is satisfied and the target action is ready to execute, then the output is transferred from the source action's OutputPin into the target action's InputPin, which would then fire the execution of the action.

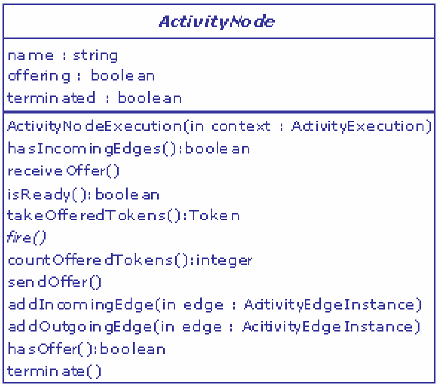

ActivityNode
name : string
offering : boolean
terminated : boolean
ActivityNodeExecution(in context : ActivityExecution)
hasIncomingEdges():boolean
receiveOffer()
isReady():boolean
takeOfferedTokens():Token
fire()
countOfferedTokens():integer
sendOffer()
addIncomingEdge(in edge : ActivityEdgeInstance)
addOutgoingEdge(in edge : AcitivityEdgeInstance)
hasOffer():boolean
terminate()

Fig. 4. Specification of the *ActivityNode's* Behavior

Although the example looks very simple in the figure, in order to execute, many actions have to be carried out. Each concept has a precise behaviour to perform. Fig. 5. shows a sequence diagram that generalizes all the operations that need to be executed in order to ensure such interactions between any kind of *Activity Nodes*. To refer to the example, it represents the interactions between the source action's outputpin, the activity edge and the target action's inputpin.

Thus, once all metamodel element's behaviours defined in terms of operations, the next step consist in implementing them using Kermeta and to weave them as *aspects* into the UML4SPM metamodel. Of course, these two steps have to be carried only once and are completely transparent to the UML4SPM process modeller, who just instantiates the metamodel (from a graphical editor for instance).

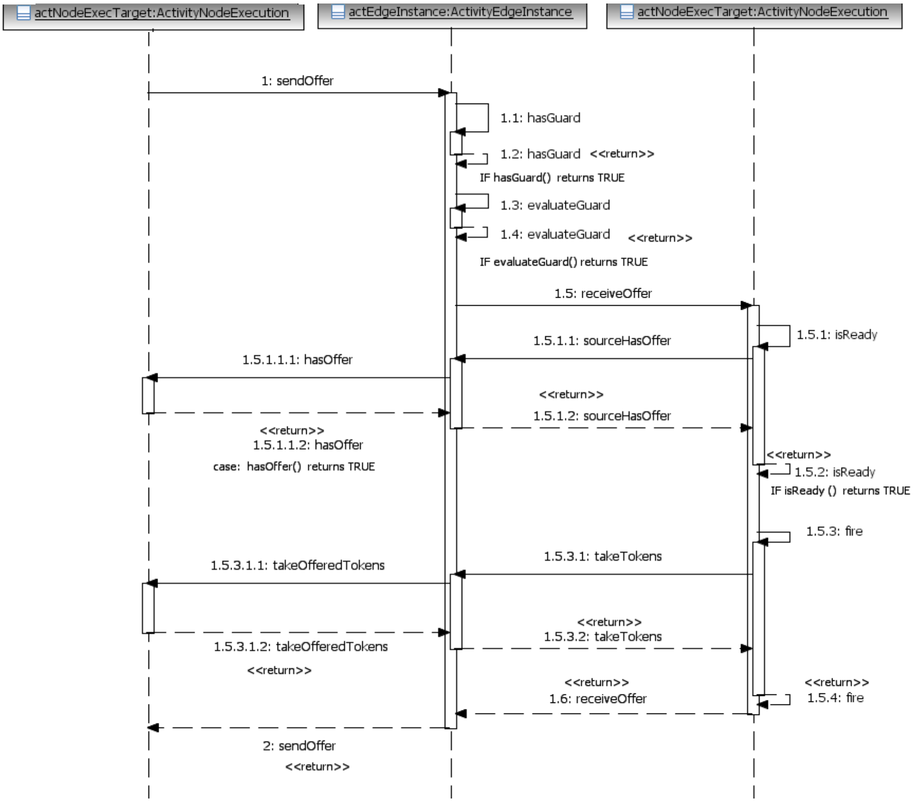


Fig. 5. *ActivityNode* and *ActivityEdge* Interactions

3.2 Implementation of the Execution Model Using Kermeta

Kermeta is an MDE platform designed to specify constraints and operational semantics of metamodels [7]. The MOF [11] supports the definition of metamodels in terms of packages, classes, properties and operations but it does not include concepts for the definition of constraints or operational semantics. Kermeta extends MOF with an imperative action language for specifying constraints and operation bodies at the metamodel level.

One of the key features of Kermeta is the static composition operator, which allows extending an existing metamodel with new elements such as properties, operations, constraints or classes. This operator allows defining various aspects in separate units and weaving them automatically into the metamodel. The weaving is done statically and the composed model is typed-checked to ensure the safe integration of all aspects. This mechanism makes it easy to reuse existing metamodels or to split metamodels in reusable pieces. It also provides flexibility. For example, several operational semantics can be defined in separate units for a single metamodel and then alternatively woven depending on a particular need. This is the case for instance in the UML metamodel where several semantics variation points are defined.

The purpose of Kermeta is to remain a core platform for safely integrating all the aspects around a metamodel. For instance, metamodels can be expressed using MOF and constraints using the OCL. Kermeta also allows importing Java classes in order to use services such as file input/output or network communications, which are not available in the Kermeta standard framework. This is very useful for instance to allow interactions between models and existing Java applications. In the case of UML4SPM, this allows processes to interact with business applications, the enterprise workflow, to call distant web services and so on.

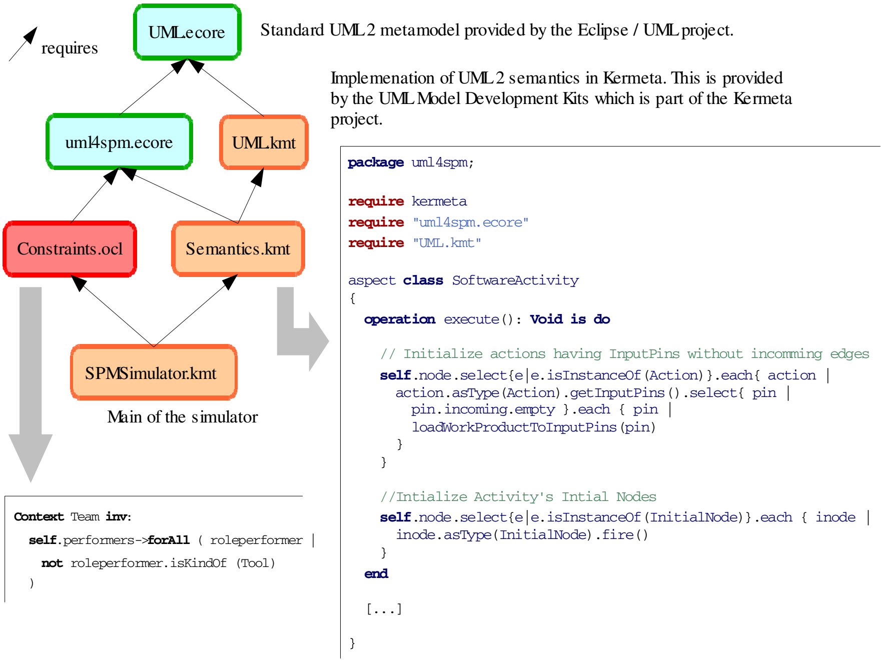
Fig. 6 presents an overview of the architecture of the UML4SPM implementation using Kermeta. The diagram shows the units to be composed in order to build the UML4SPM environment and simulator. Ecore files (UML.ecore and uml4spm.ecore) are metamodels expressed using the Eclipse Modeling Framework (EMF). Because the EMF is compliant with the EMOF standard, these metamodels can be used directly in the implementation. UML.ecore corresponds to the standardized UML 2 metamodel provided by the Eclipse/UML project. The uml4spm.ecore metamodel corresponds to the extension of UML for software process modeling given in Fig. 1.

The *.kmt files on Fig. 6 correspond to Kermeta source files. The UML.kmt is an implementation of the UML semantics in Kermeta. This file especially implements the semantics of UML 2 activity diagrams, which is reused in the context of the UML4SPM extension. The file Semantics.kmt corresponds to the implementation of the UML4SPM *Execution Model.* An excerpt of the source code of this file is shown on the right hand side of Fig. 6. The first line of the listing specifies the containing package for the definition contained in the file. Then the "require" directives are used to declare dependencies with other units. In the example, the uml4spm metamodel defines a metaclass named uml4spm::SoftwareActivity. The piece of code shown on the listing adds an operation named "execute" in this metaclass.

Adding new elements to a metaclass of the metamodel is achieved using the keyword "aspect" before the declaration of the class. The body of the operation "execute" presented in Figure 6 implements how a software activity can be executed. The execution of an activity consists of initializing actions and initial nodes of the activity. In the code, we first search for actions having input pins without incoming edges in order to initialize them with WorkProducts of the same type and then we look for initial nodes and initialize them by calling the operation "fire". In order to fully implement the execution model of the UML4SPM metamodel, all required operations are implemented in the same way as for the "execute" operation detailed on the listing.

The file Constraints.ocl shown in Figure 6 encapsulates constraints on the UML4SPM metamodel. These constrains are written in standard OCL. Figure 6 presents the listing of a simple constraint as an example. In the metamodel given in Figure 1 there is an aggregation called "performers" from the *Team* metaclass to *RolePerformer* metaclass. In practice, the performers of a team can be either teams or agents but not tools. The constraint presented is an invariant for the metaclass *Team* that ensures that no tools can be added as performers.

Finally, the Kermeta source file SPMSimulator.kmt contains the entry point for a simulator, which can load process models (i.e. instances of the uml4spm Ecore metamodel), check the constraints on these models thanks to the OCL constraints and execute these models using operations that were weaved into it.


requires
UML.ecore
Standard UML2 metamodel provided by the Eclipse / UML project.
Implemenation of UML2 semantics in Kermeta. This is provided by the UML Model Development Kits which is part of the Kermeta project.
uml4spm.ecore
UML.kmt
Constraints.ocl
Semantics.kmt
SPMSimulator.kmt
Main of the simulator

```
package uml4spm;

require kermeta
require "uml4spm.ecore"
require "UML.kmt"

aspect class SoftwareActivity
{
  operation execute(): Void is do

    // Initialize actions having InputPins without incomming edges
    self.node.select{e|e.isInstanceOf(Action)}.each{ action |
      action.asType(Action).getInputPins().select{ pin |
        pin.incoming.empty }.each { pin |
        loadWorkProductToInputPins(pin)
      }
    }

    //Intialize Activity's Intial Nodes
    self.node.select{e|e.isInstanceOf(InitialNode)}.each { inode |
      inode.asType(InitialNode).fire()
    }
  end

  [...]

}
```

```
Context Team inv:
  self.performers->forAll ( roleperformer |
    not roleperformer.isKindOf (Tool)
  )
```

Fig. 6. Weaving Executability to The UML4SPM Metamodel

4 Related Work

In this section we only deal with UML-based process modeling languages, taxonomy of first-generation PMLs can be found in [16].

In the industrial side, SPEM1.0 was the first standard SPML based on UML (UML1.4) [10]. However SPEM1.0 has had a limited success within the industry since SPEM1.0 did not offer any execution support. Process models were only contemplative models. In SPEM2.0, the main advance was the proposition of a clear separation between the content of a method of its possible use within a specific process. SPEM2.0 extends the UML2.0 Infrastructure and does not use any concept from the UML2.0 Superstructure (i.e. Activities, Actions, etc.). Regarding executability, SPEM2.0 does provide neither concepts nor formalisms for executing process models. Instead, the standard proposes to either map process definitions into some project planning tools (e.g. MS. Project) which is not considered as process execution but a process planning activity or to define transformation rules into some business process execution languages (e.g. BPEL). Unfortunately, the standard does not define any of these rules.

In Di Nitto's et al. approach [3], authors aim at assessing the possibility of employing a subset of UML1.3 as an executable PML. It comprises two main phases. The first one consists in describing processes using UML diagrams. The second phase consists in translating these UML diagrams into code that can be enacted by the team's events-based workflow engine called OPSS. Process constituents can be defined by simply specializing a set of predefined classes provided by the approach in form of a UML class diagram. The flow of work is given in activity diagrams and the lifecycle of each entity is defined by a state machine. However, the activity and class diagrams have no links with each other. The approach does not extend the UML language nor introduces new concepts. Process elements are simply instances of the UML Class metaclass, which means that they all have the same semantics and notation as the UML Class metaclass. Regarding execution, it is essentially based on how state diagrams defined by the user are precise enough and sound in order to enable a complete code generation and to allow process execution within OPSS. Otherwise, code has to be added manually. The weak point in the executability aspect remains how information defined in activity diagrams (i.e., precedence between activities), state machines and class diagrams are integrated to generate each of the Java classes needed for the execution. Authors did not detail how this integration is realized.

Another approach, called Promenade [15], basically follows the same principle as DiNitto's. To model a process, one has to specialize the set of predefined classes provided by the approach. To define precedence between process's tasks, one has to define a precedence graph, which defines the order between all tasks of the process. However, authors do not specify how the precedence graph (including precedence rules) is to be integrated with the class diagram to form a complete process description. The approach does not provide any mechanism or way to execute Promenade process models. No tool or prototype was provided.

In [2], Chou proposed a software process modeling language consisting of high-level UML1.4-Based diagrams and a low-level process language. While UML diagrams are used for process's participants understanding, the process language is used to represent the process - from UML diagrams – in a machine-readable format i.e., a program. The principal obstacle of this approach is the lack of an automatic generation of process programs from UML diagrams, which imposes the rewriting of the process by developers mastering the proprietary OO language provided by the author.

5 Discussion and Conclusion

Contrarily to traditional process model execution approaches, one key feature of our approach is the ability to execute process models without any transformation or compilation step. Indeed, current propositions require a compilation phase towards some execution languages, sometimes proprietary, in order to execute them (cf. section 4). This step is most often followed by a manual coding step for configuring some aspects of the process execution, which is error prone and may induce some traceability issues between process models and their execution. Using Kermeta, the execution behavior is defined once in the metamodel and can then be instantiated many times. Process modelers do not have to deal with code. It is completely transparent for them. Process models are directly enclosing an execution behavior and can be executed and simulated straightforwardly without any compilation or transformation phase.

It is also worth noting that the operational semantics we defined respects the one given by the UML2.0 specification. The fact that that latter is weaved into the UML2.0 metamodel makes it possible to simulate UML2.0 activity diagrams. Since UML4SPM extends UML2.0, this semantics is used as the building block of the

UML4SPM simulator. Kermeta also offers features that allow triggering actions outside the Kermeta virtual machine. This would allow the process execution to interoperate with enterprise's applications or external services.

Regarding the expressiveness of UML4SPM, we evaluated it with the well-known ISPW-6 Software Process Example [5], a standard benchmark software process problem developed by experts in the field of software process modeling. The description of the benchmark process by UML4SPM was not just limited to the eight activities of the core problem but it also succeeded to express most optional extensions. Tool invocation actions, communication mechanisms, exception handling, WorkProduct versioning and management features and other constructs offered by UML4SPM were used at this aim. This evaluation is presented in more details in [4] .

Finally, in this paper we introduced *Executability* of models in the context of UML4SPM, however, it can be generalized to any MOF-instance language. An important perspective of this work is the definition of the set of activities and constraints that would allow a process definition to be modified at runtime and without restarting the process execution. This work is ongoing using Kermeta and aspect oriented modeling techniques.

References

1. Bendraou, R., Gervais, M.-P., Blanc, X.: UML4SPM: A UML2.0-based metamodel for software process modelling. In: Briand, L.C., Williams, C. (eds.) MoDELS 2005. LNCS, vol. 3713, pp. 17–38. Springer, Heidelberg (2005)
2. Chou, S.C., Chen, J.Y.J.: Process Program Development Based on UML and Action Cases, Part 1: the Model. Journal of Object-Oriented Programming 13(2), 21–27 (2000)
3. Di Nitto, E., et al.: Deriving executable process descriptions from UML. In: Proc. of the 24th International Conference on Software Engineering (ICSE), Orlando, Fl. ACM Press, New York (2002)
4. Fuggetta, A.: Software Process: A Roadmap. In: 22nd International Conference on Software Engineering (ICSE), Limerick (Ireland), June 4–11. ACM, New York (2000)
5. Kellner, M.I., Feiler, P.H., Finklestein, A., Katayama, T., Osterweil, L.J., Penedo, M.H., Rombach, H.D.: ISPW-6 software process example. In: Proc. of the first Intern. Conf. on the Software Process, pp. 176–186. IEEE Computer Society, Washington (1991)
6. Lonchamp, J.: A structured conceptual and terminological framework for software process engineering. In: Proceedings of the 2nd International Conference on the Software Process (ICSP 2), Berlin, Germany. IEEE Computer Society Press, Los Alamitos (1993)
7. Muller, P.A., Fleurey, F., Jézéquel, J.M.: Weaving executability into object-oriented metalanguages. In: Briand, L.C., Williams, C. (eds.) MoDELS 2005. LNCS, vol. 3713, pp. 264–278. Springer, Heidelberg (2005)
8. OMG, Semantics of a Foundational Subset for Executable UML Models RFP, OMG document ad/05-04-02 (April 2005), http://www.omg.org/docs/ad/05-04-02.pdf
9. OMG, Workflow Management Facility Specification v1.2, OMG document formal/00-05-02 (April 2000), http://www.omg.org

[4] UML4SPM evolution using ISPW6: http://pagesperso-systeme.lip6.fr/Reda.Bendraou/Documents/UML4SPMEvaluation_ISPW6.pdf

10. OMG SPEM1.0, Software Process Engineering Metamodel, OMG document formal/02-11/14 (November 2002), `http://www.omg.org`
11. OMG MOF, Meta Object Facility version 2.0, adopted specification, OMG document formal/06-01-01 (January 2006), `http://www.omg.org`
12. Osterweil, L.: Software Processes Are Software Too. In: Proceedings of the 9th International Conference on Software Engineering (ICSE 9). ACM Press, New York (1987)
13. Van der Aalst, W.M.P., et al.: Workflow Patterns. Journal of Distributed and Parallel Databases 14(3), 5–51 (2003)
14. Wohed, P., et al.: Pattern-based Analysis of the Control-Flow Perspective of UML Activity Diagrams. In: Delcambre, L.M.L., Kop, C., Mayr, H.C., Mylopoulos, J., Pastor, Ó. (eds.) ER 2005. LNCS, vol. 3716, pp. 63–78. Springer, Heidelberg (2005)
15. Franch, X., Rib, J.: A Structured Approach to Software Process Modelling. In: Proceedings of the 24th Conference on EUROMICRO, vol. 2 (1998)
16. Zameli, K.Z., Lee, P.A.: Taxonomy of Process Modelling Languages. In: Proc. of the ACS/IEEE Inter. Conf. on Computer Systems and Applications (AICCSA 2001), Beirut, Lebanon (June 2001)

Dynamic COQUALMO: Defect Profiling over Development Cycles

Dan Houston, Douglas Buettner, and Myron Hecht

The Aerospace Corporation, P.O. Box 92957, Los Angeles, California 90009
{daniel.x.houston,douglas.j.buettner,myron.j.hecht}@aero.org

Abstract. Various techniques have been used for managing software quality, including those that predict defect counts over time. This paper introduces a simulation model based on COQUALMO, which can be calibrated to organizational process performance for estimating counts of residual defects. This simulator has the additional benefit of producing a set of estimated defectivity profiles over a software development cycle. Such a set of profiles can be used to support quality management decisions regarding the amount and level of defect removal activities to be applied during a development cycle.

Keywords: Defect profile, COQUALMO, defect introduction and removal.

1 Complementarity in Software Quality Management

Managing software quality requires information on both key product qualities as well as defectivity. These involve complementary views of a product, illustrated in Figure 1. The former are usually specific to a product or product line, for example, usability and response time measures. The latter are generic, based on error and defect counts, and may be analyzed through profiling over time or by categorization (severity, type, origin, and so forth). In both quality views, actual values are measured: product quality measures are compared to goals; defect counts are tracked to expected values. Also, in both product quality and defectivity views, early tracking, rather than tracking only later in product or system testing, provides information that can be used to reduce both risk and cost of development. For product qualities, this early quality management is epitomized in the predictive modeling approach of Design for Six Sigma, also called critical parameter management [5]. For product defectivity, early quality management is exemplified by defect severity profiles, defect leakage matrices, reliability growth curves, and defect discovery profiles.

This paper describes a defectivity estimation tool, Dynamic COQUALMO (DC) that falls into the last category, defect discovery profiles. DC is based on the COnstructive QUALity MOdel (COQUALMO)[1], described in the next section. Subsequent sections discuss defect profiling over time and how profiling can be combined with COQUALMO, resulting in a simulation method for defectivity estimation. Finally, an application of the method is described.

[1] COCOMO II and COQUALMO were developed at the Center for Systems and Software Engineering of the University of Southern California.

Q. Wang et al. (Eds.): ICSP 2009, LNCS 5543, pp. 161–172, 2009.

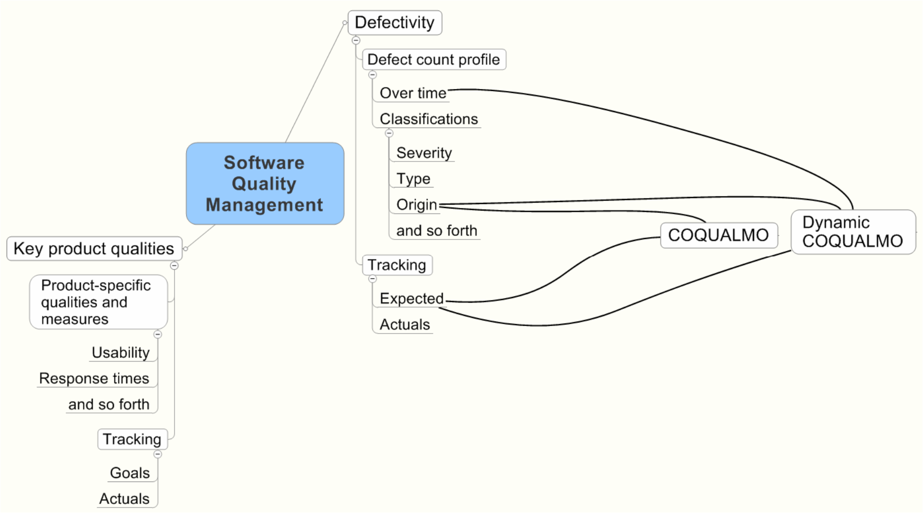


Fig. 1. Complementarity in Software Quality Management and the Positions of COQUALMO and Dynamic COQUALMO

2 COQUALMO

In an effort to produce a software quality prediction tool that relates defectivity to cost and schedule, Devnani-Chulani [7,8] extended COCOMO II[1] with two sub-models, one each for defect introduction and defect removal. The defect introduction and removal process is illustrated in Figure 2.

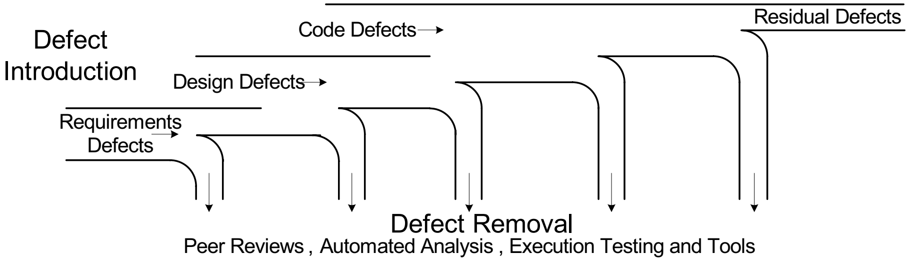


Fig. 2. COQUALMO Software Defect Introduction and Removal Model [7]

The extension treats the COCOMO II factors as quality drivers as well as cost drivers. Quality is measured in counts of non-trivial defects. These include those defects classified as critical (causes a system crash or unrecoverable data loss or jeopardizes personnel), high (causes impairment of critical system functions and no workaround solution exists), or medium (causes impairment of critical system function, though a workaround solution does exist).

For each source of defects (requirements, design and coding), the rate of defects introduced is a nominal value (DIR_{nom}) modified by a quality adjustment factor, which is a product of defect driver values assigned to COCOMO II factor scales. The defect driver values were assigned to the factor scales in a two-round Delphi exercise [6]. For example, the number of defects introduced during requirements development, *DI-$_{req}$*, is calculated by (1).

$$DI_{req} = DIR_{req,nom} * Size^{B_{req}} * \prod_{i=1}^{21} DefectDriver_{i,req} \quad (1)$$

The defect removal sub-model is based on three activities: peer reviews, automated analysis, and execution testing and tools. The process quality of each of these activities is scaled from Very Low (little or no defect removal) to Extra High (best defect removal using best process technology and tools).

Table 1. Levels of Defect Removal Activities (abbreviated from [7])

Rating	Peer Reviews	Automated Analysis	Execution Testing
Very Low	None	Simple compiler checking	None
Low	Ad hoc	Static module code analysis	Ad hoc
Nominal	Informal roles and procedures	Static code analysis; Requirements/design checking	Basic test process
High	Formal roles and procedures	Intermediate semantic analysis; Requirements/design checking	Organizat'l test process; Basic test coverage tools
Very High	Formality plus use of data	Temporal analysis & symbolic execution	Advanced test tools; Quantitative test process
Extra High	Review process improvement	Formal specification and verification	Highly advanced tools; Model-based test mgmt

Using a two-round Delphi exercise, a defect removal fraction (DRF) was assigned to each quality level of each activity. Residual defects for an artifact *j*, *DR$_j$*, are calculated as a product of the defects introduced into the artifact, *DI$_j$*, and the product of the residual defect fractions for each artifact *j* and defect removal activity *i*, (*1-DRF$_{ij}$*). For example, the number of residual requirements defects can be calculated as (2).

$$DR_{req} = DI_{req} * \prod_{i=1}^{3} (1 - DRF_{i,req}) \quad (2)$$

Using COQUALMO, then, requires estimated software size, nominal defect introduction values, project characteristics as COCOMO factor selections, and quality activity characteristics as a level selection for each activity.

3 COQUALMO and Simulation Models

COQUALMO is a static model that offers a means of estimating defect introduction and removal by artifact. However, it also provides a basis for dynamic modeling. Three research groups have employed COQUALMO in simulation models.

Choi and Bae [4] produced a software development simulation model based on COCOMO II and COQUALMO. Taking advantage of the quantitative relationships embodied in these two methods, the authors decomposed the time-aggregated effort, schedule, and defect estimates into product development and defect flows. The product flow is controlled by sectors for calculating productivity and effort-schedule distribution. The defect flows are controlled by a sector dedicated to COQUALMO calculations. To this model, they prepended sectors representing an end user producing requirements creep and an acquirer choosing trade-offs between cost, quality, and schedule. The model was used to illustrate results of several policies, including two ways of treating cost as an independent variable.

Tawileh *et al.* [16] modified Abdel-Hamid and Madnick's [1] model to incorporate a COQUALMO sector for calculating defect injection and removal rates. This model, which retains the highly aggregated representation of the software development flow, was used to study residual defect counts under various levels of peer reviews. The authors conclude that peer reviews offered the most effective method of defect removal compared to automated analysis and to testing.

Madachy and Boehm [12] developed a model of defect flows based on Orthogonal Defect Classification (ODC) categories and the COQUALMO defect generation and removal rates. The model was calibrated with NASA software defect data supporting identification of different detection efficiencies for pairings of defect removal techniques and defect types. For example, peer reviews are successful in finding requirements completeness defects but not timing errors. The ODC COQUALMO model provides insights into defect trends by type over time.

4 Defectivity Profiling over Time

Many researchers and practitioners have sought to establish expected values for defect profiles over time by fitting distributions to defect discovery data. For system test data, the resulting curves have been used to estimate latent defects and support test progress and readiness-for-release decisions. But over the last four decades, quality experts have sought to exert more influence in the early stages of product development to reduce the cost of product quality. The challenge of defectivity profiling has been obtaining defect-rate data spread over the course of a development cycle [13]. Fortunately, increased use of software inspections has provided data for time-based defectivity profiling tools such as the defect leakage matrix [3,15].

This investigation seeks to advance defectivity profiling by utilizing the quantitative relationships developed and refined in COCOMO II and COQUALMO. COCOMO II is used to estimate project duration, then the user of Dynamic COQUALMO (DC) may specify the duration of each phase. DC then provides a decomposition of defect estimates across phases so as to achieve a project defectivity profile. Figure 1 illustrates the conceptual role of COQUALMO and DC in software quality management.

5 Model Description

To translate COQUALMO into a simulation model, it was necessary to decompose some of its components. First, peer reviews were separated into requirements, design, and code reviews by decomposing and allocating the peer review DRFs, under the constraint that they aggregate to those specified in COQUALMO. Also, COQUALMO assumes the same quality of practice for each type of quality activity, for example, all peer reviews at performed at a nominal level. Realistically, quality activities are performed to varying degrees within a project. The model accommodates this variation by weighting the quality levels of each activity and taking a weighted average.

Testing is also decomposed to distinguish software development testing from system testing, in which software reliability is usually measured. This decomposition was also accomplished by weighting the two sets of testing DRFs under the constraint that they aggregate to COQUALMO values. Due to the many differences in testing processes across software types and organization, the two testing phases are simply called Testing 1 and Testing 2, thereby allowing a user to define the differences between them and weight their effectiveness accordingly.

Dynamic COQUALMO has six defect flows, an inflow and an outflow for each artifact type: requirements, design, and code. Each of these flows has a single source, but multiple outflows, one for each quality-inducing activity. For example, the requirements defect flow is fed by requirements defect generation, but drained by requirements reviews, automated analysis, design reviews, code reviews, testing 1, and testing 2 (Figure 3).

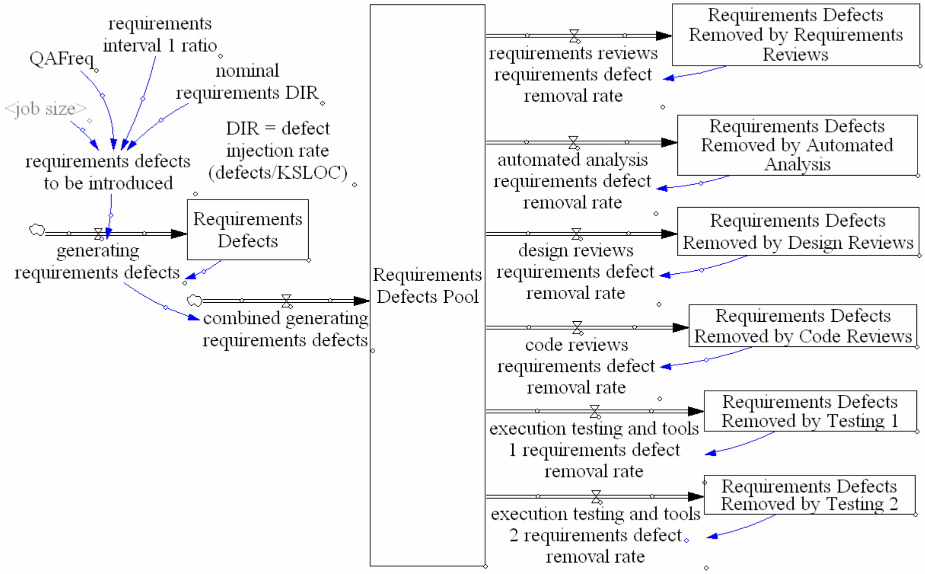

requirements interval 1 ratio
QAFreq
nominal requirements DIR
<job size>
DIR = defect injection rate (defects/KSLOC)
requirements defects to be introduced
Requirements Defects
generating requirements defects
combined generating requirements defects
Requirements Defects Pool
Requirements Defects Removed by Requirements Reviews
requirements reviews requirements defect removal rate
Requirements Defects Removed by Automated Analysis
automated analysis requirements defect removal rate
Requirements Defects Removed by Design Reviews
design reviews requirements defect removal rate
Requirements Defects Removed by Code Reviews
code reviews requirements defect removal rate
Requirements Defects Removed by Testing 1
execution testing and tools 1 requirements defect removal rate
Requirements Defects Removed by Testing 2
execution testing and tools 2 requirements defect removal rate

Fig. 3. Requirements Defects Flows

5.1 Inputs and Outputs

DC inputs include the following.

- Estimated job size in KSLOC.
- Settings for COCOMO II factors, including effort multipliers and scale factors.
- Estimated phase durations and degrees of phase concurrency such that they sum to the project duration.

- Usage profile of quality levels for each defect removal activity.
- Relative effectiveness estimates:
    - Relative effectiveness of requirements, design, and code reviews in finding requirements defects.
    - Relative effectiveness of design and code reviews in finding design defects.
    - Relative effectiveness of the two test phases in finding defects (requires definition of the differences between the two phases).

The DC model is designed to include changes of inputs resulting from significant changes during the course of a project. For example, such changes could include the effects of funding cuts, redeployment of very experienced personnel, or a redesign of a project for a greater degree of process discipline. These changes in project conditions can be reflected in COCOMO II factors and the usage profiles of defect removal activities. To accommodate a discrete project change, DC is implemented as an array of flows governed by input vectors for different time intervals. At any given simulation time, either the set of flows before a major change or the set following a major change dominates.

The COCOMO II scales for defect introduction and COQUALMO scales for DRFs place constraints on defect profiling that reflect project conditions. However, COQUALMO provides several parameters that may be used to calibrate profiles to organizational circumstances. These include the three nominal defect introduction rates, the three *Size* exponents representing economy/diseconomy of scale in defect introduction, and a multiplier to each of the defect removal rates.

The DC model produces a project defect profile and component profiles for generation and removal, both aggregated and by defect origin. Figure 4 is a particularly interesting example. The curve running the length of the project represents the project's defect profile. The upper right line indicates the number of defects after Test 1; the lower right line, the "reliability growth" curve typically plotted during a system test.

5.2 Composing Defect Profiles with Rayleigh Curves

Following Madachy's [12] example, Rayleigh distributions are used to model both defect introduction and removal at lower levels. Madachy modeled defects by ODC type, whereas this model uses a separate curve for defects introduced in each production activity (requirements, design, coding) and for defects removed in each combination of production activity and detection activity (requirements peer reviews, design peer reviews, and so forth). Rayleigh curves were used to model defect rates "in the small" for several reasons. First, they have the intuitive appeal of an early rapidly increasing rate and a long tail. Second, they are easy to implement as a function of time and amount flowing. Third, the Rayleigh distribution assumptions are often true "in the small" and violations can be mitigated in the model by timing offsets. (Assumptions are discussed further in the Appendix).

Rayleigh distributions have long been used to model project effort loading. Because defect generation can be considered as proportional to effort for a given set of project conditions, these distributions have also been used to model defect discovery rates. A consensus has emerged that a Rayleigh distribution can characterize defect discovery well over the course of a project. In an early analysis, Schick and Wolverton [14] used a hazard function to derive a software reliability model in the form of a

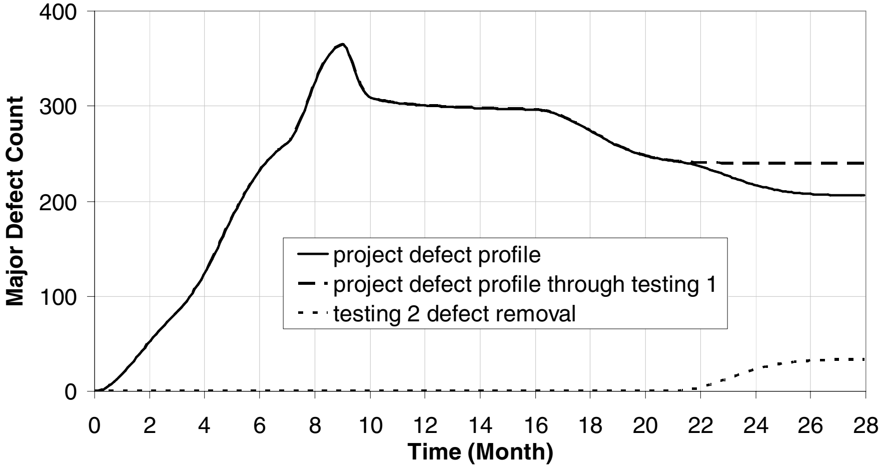


Fig. 4. Project Defect Profile

Rayleigh distribution. Trachtenberg [18] was one of the first to observe a pattern of defect occurrence like a Rayleigh curve. He later published a derivation of the Rayleigh distribution for defect discovery [19]. Following Trachtenberg's suggestion, Putnam and Myers [13] used a Rayleigh distribution to develop a defect rate equation and calibrated its parameters using data from many software development organizations. Gaffney's model [9] is a Rayleigh distribution based on data he analyzed. Thangarajan and Biswas [17] provided another example of development cycle defect discovery data that corresponds to a Rayleigh curve. Kan [10] treats the use of Rayleigh distributions for project defect profiles extensively, distinguishing between their uses for reliability assessment and for in-process quality management.[2]

The forgoing examples all considered the distribution of defect discovery over a development cycle. Their use for modeling defect discovery in phases has precedents. Kan [10] applied a Rayleigh distribution to defect discovery of field defects and found a good fit. Similarly, Modroiu and Schieferdecker [11] reported that defect discovery in each testing phase of communication network software was correlated highly with Rayleigh curves.

5.3 Rayleigh Curve Implementation

The Rayleigh distribution, a specific case of the Weibull distribution with shape parameter = 2, has one parameter describing its dispersion. A system dynamics model implements a rate described by a Rayleigh curve as (3).

(*Total amount to be processed – amount processed*) * *Time* * *buildup parameter* (3)

[2] Interestingly, Kan has found that a Rayleigh curve over a development cycle consistently underestimate field defect rates. Thus, he recommends a Weibull distribution with $m = 1.8$ when estimation accuracy at the tail is important. It is not important for this model.

COQUALMO provides *Total amount to be processed*, in this case, the total number of defects to be introduced or to be removed. A downstream stock accumulates the *amount processed*, in this case, the number of defects that have been introduced or removed). Therefore, the only parameter to be provided for rate calculation is the *buildup parameter*. This parameter can be formulated as a function of project duration and the desired duration of the activity represented by the curve described by (4).

$$(\textit{coefficient} * \textit{fractional duration}^\wedge\textit{exponent}) \;/\; \textit{planned development duration} \qquad (4)$$

Fortunately, the values of coefficient and exponent can be derived empirically. A Rayleigh curve generator was used to calculate coefficient and exponent values for various combinations of *planned development duration* (3 – 60 months) and *fractional durations* (.05 – 1.0) of the planned development schedule. The exponent for 95% Rayleigh curve completion is calculated by (5).

$$-0.01*\ln(\textit{planned development duration})-2.0377 \qquad (5)$$

Similarly, the coefficient for 95% Rayleigh curve completion is calculated by (6).

$$6.3889*\textit{planned development duration}^\wedge(-1.0564) \qquad (6)$$

Using these expressions, Dynamic COQUALMO can produce a buildup parameter for a Rayleigh curve that fits any portion of any development schedule.

6 Model Testing and Usage

The model has been subjected to a modest amount of sensitivity analysis and replication testing. It has been found to be most sensitive to the parameters controlling defect introduction: the product size, nominal defect introduction values, and the 23 Defect Rate Multipliers (DRMs). A larger set of parameters controls defect removal and individually these are less influential than the defect introduction parameters.

Size is by far the dominant parameter. The next most influential are the nominal defect introduction values. Individually the DRMs are far less influential; however, when taken together as a set of QAF values, they are very influential. The 23 DRMs can produce $1.7496\text{x}10^{16}$ values for each QAF with a wide range for each (Table 2).

The distributions of the QAFs were found by producing random sets of DRM values and calculating the three QAFs. The QAF distributions were found to be lognormally distributed with means between 1.3 and 1.7 and standard deviations between 1.1 and 1.7.

Table 2. QAF value ranges

	QAF_{Req}	QAF_{Des}	QAF_{Cod}
Lowest value	0.02	0.01	0.01
Highest value	67.71	136.06	152.78
Geometric Range	3921.98	12922.82	16160.13

The model was tested for its ability to replicate defect discovery curves of two projects, referred to as Project A and Project C. These are space system flight software projects that produced 68 KSLOC (Ada) and 99 KSLOC (50 Ada, 49 assembly), respectively. The defect data was collected from peer review records and defect record repositories by Buettner [2]. Though it has a much larger number of defects, Project C's better use of peer reviews and testing produced a mature product in a much shorter time than did Project A. (Figure 5 includes major defects from both peer reviews and testing.) Each project experienced a major change during its course. Project C was redesigned during its third year (following initial defect discovery) and Project A was revised in its eighth year during testing. The change was much more dramatic for Project C (average QAF change from 11.4 to 0.31) than for Project A (average QAF change from 3.2 to 1.5).

To replicate the defect discovery curves, DC was calibrated to the projects by two people very familiar with them: they set the values of size, of DRMs, of major change time, and of the removal activity profiles. Further adjustments were made to the usage profiles and to the nominal defect introduction values to produce Figure 5.

Several lessons were learned from the replication exercise.

- COQUALMO values for nominal defects introduced (10, 20, and 30 defects /KSLOC for requirements, design, and code) appear to be high. Values between .5 (Project C requirements) and 6.1 (Project C code) were used to produce the modeled curves, loosely constrained by limited knowledge of the ratios of the actual defect sources.
- The need to adjust the usage profiles suggests that either COQUALMO's DRF values require adjustment, or the usage of defect removal activities was reported inaccurately, or both.
- Software development projects seem to have characteristic defect discovery profiles. DC can replicate a discovery profile and, by inference, produce a realistic defect profiles for use in managing quality effort in future projects.

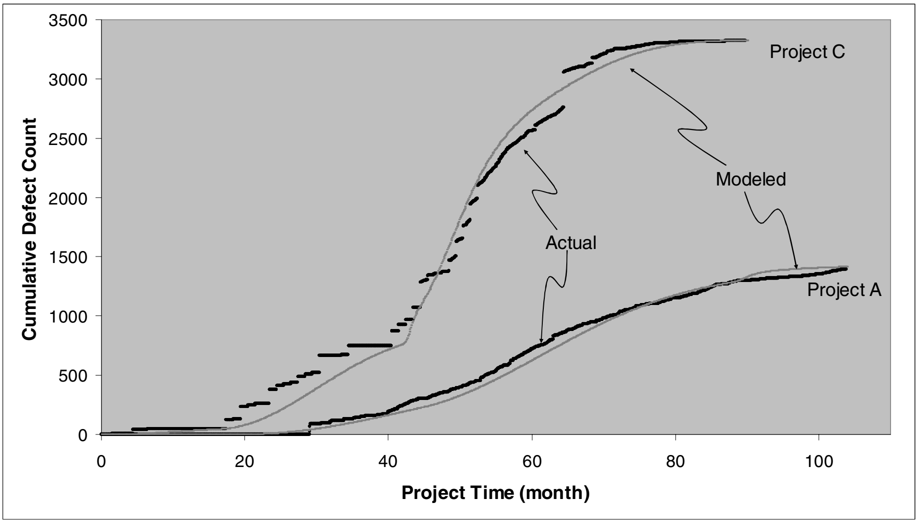


Fig. 5. Major Defect Discovery Profiles for Projects A & C, actual and modeled

7 Further Research and Usage

Work with DC is ongoing. It is being used by a software reliability expert to estimate the defect load that a project carries into system testing and field acceptance. We also plan to use it as part of a toolset for assessing the health of a project and the likely effects of introducing major changes, either desired or undesired. Application to more projects is expected to produce more facility with the tool, better understanding of software quality factors, and more accurate estimation of defect removal values.

References

1. Abdel-Hamid, T.K., Madnick, S.E.: Software Project Dynamics. Prentice Hall, Englewood Cliffs (1991)
2. Buettner, D.J.: Designing an Optimal Software Intensive System Acquisition: A Game Theoretic Approach. Doctoral Dissertation, University of Southern California (2008)
3. Card, D.N.: Managing Software Quality with Defects. Crosstalk 16(3), 4–7 (2003), `http://www.stsc.hill.af.mil/crosstalk/2003/03/mar03.pdf` (accessed July 22, 2008)
4. Choi, K.S., Bae, D.H.: COCOMO II-based dynamic software process simulation modeling method. Technical report CS-TR-2006-261, Computer Science Department, Korea Advanced Institute of Science and Technology, Daejeon, Korea (2006)
5. Creveling, C.M., Slutsky, J., Antis, D.: Design for Six Sigma in Technology and Product Development. Prentice Hall PTR, Upper Saddle River (2003)
6. Devnani-Chulani, S.: Results of Delphi for the Defect Introduction Model - Sub-Model of the Cost/Quality Model Extension to COCOMO II. Technical Report USC-CSE-97-504, University of Southern California (1997), `http://sunset.usc.edu/publications/TECHRPTS/1997/usccse97-505/usccse97-505.pdf` (accessed July 24, 2008)
7. Devnani-Chulani, S.: Modeling Software Defect Introduction and Removal: COQUALMO (COnstructive QUALity MOdel). Technical Report USC-CSE-99-510, University of Southern California (1999), `http://sunset.usc.edu/publications/TECHRPTS/1999/usccse99-510/usccse99-510.pdf` (accessed July 23, 2008)
8. Devnani-Chulani, S.: Bayesian Analysis of Software Cost and Quality Models. Doctoral Dissertation, University of Southern California (May 1999)
9. Gaffney, J.: Some Models for Software Defect Analysis. In: Lockheed Martin Software Engineering Workshop, Gaithersburg, Maryland (November 1996)
10. Kan, S.H.: Models and Metrics in Software Quality Engineering, 2nd edn. Addison-Wesley, New York (2003)
11. Modroiu, E.R., Schieferdecker, I.: Defect Rate Profile in Large Software-Systems. In: Tyugu, E., Yamaguchi, T. (eds.) Proc. of the 7th Joint Conference on Knowledge-Based Software Engineering. IOS Press, Amsterdam (2006)
12. Madachy, R., Boehm, B.: Assessing Quality Processes with ODC COQUALMO. In: Wang, Q., Pfahl, D., Raffo, D.M. (eds.) ICSP 2008. LNCS, vol. 5007, pp. 198–209. Springer, Heidelberg (2008); R. Madachy also discusses the ODC COQUALMO model in Software Process Dynamics. Wiley & Sons, New Jersey (2008)
13. Putnam, L.H., Myers, W.: Familiar Metric Management—Reliability (1995), `http://www.qsm.com/fmm_03.pdf` (accessed July 21, 2008)

14. Schick, G.J., Wolverton, R.W.: An Analysis of Competing Software Reliability Models. IEEE Transactions on Software Engineering 4(2), 104–120 (1978)
15. Seider, R.: Implementing Phase Containment Effectiveness Metrics at Motorola. Crosstalk 19(11), 12–14 (2006), `http://www.stsc.hill.af.mil/crosstalk/2006/11/index.html` (accessed July 22, 2008)
16. Tawileh, A., McIntosh, S., Work, B., Ivins, W.: The Dynamics of Software Testing. In: Proceedings of the 25th System Dynamics Conference, July 29- August 2. MIT, Boston (2007), `http://systemdynamics.org/conferences/2007/proceed/papers/TAWIL320.pdf` (accessed October 7, 2008)
17. Thangarajan, M., Biswas, B.: Mathematical Model for Defect Prediction across Software Development Life Cycle (2000), `http://www.qaiindia.com/conferences/SEPG2000/Selected/Best%20Practices/M%20Thangarajan%20&%20Biswajit%20Biswas.doc` (accessed July 22, 2008)
18. Trachtenberg, M.: Discovering how to ensure software reliability. RCA Engineer, 53–57 (January–February 1982)
19. Trachtenberg, M.: A General Theory of Software-Reliability Modeling. IEEE Transactions of Reliability 39(1), 92–96 (1990)

Appendix

Trachtenberg (1990) lists the assumptions underlying use of the Rayleigh distribution:

1. The workload is constant; it is neither increasing nor decreasing during the time covered by the distribution.
2. Processing is uniformly distributed across the workload. For defect generation, this implies creation of a uniform error density; for error removal, uniform opportunity for error discovery.
3. Average-errors “size” increases linearly with time. That is, the probability of introducing the “remaining” errors increases during defect generation, and the probability of finding remaining errors increases during defect removal.
4. Discovered defects are removed immediately and no additional defects are introduced (negligible bad fix rate).

At a project level, these may be violated to varying degrees. Examples are:

1. Workload continually grows due to scope creep, or functionality is significantly cut after the project is partially completed.
2. A significant proportion of development is reused software, or testing either targets or avoids risky functional areas.
3. Neither product complexity nor schedule pressure increase during generation activities. Defect removal efficiency decreases over time.
4. A substantial portion of major defects are not fixed in the phase found, or the fixes either don’t work or produce undesirable side effects.

Similar activity-level conditions can be listed as reasons that a Rayleigh distribution may not adequately characterize defect generation or defect removal activities.

1. Workload fluctuates significantly during the phase, for example required functional capability grows 50% during the requirements definition phase.

2. The workload is not processed uniformly, for example some design work is performed in great detail while other parts of design are left at a high level relative to implementation.
3. The probabilities of introducing and removing remaining errors decrease, for example a high turnover in testing staff produces inefficient testing.
4. Each product generation or defect removal activity is not distinct but is interleaved with other activities, for example code is integrated as it is produced.
5. Defect generation and removal rates are distorted by high rates of bad fixes.

To the degree that the foregoing conditions are can be avoided in a project, Rayleigh curves provide better models of defect introduction and removal. However, some conditions that undermine Rayleigh assumptions are desirable, for example continuous integration with daily builds, or incremental deliveries. For such cases, Dynamic COQUALMO provides currency parameters for modifying curve offset.

A Hybrid Model for Dynamic Simulation of Custom Software Projects in a Multiproject Environment

Javier Navascués[1], Isabel Ramos[2], and Miguel Toro[2]

[1] Universidad de Sevilla,
Departamento de Organización Industrial y Gestión de Empresas,
Av. Descubrimientos s/n, 41092 Sevilla, Spain
jnavascues@us.es
http://io.us.es

[2] Universidad de Sevilla,
Departamento de Lenguajes y Sistemas Informáticos,
Av. Reina Mercedes s/n, 41011 Sevilla, Spain
iramos@us.es, mtoro@us.es

Abstract. This paper describes **SimHiProS**, a hybrid simulation model of software production. The goal is to gain insight on the dynamics induced by resource sharing in multiproject management. In order to achieve it the hierarchy of decisions in a multiproject organization is modeled and some resource allocation methods based on algorithms from the OR/AI domain are used. Other critical issues such as the hybrid nature of software production and the effects of measurement and control are also incorporated in the model. Some first results are presented.

Keywords: Hybrid simulation, multiproject resource management, hierarchical decision making.

1 Introduction

Dynamic simulation of software projects is a well established field of research and application. Software project simulation models have been able to provide significant insight on many characteristics of the software production process. Nevertheless some real-world settings do not lend themselves easily to existing models. One particular question is the problem of resource allocation to projects in the case of software developed by multiproject organizations, i.e. software built by organizations working simultaneously in several projects for different customers with shared resources. Depending on the circumstances, this kind of software projects can become extremely risky as the intrinsic uncertainty of each particular project compounds with the mutual dependency between projects.

The research work presented here aims to develop modeling tools adequate to one of these cases: a medium-sized software engineering company whose business consists in developing custom software projects for State agencies with a

Q. Wang et al. (Eds.): ICSP 2009, LNCS 5543, pp. 173–185, 2009.

relatively stable set of human resources. The work adapts state-of-the-art models and methods both from the field of Software Engineering and from Project Management. In particular it builds on dynamic models aimed at the simulation of software production on one hand, and constrained resource project scheduling algorithms on the other.

The rest of this paper is organized as follows: Section 2 presents the critical issues this research tries to address. Section 3 comments on previous work carried by the academic community which has been revised and, when appropriate, used to build the model. Section 4 introduces the model, its main features and its operation. Section 5 presents some of the first results, draws conclusions and presents the limitations of the model in its current state. Section 6 announces future work.

2 Critical Issues in Custom Software Projects

The aim of this research is to gain understanding in resource allocation methods to software development in a multiproject context. Although it is focused on a concrete software company the class of problems studied are relevant for some branches within the software industry. Four are the issues that have been judged as being critical for the case under study:

1. Uncertainty owing to the combined effect of individual project risk and resource sharing
2. Coexistence of different levels of decision in a multiproject environment
3. The need to account both for continuous and discrete features in software production simulation
4. The trade-offs of implementing more or less tight measurement and control processes and SPI programmes in this context.

2.1 Resource Allocation and Uncertainty

Custom software developed for governmental agencies consists mostly in applications and functionalities which run on top of or interface with large existing legacy systems. This means that the developers must know and understand not only the project they are involved in but also the system or systems with which it will interact. Additionally in these cases the development model is usually incremental and evolutionary as opposed to waterfall models. Another particular feature of this kind of projects is what can be termed as *volatility of requirements* meaning that requirements are a source of uncertainty through the whole development cycle due to the complexity of the larger systems involved and of interactions which cannot easily be forecast in advance.

All these circumstances call for development teams composed by (at least a core of) experienced personnel who must remain committed to the project throughout the whole development cycle. To make things worse, these projects are prone to waiting periods owing to the number of stakeholders concerned and the complexity of decision making involved. Ideally the development team should not remain idle during these periods but this could affect productivity by switching between tasks.

This are the kind of circumstances which determine the real difficulties of managing this kind of projects, both at the single project level when coping with the perturbations inherent to its development or induced by other projects, and at the company level when facing the problem of dynamically assigning and reassigning staff to meet delivery dates while minimizing the idle periods, keeping a check on multitasking and - of course - avoiding excesses of personnel.

2.2 Multiple Decision Levels in a Multiproject Environment

In [9] the multiproject management problem is viewed under two different optics: the first focuses on the decision scope traditionally classified as *strategical*, *tactical* or *operational*. The second framework views multiproject environment in terms of *variability* and *dependency*. An organization specializing in custom software development faces a significant degree of variability at the project level. If resources are shared across projects, then a great dependency appears at is has been described in the previous paragraph. A rational management strategy typically will try to *reduce dependency between projects at the operational level coping with it at the tactical one.*

Accepting a project is an strategical decision and is left out of the problem under consideration. So projects appear earmarked with their intermediate and final deadlines conceptualized in terms of time windows. The problem then becomes:

- At the *tactical* level; allocate resources to each project so that delivery dates are respected. This should be attained using the existing capacity in the most economic possible way within quality standards. It is then a *tactical capacity planning* problem minimizing costs and respecting both temporal constraints and previous allocations.
- At the *operational* level; schedule each project within the margins imposed by the tactical planning providing for the intrinsic variability affecting each one. It is then a *resource constrained project scheduling* problem with *general precedence relations*(GRP-RCPSP) [4,6] under uncertainty where the objective function is minimizing makespan.

2.3 Projects and Processes: The Hybrid Nature of Software Production

The life-cycle of a software project is populated by concrete artifacts built, verified and validated by concrete agents. These entities are discrete in number and different from each other even if they can be grouped in categories. On the other hand the concept of process embodies the idea of commonalities among entities within categories. This allows for the possibility of aggregation into continuous magnitudes of some quantifiable features of software development. To account for both views, the discrete and the continuous one, using a hybrid approach to modeling is needed.

When the issue is the production of *custom* software in a multiproject environment the need for hybrid modeling becomes even greater. Custom software projects are essentially distinct from each other so many product metrics cannot accumulate nor average; but custom software projects developed in a multiproject environment draw the workforce used from a common manpower stock and on the other hand the current quality improvement paradigm relies on the notion of *process* associated to the idea of repeatability end, thus, akin to continuity.

2.4 Software Process Management and Improvement

A difficulty arising when facing a poorly structured or inmature process is that it does not lend itself easily to formal modeling. The mechanisms which can explain the performance remain hidden behind a conglomerate of informal and opaque practices. The very effort of modeling becomes a sort of diagnosis of the current state and can actuate as a first step toward process improvement. When simulating it seems reasonable, instead of taking for granted the existence of formal metrics and procedures to acquire them, try to model how this is done.

3 Previous Work

Software project simulation is an active research field originating in the seminal work by Abdel-Hamid and Madnick [2] based on System Dynamics which has been refined and enriched with many contributions and integrated with other lines of research and simulation paradigms.

From a professional and academic point of view the major reference consists in the series of PROSIM workshops (*Workshop of Software Process Simulation and Modeling*) starting in 1998 which became in 2006 the SPW (*Software Process Workshop*), and later a special track of ICSP, an ICSE (*International Conference on Software Engineering*) co-located event. A recent review by [21] provides a comprehensive outline of the kind of problems that are researched and the methodological approaches used. When the research here presented was in its final stage, a book by Madachy [12] was published in which the state of the art of continuous simulation of the software process is thoroughly presented.

3.1 Multiproject and Incremental Models

It is worthwhile mentioning that even though as soon as 1993 Abdel-Hamid [1] pointed out that estimation models and, by extension, simulation models would be of little use if the implications of staffing at the whole organization level are not taken into account, the amount of work produced relating to multiproject settings is quite scarce. Recently published work [21] shows that this gap in research remains to be filled. In a review of the proceedings of the before mentioned workshops between 1998 and 2007 only one out of 61 papers is classified as *multi-project* in scope. Another paper from the 2008 edition [7] focused entirely on multiproject resource allocation literature, points only six cases of simulation models.

Most published multiproject models, on the other hand, explicitly model a particular instance of a multiproject or incremental setting but they are not intended to model *any* configuration nor superposition of projects. An interesting exception is constituted by Powell *et al.* [16], a System Dynamics model of concurrent development. The authors propose an abstracted model connecting resources, time and effort defined in a modular way at several hierarchical levels: work package, phase, deliverable, project and the organization.

3.2 Simulation and Advanced Methods for Resource Allocation

Most approaches combining simulation and advanced methods for resource allocation between several projects are oriented either to assessing Operation Research based solutions using Monte Carlo simulation or simulating the state space where a heuristic method is used to find a 'good enough' solution.

The first approach, as shown in [3,13], requires statistical chraracterizing of the problem and it is oriented basically toward risk assessment. Lee and Miller [11] is another example combining dynamic simulation with project management techniques. Padberg [15] presents a line of work in which schedules are generated through an approximate *dynamic programming* algorithm optimized over a subspace of the whole solution space. Neither of the two approaches consider dynamic reallocation policies so in fact decision making is not simulated.

To simulate decision making resource allocation should change dynamically in respisne to certain events. This is implementeted by Özdamar [19] who employs priority rules in projects defined in *fuzzy* terms. Another interesting example is Joslin and Poole [10] who present an agent-based model which simulates dynamically the assignment of staff to a project with several functionalities that must be delivered within previously fixed deadlines.

3.3 Hybrid Models

During the last decade, hybrid simulation has been one of the most frequent research themes. Most of the work is based on the combination of continuous (System Dynamics) and discrete-event simulation.

Rus et al. [18] and Martin and Raffo [17] are examples where the work environment, productivity and resources are treated as continuous variables while the dynamics of the work products and artifacts is presented as essentially discrete. The model presented in [14] follows the same scheme although it also implements a hierarchical approach when representing global software development.The model presented by Donzelli and Iazolla [8] treats the work process as basically continuous and resources as queue servers which obviously are discrete.

The DEVS formalism established by Ziegler [20] is used by Choi *et al.* [5] in what seems the more methodologically consistent proposal for hybrid modeling at least as far as the authors of this paper have been able to identify. Based on this formalism the cited authors formulate a model obviously inspired in the classical one by Abdel-Hamid and Madnick [2] in which the usage of an adequately small time step and the so called QSS - *quantized-state system* - as an alternative to interpolation for numerical integration provides at least theoretically a natural way to hybrid simulation.

4 The SimHiPros Model

SimHiProS is an hybrid model which simulates the production of software in a multiproject environment through a hierarchy starting from the most elementary work processes up to the wohle organization. The first version has been coded in Modelica simulation language and implemented in a MathModelica[1] simulation environment. The hierarchy is instantiated through a modular architecture comprising three levels: *package*, *project* and *multiproject*. Apart from these hierarchically encapsulated modules which represent the hierarchy of the production process the model comprises a module of *environment* which allows to simulate the dynamics of the workforce in the organization and a *functional* module which implements functions and algorithms and is called by the former modules. Each of the three hierarchically ordered modules has got three components: *activity*, *allocation* and *measure and control*. The block diagrams presented in figures 1 and 2 represent these components as blue, orange and green boxes respectively.

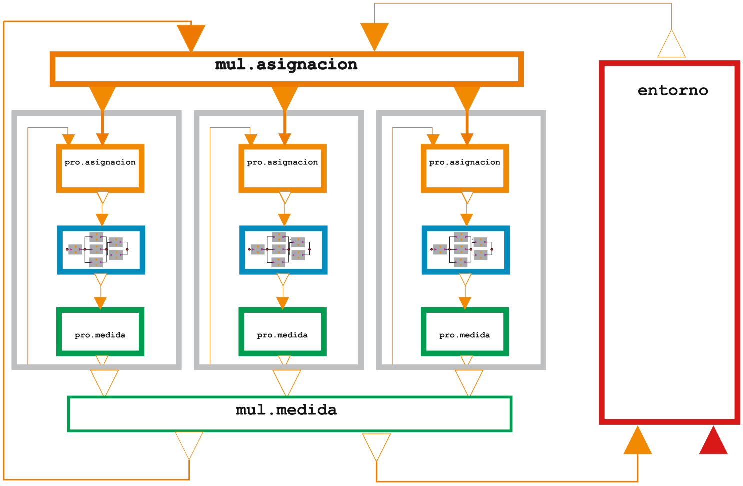


Fig. 1. Block diagram of the general architecture of SimHiProS

4.1 The Basic Level: Package Module

The *package module* simulates the work process as a number of elementary work units, *tasks*, which must be carried out to obtain a certain artifact. The *package size* is the number of tasks that must be completed. The *activity* component within the package simulates the continuous work process as tasks moving

[1] MathModelica is a Trademark of MathCore Engineering AB.

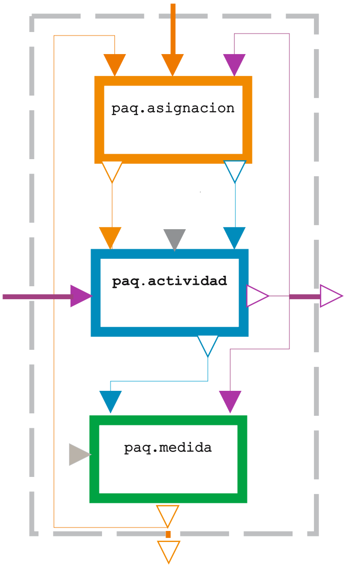


Fig. 2. The *package* module

through four consecutive *backlogs*: initial tasks, tasks to build, tasks to verify and eventually tasks to iterate and accumulating in a finished tasks level. The rates represent the actual work in preparation, construction, verification, error detection and correction and are determined by the resources allocated and the productivity of these resources. The productivity and the fraction of tasks (error production) to iterate is fixed in the current implementation although both are typically modeled as non-linear phenomena in state-of-the-art models. These non linearities have been intentionally set aside because the emphasis in the current model has been set upon the dynamic effects of the allocation policies.

The *allocation* component at the package level distributes the resources available to the package, which are themselves assigned at the *project* level, to the different rates acting upon the backlogs. Several elementary intra-package allocation methods have been implemented such as a FIFO dispatching rule or allocating personnel in proportion to pending backlogs. The assigned resources to each rate vary in a discrete manner in a double sense: the assignments do not change continuously but at fixed time intervals and the number of resources (developers, engineers, ...) assigned is always integer.

The *measurement* component of this module samples at fixed intervals the levels (backlogs) and records the resource usage. These data are used by the *allocation* component and are also passed to the *project* module to be used as is described in the following paragraph.

4.2 The Operational Level: The Project Module

At the operational level, the *activity* component is a *network of packages* (see figure 3) logically connected through precedence relations representing the sequence

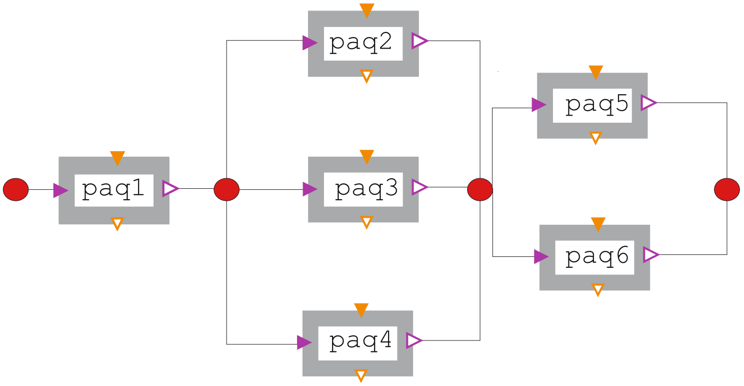


Fig. 3. Activity component of the *project* module

of artifacts in which the whole development process is organized. A package is activated when the logically preceding artifacts are complete. This gives the model enough flexibility to represent different WBS (work breakdown structures) and even different development cycle models.

The *measurement* component gathers the output of the corresponding components at the package level and calculates project metrics designed following an Earned Value Management framework. This metrics are passed on to the tactical (multiproject) level and are also used to activate the reallocation algorithms if necessary. In order to trigger the reassignment procedures, the *measurement* component makes estimates of the finish dates and compares them to the planned values. In case the deviation grows bigger than a previously set value, resources are reassigned.

The *allocation* component distributes between packages the resources allocated to the current project following a *squeaky wheel optimization* algorithm aiming to reduce makespan subject to time-window constraints. If this algorithm is not able to find a solution respecting the due dates, a message is passed to the tactical level to allow for resource reallocation across projects. The non-linear effects of schedule pressure on productivity and error generation which have been laid apart in the initial implementation of the model would naturally be modeled within this module as the relevant factors pertain to the project level.

4.3 The Tactical Level: Multiproject Module

At the tactical level, the *activity* component is made up of the projects currently active in the organization. The *measurement* component keeps track of the metrics of each individual project and their deviations and provides signals to the *allocation* component which reassigns resources between projects.

4.4 The Environment and the Functional Modules

The *environment* module models in a traditional System Dynamics fashion the evolution of human resources in terms of new personnel hiring, dismissals,holiday or illness leaves, or resources temporarily unavailable due to infrastructural tasks, training schemes, etc. Although effective personnel is modeled in an homogeneous category it is possible to divide this into several career levels accounting for different degrees of experience and maturity.

The *functional* module contains the different functions called by the *allocation components* thus implementing the resource allocation algorithms and other auxiliary routines such as extrapolation of forecast values, precedence rules maintenance, WBS network topology description, etc.

4.5 Operation of the Model

The model is initialized with the projects under study, their WBS and size; the initial resource allocation provided by the tactical planning; and a forecast for staff demography (new contracts, people on leave, ...). In absence of any disruption, the model will follow the planned course allocating resources to projects following the tactical plan, and to work packages within projects following the operational model. In this sense, each project (itself an instantiation of the project module) actuates as an autonomous agent. All this is constrained by the dynamics of available staff simulated by the environment module.

Once the model has completed a base run, the results translate into planned time profiles for the metrics of each individual project. These values are recorded and experiments of disruptions can be carried out. These experiments consist in unexpected events affecting staff (such as people leaving in the middle of a project) of affecting projects (new requirements, errors discovered, ...). The operational agent will try to mend the schedule within the corresponding project initially by expediting all feasible resources to the affected tasks and subsequently exploring better alternatives with the randomized *squeaky wheel algorithm.* In case it is not enough, a message will be passed to the tactical level. In this case, the tactical plan is repaired with a simple neighborhood heuristics.

5 First Results, Conclusions and Limitations of the Model

5.1 First Results

The model's parameters have been estimated with data obtained from a sample of projects carried out by a middle sized company working for public agencies in Spain. In fact the model was developed to gain insight in the production management problems of this company. With this parameters and a set of 'stylized' events representative of the kind of disruptions the projects are liable to, a series of simulation experiments has been carried out.

Figures 5 and 6 show the responses of the model in a simplified case concerning two projects (whose activity networks are depicted in figure 4). In the first case there is a reassignment between activities within the first project. The second figure shows what happens when the resources assigned to the first project are not enough to repair the schedule and they must be borrowed from the second project.

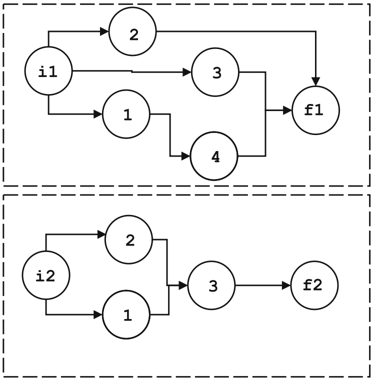


Fig. 4. Two sample projects on *activity on node* notation

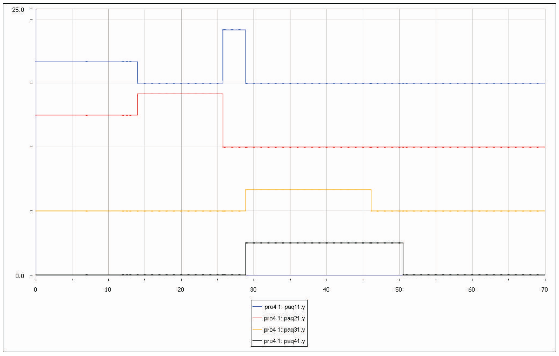


Fig. 5. Resource reassignment between activities of the first sample project

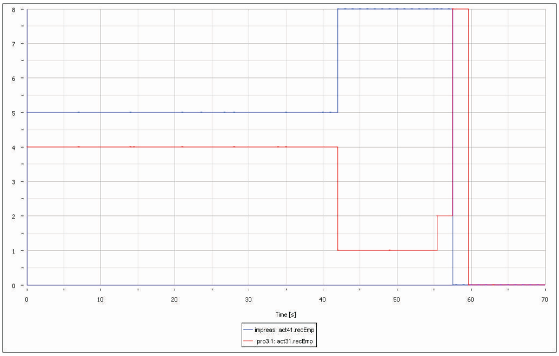


Fig. 6. Resource reassignment between the two sample projects

5.2 Preliminary Conclusions

Although the model is still in a first version, some conclusions can be drawn in the sense that the features which were conceptualized as critical for the kind of software projects under study have been successfully incorporated to the model. In particular the model represents, to the best of our knowledge, a step forward in the state of the art of dynamic simulation of the software process at least in the following issues:

- It is a *naturally* hybrid model based on DAE *differential algebraic equations* with a numerical solver which manages the integration step dynamically allowing for very short intervals in the vicinity of discrete events
- It has got a modular architecture allowing for simulation of various life-cycle models (waterfall, incremental, evolutionary, ...) and capable of representing hierarchies
- It implements three levels of hierarchically differentiated decision making levels (work package, project and multiproject)
- The resource allocation decisions at project and multiproject level are dynamically activated by the results of the simulation and are based on algorithms and models from the OR/AI community
- Measurement effort is explicitly modeled and associated costs are recorded and accumulated as a first step towards assessing the trade-offs of control policies.

5.3 Limitations of the Model

In its current version, the model has got the following limitations:

- The programming language is a declarative one and its algorithmic possibilities are not very reaching so the model implements relatively simple allocation algorithms; at the operational level serial generation schemes based on priority rules and SWO, at the tactical level a very simple neighborhood heuristic.
- Many state-of-the-art non linearities have not been implemented: such as the impact of schedule pressure on productivity and percentage of errors, communication overheads, ... These effects have intentionally been laid apart because the research interest was focused on dynamics provoked by resource sharing, but the model is prepared to incorporate them.
- The WBS of the projects remain static during the simulation although the work contents of any particular task can change; this way a change in the structure of a project which can be the outcome of a decision by the project manager in response to some disruption cannot be simulated.

6 Further Developments

Further developments planned consist basically in overcoming some of the limitations stated above and exploiting the model's features and capabilities as a support tool for the concrete project management problems under consideration in incumbent company.

In the first strand, the immediate tasks will be incorporating the results of previous work concerning non-linearities in relation to productivity, quality, ... to the model. Secondly, implementing more sophisticated algorithms of resource scheduling. The Modelica language is oriented toward scientific simulation of models of physical and/or technical systems and lacks powerful algorithmic features. The only possibility to employ more sophisticated procedures is through calls to C++ or FORTRAN coded functions.

A more ambitious goal is developing the possibility of dynamical reconfiguration of the WBS of projects as response to disruptions, something that is consistent with rational management practices but it is very difficult to implement with the tools used up to now. Both this and the afore mentioned issue of the algorithmic power calls for a reconsideration of the modeling language.

Concerning the usage of the model as a tool for production management a debate is currently under way with the incumbent company on how to evolve the model to integrate it with the current project control system to use it in a Decision Support Tool for project management.

Acknowledgments

The work here presented has been supported by the Spanish Ministry of Science and Innovation grants no. TIN2004-06689-C03-03 and TIN2007-67843-C06-03. The authors wish to thank the useful remarks and suggestions received from three anonymous reviewers of a previous version of this paper.

References

1. Abdel-Hamid, T.K.: A multiproject perspective of single-project dynamics. Journal of Systems and Software 22(3), 151–165 (1993)
2. Abdel-Hamid, T.K., Madnick, S.E.: Software Project Dynamics An Integrated Approach. Prentice-Hall, Englewood Cliffs (1991)
3. Antoniol, G., Cimitile, A., Di Lucca, G.A., Di Penta, M.: Assessing staffing needs for a software maintenance project through queuing simulation. IEEE Transactions on Software Engineering 30(1), 43–58 (2004)
4. Brucker, P., Drexl, A., Möhring, R., Neumann, K., Pesch, E.: Resource-constrained project scheduling: Notation, classification, models, and methods. European Journal of Operational Research 112(1), 3–41 (1999)
5. Choi, K., Bae, D., Kim, T.: An approach to a hybrid software process simulation using the DEVS formalism. Software Process: Improvement and Practice 11(4), 373–383 (2006)
6. Demeulemeester, E.L., Herroelen, W.: Project scheduling, vol. 49. Kluwer Academic Publishers, Boston (2002)
7. Dong, F., Li, M., Zhao, Y., Li, J., Yang, Y.: Software Multi-project Resource Scheduling: A Comparative Analysis. In: Wang, Q., Pfahl, D., Raffo, D.M. (eds.) ICSP 2008. LNCS, vol. 5007, pp. 63–75. Springer, Heidelberg (2008)

8. Donzelli, P., Iazeolla, G.: Hybrid simulation modelling of the software process. The Journal of Systems and Software 59(3), 227–235 (2001)
9. Herroelen, W.: Project scheduling-theory and practice. Production and Operations Management 14(4), 413 (Winter 2005)
10. Joslin, D., Poole, W.: Agent-based simulation for software project planning. In: Winter Simulation Conference, pp. 1059–1066 (2005)
11. Lee, B., Miller, J.: Multi-project management in software engineering using simulation modelling. Software Quality Journal 12, 59–82 (2004)
12. Madachy, R.J.: Software process dynamics. Wiley, IEEE Press, Hoboken, Piscataway (2008)
13. Meier, C., Yassine, A.A., Browning, T.R.: Design process sequencing with competent genetic algorithms. Transactions of the ASME 129, 566 (2007)
14. Setamanit, S., Wakeland, W., Raffo, D.: Planning and improving global software development process using simulation. In: GSD (2006)
15. Padberg, F.: On the potential of process simulation in software project schedule optimization. In: 29th Annual International Computer Software and Applications Conference, 2005. COMPSAC 2005, vol. 2, pp. 127–130 (2005)
16. Powell, A., Mander, K., Brown, D.: Strategies for lifecycle concurrency and iteration â a system dynamics approach. Journal of Systems and Software 46(2-3), 151–161 (1999)
17. Raffo, D., Martin, R.H.: A model of the software development process using both continous and discrete models. Software Process Improvement and Practice 5, 147–157 (2000)
18. Rus, I., Collofello, J., Lakey, P.: Software process simulation for reliability management. The Journal of Systems and Software 46(2–3), 173–182 (1999)
19. Özdamar, L., Alanya, E.: Uncertainty modelling in software development projects (with case study). Annals of Operations Research 102(1-4), 157–178 (2001)
20. Zeigler, B.P., Kim, T.G., Praehofer, H.: Theory and practice of modeling and simulation. Academic Press, New York (2000)
21. Zhang, H., Kitchenham, B., Pfahl, D.: Reflections on 10 Years of Software Process Simulation Modeling: A Systematic Review. In: Wang, Q., Pfahl, D., Raffo, D.M. (eds.) ICSP 2008. LNCS, vol. 5007, pp. 345–356. Springer, Heidelberg (2008)

On the Relative Merits of Software Reuse

Andres Orrego[1,2], Tim Menzies[2], and Oussama El-Rawas[2]

[1] Global Science & Technology, Inc., Fairmont, WV, USA
andres.orrego@gst.com
[2] West Virginia University, Morgantown, WV, USA
tim@menzies.us, oelrawas@mix.wvu.edu

Abstract. Using process simulation and AI search methods, we compare software reuse against other possible changes to a project. such as reducing functionality or improving the skills of the programmer population. In one case, two generations of reuse were as good or better than any other project change (but a third and fourth generation of reuse was not useful). In another case, applying reuse to a project was demonstrable worse than several other possible changes to a project.

Our conclusion is that the general claims regarding the benefits of software reuse do not hold for specific projects. We argue that the merits of software reuse need to be evaluated in a project by project basis. AI search over process models is useful for such an assessment, particularly when there is not sufficient data for precisely tuning a simulation model.

Keywords: Software Reuse, COCOMO, COQUALMO, AI search.

1 Introduction

We need to better understand software reuse. In theory, reuse can lower development cost, increase productivity, improve maintainability, boost quality, reduce risk, shorten life cycle time, lower training costs, and achieve better software interoperability [1,2]. However, in practice, studies have shown that reuse is not always the best choice: it may be hard to implement, and the benefits of reuse cannot be reliably quantified [1]. Also, in some cases, reuse has resulted in economic loses [3] and even personal injury and loss of life [4].

Process simulations can be used to assess the value of reuse in a particular project. Traditionally, such simulators are commissioned using using data collected from a particular organization (e.g. [5]). Often, such local data is hard to collect. Accordingly, we have been been exploring an AI method called STAR that reduce the need for calibration from local data. To understand STAR, note that project estimates are some function of the project options and the internal model calibration variables. Conceptually, we can write this as:

$$estimate = project * calibration$$

The estimate variance is hence a function of *variance in the project options* and the *space of possible calibrations*. Traditional approaches use historical data to reduce the

Q. Wang et al. (Eds.): ICSP 2009, LNCS 5543, pp. 186–197, 2009.

Reuse Iteration	**Description**
First reuse	Using software from a previous project for the first time.
Second reuse	Reusing software from a previous project for the second time
Third reuse	Reusing the same software in a new project for the third time.
Fourth reuse	Reusing software using a mature reuse approach, tools, and personnel.

Fig. 1. Process changes imposed by implementing reuse incrementally

Drastic change	**Possible undesirable impact**
1 Improve personnel	Firing and re-hiring personnel leading to wide-spread union unrest.
2 Improve tools, techniques, or development platform	Changing operating systems, IDEs, coding languages
3 Improve precedentness / development flexibility	Changing the goals of the project and the development method.
4 Increase architectural analysis / risk resolution	Far more elaborate early life cycle analysis.
5 Relax schedule	Delivering the system later.
6 Improve process maturity	May be expensive in the short term.
7 Reduce functionality	Delivering less than expected.
8 Improve the team	Requires effort on team building.
9 Reduce quality	Less user approval, smaller market.

Fig. 2. Nine drastic changes from [9]

space of possible calibrations (e.g. using regression). In our approach, we leave the calibration variables unconstrained and instead use an AI search engine to reduce the space of possibilities in the project options. In numerous studies (including one reported last year at ICSP'08 and elsewhere [6,7,8]) we showed that this methods can yield estimates close to those seem using traditional methods, without requiring a time consuming data collection exercise.

In this paper, we use STAR to comparatively assess 13 possible changes to a project. Figure 1 shows four changes to a project based on reuse while Figure 2 defines some alternatives. These alternatives are drastic changes a project manager could implement in an effort to reduce effort, schedule and defects in a particular project [9]. Our results will show that some projects gain the most benefit from applying reuse, while there are often other changes (such as those listed in Figure 2) that can be more effective. Hence, we recommend assessing the value of reuse on a project-by-project basis. Process simulation tools are useful for making such an assessment and tools like STAR are especially useful when there is insufficient data for local calibration.

In the remainder of this paper, we present background information about software reuse and process estimation models. Then we document the simulation approach utilized to evaluate the effects of adopting software reuse compared to alternative strategies for two NASA case studies.

2 The Models: COCOMO and COQUALMO

For this study we use two USC software process models. The COQUALMO software *defect* predictor [10, p254-268] models two processes (defect introduction and defect removal) for three phases (requirements, design, and coding). Also, the COCOMO software *effort* and development *time* predictor [10, p29-57] estimates development months

	Definition	Low-end = {1,2}	Medium ={3,4}	High-end= {5,6}
Defect removal features				
execution-based testing and tools (etat)	all procedures and tools used for testing	none	basic testing at unit/ integration/ systems level; basic test data management	advanced test oracles, assertion checking, model-based testing
automated analysis (aa)	e.g. code analyzers, consistency and traceability checkers, etc	syntax checking with compiler	Compiler extensions for static code analysis, Basic requirements and design consistency, traceability checking.	formalized specification and verification, model checking, symbolic execution, pre/post condition checks
peer reviews (pr)	all peer group review activities	none	well-defined sequence of preparation, informal assignment of reviewer roles, minimal follow-up	formal roles plus extensive review checklists/ root cause analysis, continual reviews, statistical process control, user involvement integrated with life cycle
Scale factors:				
flex	development flexibility	development process rigorously defined	some guidelines, which can be relaxed	only general goals defined
pmat	process maturity	CMM level 1	CMM level 3	CMM level 5
prec	precedentedness	we have never built this kind of software before	somewhat new	thoroughly familiar
resl	architecture or risk resolution	few interfaces defined or few risks eliminated	most interfaces defined or most risks eliminated	all interfaces defined or all risks eliminated
team	team cohesion	very difficult interactions	basically co-operative	seamless interactions
Effort multipliers				
acap	analyst capability	worst 35%	35% - 90%	best 10%
aexp	applications experience	2 months	1 year	6 years
cplx	product complexity	e.g. simple read/write statements	e.g. use of simple interface widgets	e.g. performance-critical embedded systems
data	database size (DB bytes/SLOC)	10	100	1000
docu	documentation	many life-cycle phases not documented		extensive reporting for each life-cycle phase
ltex	language and tool-set experience	2 months	1 year	6 years
pcap	programmer capability	worst 15%	55%	best 10%
pcon	personnel continuity (% turnover per year)	48%	12%	3%
plex	platform experience	2 months	1 year	6 years
pvol	platform volatility ($\frac{frequency\ of\ major\ changes}{frequency\ of\ minor\ changes}$)	$\frac{12\ months}{1\ month}$	$\frac{6\ months}{2\ weeks}$	$\frac{2\ weeks}{2\ days}$
rely	required reliability	errors are slight inconvenience	errors are easily recoverable	errors can risk human life
ruse	required reuse	none	multiple program	multiple product lines
sced	dictated development schedule	deadlines moved to 75% of the original estimate	no change	deadlines moved back to 160% of original estimate
site	multi-site development	some contact: phone, mail	some email	interactive multi-media
stor	required % of available RAM	N/A	50%	95%
time	required % of available CPU	N/A	50%	95%
tool	use of software tools	edit,code,debug		integrated with life cycle

Fig. 3. Features of the COCOMO and COQUALMO models used in this study

(225 hours) and calendar months and includes all coding, debugging, and management activities. COCOMO assumes that effort is exponentially proportional to some *scale factors* and linearly proportional to some *effort multipliers*.

From our perspective, these models have several useful features. Unlike other models such as PRICE-S [11], SLIM [12], or SEER-SEM [13], the COCOMO family of models is fully described in the literature. Also, at least for the effort model, there exist baseline results [14]. Also, we work extensively with government agencies writing software. Amongst those agencies, these models are frequently used to generate and justify budgets. Further, The space of possible tunings within COCOMO & COQUALMO is well defined. Hence, it is possible to explore the space of possible tunings.

The process simulation community (e.g., Raffo [15]) studies models far more elaborate than COCOMO or COQUALMO. For example, COCOMO & COQUALMO assume linear parametric equations while other researchers explore other forms:

- discrete-event models [16];
- system dynamics models [17];
- state-based models [18];
- rule-based programs [19];
- standard programming constructs such as those used in Little-JIL [20].

These rich modeling frameworks allow the representation of detailed insights into an organization. However, the effort required to tune them is non-trivial. For example, Raffo spent two years tuning and validating one of such models to one particular site [5]. Also, we have found that the estimation variance of COCOMO can be reduced via intelligent selection of input variables, even allowing for full variance in the tuning parameters. We would consider switching to other models if it could be shown that the variance of these other models could be controlled just as easily.

Our models use the features presented in Figure 3. This figure lists a variety of project *features* with the *range* $\{$very low, low, nominal, high, very high, extremely high$\}$ or $\{vl = 1, l = 2, n = 3, h = 4, vh = 5, xh = 6\}$. For specific projects, not all features are known with certainty. For example, until software is completed, the exact size of a program may be unknown. Hence, exploring our effort, schedule, and defect models requires exploring a large trade-space of possible model inputs.

2.1 Effect of Reuse on Model Parameters

The effects of ad-hoc software reuse can be mapped to changes to settings to the COCOMO parameters. For instance, programmers capability (pcap) inherently increases every time a piece of software is reused given that in the process the same programmer is employed. This is the case with NASA spacecraft software, where reuse can be found within the same software development company and where the software modules are signed by the same developers [21]. Similarly, we can assume direct inherent effects to the analyst capability (acap), the application experience (apex), the analyst capability (acap), the precedence of the software (prec), the process maturity (pmat), and the language and tool experience (ltex). On the other hand, the software platform must remain fairly unchanged throughout reuses so software pieces can be reused with ease. For this reason we assume that platform volatility has to decrease as a piece of software is reused from project to project.

A final assumption on the size of the software comes from the observation that the progressive reuse of software components allows the construction of more sizeable systems. For instance, when looking at NASA Space fligh software, new systems commonly result in bigger and more complex software, usually about 50% more lines of

Incremental Reuse	Effects on Projects

1 First Reuse	acap = ACAP$_L$; apex = APEX$_L$; pcap = PCAP$_L$; prec = PREC$_L$
	pmat = PMAT$_L$; ltex = LTEX$_L$; pvol = PVOL$_H$; kloc = KLOC$_L$

2 Second Reuse	acap = ACAP$_L$+1 ; apex = APEX$_L$+1 ; pcap = PCAP$_L$+1 ; prec = PREC$_L$+1
	pmat = PMAT$_L$+1 ; ltex = LTEX$_L$+1 ; pvol = PVOL$_H$-1 ; kloc = KLOC$_L$*1.25

3 Third Reuse	acap = ACAP$_L$+2 ; apex = APEX$_L$+2 ; pcap = PCAP$_L$+2 ; prec = PREC$_L$+2
	pmat = PMAT$_L$+2 ; ltex = LTEX$_L$+2 ; pvol = PVOL$_H$-2 ; kloc = KLOC$_L$ $* (1.25)^2$

4 Fourth Reuse	acap = ACAP$_L$+3 ; apex = APEX$_L$+3 ; pcap = PCAP$_L$+3 ; prec = PREC$_L$+3
	pmat = PMAT$_L$+3 ; ltex = LTEX$_L$+3 ; pvol = PVOL$_H$-3 ; kloc = KLOC$_L$ $* (1.25)^3$

Fig. 4. Implementing software reuse incrementally

code [22]. Of this kloc increase NASA developers may only be able to save a small percentage of the size of the code base by reusing components due to their ad-hoc, white-box reuse approach, which results in sizeable modification of the original code [23]. In our simulations we therefore assume that the code base increases from system to system by 25%.

Figure 4 shows the constraints we claim software reuse imposes on model parameters for a given project. In the figure the variable X_L represents the lowest value in the range of the model variable X. Similarly, X_H represents the highest value for the X variable. In order to impose the constraint of reuse in a particular project we increase the X_L and lower the X_H for the particular variable X according to the logic above. For instance, let's say that for a particular project P, pcap ranges between 2 and 4, and pvol ranges between 3 and 5. In this case, PCAP$_L = 2$, PCAP$_H = 4$, PVOL$_L = 3$, and PVOL$_H = 5$. if we imposed a "Second Reuse" strategy, pcap would be set to 3 (PCAP$_L + 1$), and pvol would be set to 4 (PVOL$_H - 1$).

2.2 Defining the Alternatives to Reuse

Similarly to the constraints imposed by the incremental software reuse strategies, Figure 5 defines the values we imposed on each case study as part of each drastic change. Most of the values in Figure 5 are self-explanatory with two exceptions. Firstly, the $kloc * 0.5$ in "reduce functionality" means that, when imposing this drastic change, we only implement half the system. Secondly, most of the features fall in the range one to five. However, some have minimum values of 2 or higher (e.g., $pvol$ in "improve tools/tech/dev"), and some have maximum values of 6 (e.g., $site$ in "improve tools/tech/dev"). This explains why some of the drastic changes result in values other than one or five.

To impose a drastic change on a case study, if that change refers to feature X (in the right-hand column of Figure 5), then we first (a) removed X from the values and ranges of the case study (if it was present); then (b) added the changes of Figure 5 as fixed values for that case study.

3 Case Studies

Recall that the goal of our study is to analyze simulations of process estimation using COCOMO and COQUALMO models on typical NASA projects. Our purpose is to evaluate the relative merits of adopting software reuse compared to other project improvement strategies. The comparison is based on effort, quality, and schedule measured in

Drastic change	Effects on Projects

1 Improve personnel	acap = 5; pcap = 5; pcon = 5
	apex = 5 ; plex = 5 ; ltex = 5

2 Improve tools, techniques, or development platform	time = 3; stor = 3
	pvol = 2; tool = 5
	site = 6

3 Improve precedentness / development flexibility	prec = 5; flex = 5
4 Increase architectural analysis / risk resolution	resl = 5
5 Relax schedule	sced = 5
6 Improve process maturity	pmat = 5
7 Reduce functionality	data = 2; kloc * 0.5
8 Improve the team	team = 5

9 Reduce quality	rely = 1 ; docu = 1
	time = 3 ; cplx = 1

Fig. 5. Implementing drastic changes

	ranges			values	
project	feature	low	high	feature	setting
JPL flight software	rely	3	5	tool	2
	data	2	3	sced	3
	cplx	3	6		
	time	3	4		
	stor	3	4		
	acap	3	5		
	apex	2	5		
	pcap	3	5		
	plex	1	4		
	ltex	1	4		
	pmat	2	3		
	KSLOC	7	418		

	ranges			values	
project	feature	low	high	feature	setting
JPL ground software	rely	1	4	tool	2
	data	2	3	sced	3
	cplx	1	4		
	time	3	4		
	stor	3	4		
	acap	3	5		
	apex	2	5		
	pcap	3	5		
	plex	1	4		
	ltex	1	4		
	pmat	2	3		
	KSLOC	11	392		

Fig. 6. Two case studies. Numeric values $\{1, 2, 3, 4, 5, 6\}$ map to $\{verylow, low, nominal, high, veryhigh, extrahigh\}$

person-months, defects per KSLOC, and months, respectively. In this study we explore the perspective of the Project Manager in the context of NASA software development.

Figure 6 partially describes the two NASA case studies we explore in terms of the COCOMO and COQUALMO input parameters. Both case studies reflect typical ranges seen at NASA's Jet Propulsion Laboratory [7].

Inside our model, project choices typically range from 1 to 5 where "3" is the nominal value that offers no change to the default estimate. Some of the project choices in Figure 6 are known precisely (see all the choices with single *values*). But many of the features in Figure 6 do not have precise values (see all the features that *range* from some *low* to *high* value).

We evaluate the effects of the project improvement strategies on the case studies above using STAR: a Monte Carlo engine augmented by a simulated annealer.

STAR runs as follows. First, a project P is specified as a set of min/max ranges to the input variables of STAR's models:

- If a variable is known to be exactly x, then then $min = max = x$.
- Else, if a variable's exact value is not known but the range of possible values is known, then min/max is set to the smallest and largest value in that range of possibilities.
- Else, if a variable's value is completely unknown then min/min is set to the full range of that variable in Figure 3.

Second, STAR's simulated annealer[1] seeks constraints on P that most improve the model's score. A particular subset of $P' \subseteq P$ is scored by using P' as inputs to the COCOMO and COQUALMO. When those models run, variables are selected at random from the min/max range of possible tunings T and project options P. In practice, the majority of the variables in P' can be removed without effecting the score; i.e. our models exhibit a *keys effect* where a small number of variables control the rest [25]. Finding that minimal set of variables is very useful for management since it reveals the *least* they need to change in order to *most* improve the outcome. Hence, simulated annealing, STAR takes a third step.

In this third step, a Bayesian sensitivity analysis finds the smallest subset of P' that most effects the output. The scores seen during simulated annealing are sorted into the (10,90)% (best,rest) results. Members of P' are then ranked by their Bayesian probability of appearing in $best$.

After ranking members of P', STAR then imposes the top i-th ranked items of P' on the model inputs, then running the models 100 times. This continues until the scores seen using $i + 1$ items is not statistically different to those seen using i (t-tests, 95% confidence). STAR returns items $1..i$ of P' as the *least* set of project decisions that *most* reduce effort, defects, and development time. We call these returned items the *policy*.

Note that STAR constraints the project options P but never the tuning options T. That is, the *policy* generated by STAR contains parts of the project options P that most improve the score, despite variations in the tunings T. This approach has the advantage that it can reuse COCOMO models without requiring local tuning data.

Previously [7] we have shown that this approach, that does not use local tuning, generates estimates very similar to those generated after using local tuning via the "LC" method proposed by Boehm and in widespread use in the COCOMO community [26]. We have explained this effect as follows. Uncertainty in the project options P and the tuning options T contribute to uncertainty in the estimates generated by STAR's models. However, at least for the COCOMO and COQUALMO models used by STAR, the uncertainty created by P dominates that of T. Hence, any uncertainty in the output can be tamed by constraining P and not T.

The reader may wonder why we use an stochastic method like STAR: would not a simpler method suffice? For example, Many of the relationships inside COCOMO model are linear and a simple linear extrapolation across the space of possibilities could assess the relative effectiveness of different changes. In reply, we note that:

- Even after tuning the gradient of the relationships may not be known with certainty. For example, in the COCOMO effort model, predictions are affected linearly and exponentially by two types of input parameters; the new project data and the historical dataset. In COCOMO this results in the coefficients, a and b, which define the relationship between size and effort. Baker [27] tuned these a, b values using

[1] Simulated annealers randomly alter part of the some $current$ solution. If this new solution scores better than the current solution, then $current = new$. Else, at some probability determined by a temperature variable, the simulated annealer may jump to a sub-optimal new solution. Initially the temperature is "hot" so the annealer jumps all over the solution space. Later, the temperature "cools" and the annealer reverts to a simple hill climbing search that only jumps to new better solutions. For more details, see [24].

data from NASA systems. After thirty 90% random samples of that data, the a, b ranges were surprisingly large: $(2.2 \leq a \leq 9.18) \wedge (0.88 \leq b \leq 1.09)$. Baker's results forced a rethinking of much our prior work in this area. Instead of exploring better learners for local calibration, now we use tools like STAR to search models for conclusions that persist across the space of possible calibrations.

- Simplistic linear extrapolation may be inappropriate when optimizing for effort *and* time *and* defects, there may be contradictory effects. For example, we have results where reducing effort leads to a dramatic increase in defects [8]. Hence, optimizing our models is not a simple matter of moving fixed distances over some linear effect: there are also some trade-offs to be considered (e.g. using a tool that considers combinations of effects, like STAR).

4 Results

For each case study of Figure 6, STAR searches within the ranges to find constraints that most reduce development effort, development time, and defects. The results are shown in Figure 7, Figure 8, and Figure 9. In those figures:

- All results are normalized to run 0..100, min..max.
- Each row shows the 25% to 75% quartile range of the normalized scores collected during the simulation.
- The median result is shown as a black dot.
- All the performance scores get *better* when the observed scores get *smaller*.
- The "none" row comes from Monte Carlo simulations of the current ranges, without any changes.

In each figure, the rows are sorted by the number of times a drastic change scores below (*looses* to) other drastic changes. In order to assess number of *losses*, we used the Mann-Whitney test at 95% confidence (this test was chosen due to (a) the random nature of Monte Carlo simulations which results in non-paired tests; and (b) ranked tests make no assumption about the normality of the results). Two rows have the same rank if there is no statistical difference in their distributions.

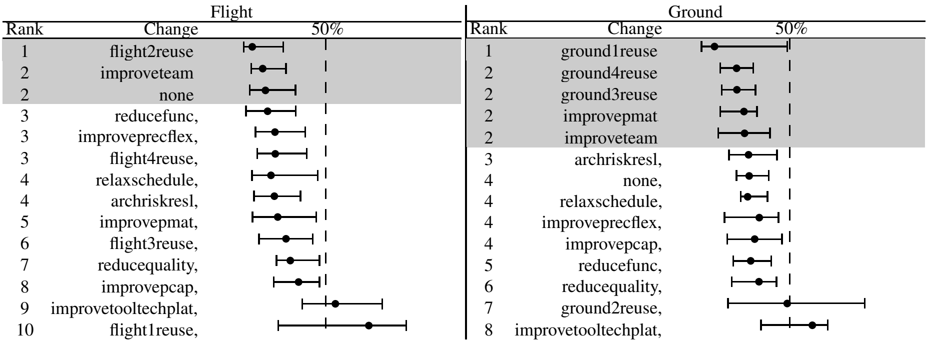
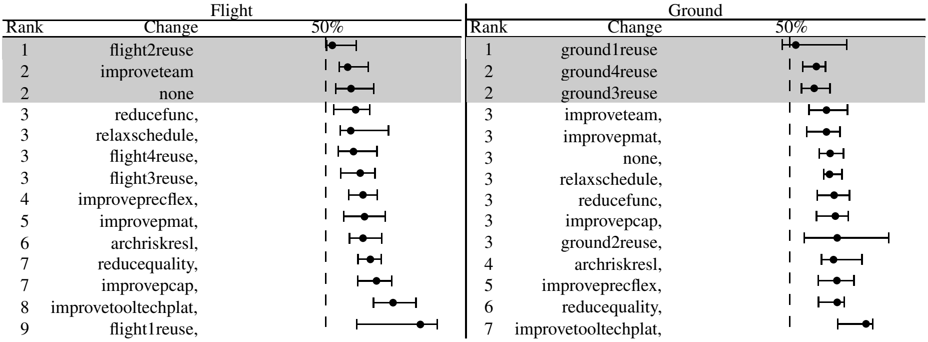
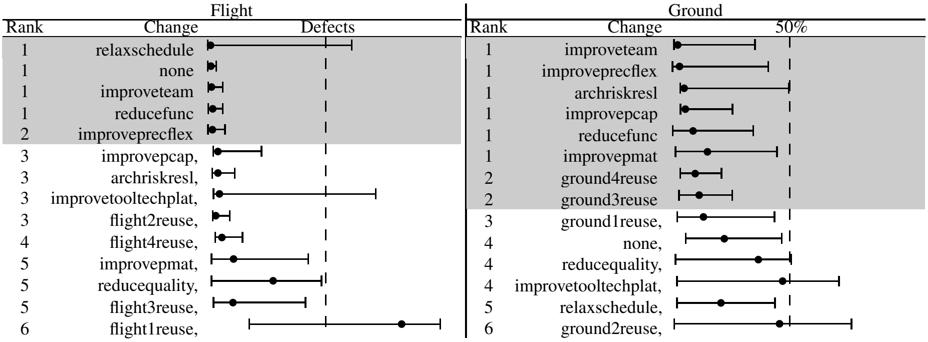


Fig. 7. EFFORT: staff months (normalized 0..100%): top-ranked changes are shaded



Flight		Ground	
Rank	Change	Rank	Change
1	flight2reuse	1	ground1reuse
2	improveteam	2	ground4reuse
2	none	2	ground3reuse
3	reducefunc,	3	improveteam,
3	relaxschedule,	3	improvepmat,
3	flight4reuse,	3	none,
3	flight3reuse,	3	relaxschedule,
4	improveprecflex,	3	reducefunc,
5	improvepmat,	3	improvepcap,
6	archriskresl,	3	ground2reuse,
7	reducequality,	4	archriskresl,
7	improvepcap,	5	improveprecflex,
8	improvetooltechplat,	6	reducequality,
9	flight1reuse,	7	improvetooltechplat,

Fig. 8. MONTHS: calendar (normalized 0..100%): top-ranked changes are shaded.



Flight		Ground	
Rank	Change	Rank	Change
1	relaxschedule	1	improveteam
1	none	1	improveprecflex
1	improveteam	1	archriskresl
1	reducefunc	1	improvepcap
2	improveprecflex	1	reducefunc
3	improvepcap,	1	improvepmat
3	archriskresl,	2	ground4reuse
3	improvetooltechplat,	2	ground3reuse
3	flight2reuse,	3	ground1reuse,
4	flight4reuse,	4	none,
5	improvepmat,	4	reducequality,
5	reducequality,	4	improvetooltechplat,
5	flight3reuse,	5	relaxschedule,
6	flight1reuse,	6	ground2reuse,

Fig. 9. Defect / KLOC (normalized 0..100%): top-ranked changes are shaded

The shaded rows in Figures 7, 8, and 9 mark the top ranked changes. Observe how, with Ground systems, first and second generation reuse always appears in the top rank for effort, months, and defects. That is, for these systems, two generations of reuse is as good, or better, than other proposed changes to a project. That is, for NASA ground software a case can be made in support of software reuse as it is shown to be as good or better than the other strategies.

The results are quite different for Flight systems where all reuse methods are absent from the top ranked changes. In fact, no reuse change appears in the Flight results till rank three or above and all reuse strategies rank below ”none” (meaning that adopting no strategy could yield better results). For NASA flight software, no case can be made for adopting software reuse.

What is surprising is how different the strategies work for similar projects such as the ones presented in this study. The only difference between NASA ground and flight systems lies in size (KLOC), reliability (rely), and complexity(cplx). Ground tends to have lower values for these ranges. Other than that, both case studies have the same ranges and values.

Another finding worth mentioning is how ”improve team” consistently ranks top along side with reuse for ground systems. This might be related to the notion that ”sociology beats technology in terms of successfully completing projects,” [28] or it might be at least comparable.

5 External Validity

Our results make no use of local calibration data. That is, our results are not biased by the historical record of different sites. Hence, in this respect, our results score very highly in terms of external validity.

However, in another respect, the external validity of our results is highly biased by the choice of underlying model (COCOMO, COQUALMO); and the range of changes and projects we have explored:

- *Biases from the model:* While the above results hold over the entire space of calibration possibilities of COCOMO/COQUALMO, then may *not* hold for different models. One reason we are staying with COCOMO/COQUALMO (at least for the time being) is that we have shown that STAR can control these models without requiring local calibration data. We find this to be a compelling reason to prefer these models.
- *Biased from the range of cases explored:* Another threat to the external validity of our models is the range of changes explored in this study. This paper has only ranked reuse against the changes listed in Figure 2. Our changes may do better than and we will explore those in future work.
- *Biases from our selected case studies:* Lastly, we have only explored the projects listed in Figure 6. We are currently working towards a more extensive study where we explore more projects, including projects that do not come from NASA.

6 Conclusion

Our reading of the literature is that much prior assessment of reuse has focused on a very narrow range of of issues. Here, we have tried broadening the debate by assessing reuse with respect to the broader context of minimizing effort and defects and development time.

This paper has explored the case of (a) ranking reuse against different effort, schedule, and defect reduction strategies using (b) models with competing influences that (c) have not been precisely tuned using local data. In this case, we avoided the need for local data to calibrate the models via using the STAR tool. STAR leaves the calibration variables of a model unconstrained, then uses AI search to find project options that most reduces effort, development time, and defects.

STAR was applied here to a study of four incremental reuse strategies and the eight drastic changes. These 13 project changes were applied to two NASA case studies. We found that reuse strategies in general performed as well or better than drastic change strategies on ground software, but did worse than adopting no strategy in the case of flight software systems.

These results suggest that project managers looking for implementing software reuse into their projects may find worthwhile checking the relative merits of reuse against other project improvement options. That is, the relative merits of software reuse should be evaluated in a project-by-project basis. reuse strategies against other project improvement strategies.

In conclusion, in theory, software reuse is an attractive approach to any software project capable of adopting it. However, in practice, reuse might not be the most useful strategy and changing something else (or changing nothing at all) could be more beneficial. AI search over process simulation models is useful for finding the best changes, particularly when there is not sufficient data for precisely tuning a simulation model.

References

1. Trauter, R.: Design-related reuse problems – an experience report. In: Proceedings of the International Conference on Software Reuse (1998)
2. Poulin, J.S.: Measuring Software Reuse. Addison-Wesley, Reading (1997)
3. Lions, J.: Ariane 5 flight 501 failure (July 1996), http://www.ima.umn.edu/~arnold/disasters/ariane5rep.html
4. Leveson, N.G., Turner, C.S.: An investigation of the therac-25 accidents. IEEE Computer 26(7), 18–41 (1993)
5. Raffo, D.: Modeling software processes quantitatively and assessing the impact of potential process changes of process performance, Ph.D. thesis, Manufacturing and Operations Systems (May 1996)
6. Menzies, T., Elwaras, O., Hihn, J., Feathear, M., Boehm, B., Madachy, R.: The business case for automated software engineerng. In: IEEE ASE (2007), http://menzies.us/pdf/07casease-v0.pdf
7. Menzies, T., Elrawas, O., Barry, B., Madachy, R., Hihn, J., Baker, D., Lum, K.: Accurate estiamtes without calibration. In: International Conference on Software Process (2008), http://menzies.us/pdf/08icsp.pdf
8. Menzies, T., Williams, S., El-waras, O., Boehm, B., Hihn, J.: How to avoid drastic software process change (using stochastic statbility). In: ICSE 2009 (2009)
9. Boehm, B., In, H.: Conflict analysis and negotiation aids for cost-quality requirements. Software Quality Professional 1(2), 38–50 (1999)
10. Boehm, B., Horowitz, E., Madachy, R., Reifer, D., Clark, B.K., Steece, B., Brown, A.W., Chulani, S., Abts, C.: Software Cost Estimation with Cocomo II. Prentice-Hall, Englewood Cliffs (2000)
11. Park, R.: The central equations of the price software cost model. In: 4th COCOMO Users (November 1988)
12. Putnam, L., Myers, W.: Measures for Excellence. Yourdon Press Computing Series (1992)
13. Jensen, R.: An improved macrolevel software development resource estimation model. In: 5th ISPA Conference, pp. 88–92 (April 1983)
14. Chulani, S., Boehm, B., Steece, B.: Bayesian analysis of empirical software engineering cost models. IEEE Transaction on Software Engineerining 25(4) (July/August 1999)
15. Raffo, D., Menzies, T.: Evaluating the impact of a new technology using simulation: The case for mining software repositories. In: Proceedings of the 6th International Workshop on Software Process Simulation Modeling (ProSim 2005) (2005)
16. Kelton, D., Sadowski, R., Sadowski, D.: Simulation with Arena, 2nd edn. McGraw-Hill, New York (2002)
17. Abdel-Hamid, T., Madnick, S.: Software Project Dynamics: An Integrated Approach. Prentice-Hall Software Series (1991)
18. Martin, R., Raffo, D.M.: A model of the software development process using both continuous and discrete models. International Journal of Software Process Improvement and Practice (June/July 2000)

19. Mi, P., Scacchi, W.: A knowledge-based environment for modeling and simulation software engineering processes. IEEE Transactions on Knowledge and Data Engineering, 283–294 (September 1990)
20. Cass, A., Lerner, B.S., McCall, E., Osterweil, L., Sutton Jr., S.M., Wise, A.: Little-jil/juliette: A process definition language and interpreter. In: Proceedings of the 22nd International Conference on Software Engineering (ICSE 2000), pp. 754–757 (June 2000)
21. Orrego, A.: Software reuse study report. Technical report, NASA IV&V Facility, Fairmont, WV, USA (April 2005)
22. Orrego, A., Mundy, G.: A study of software reuse in nasa legacy systems. Innovations in Systems and Software Engineering 3(2) (2007)
23. Orrego, A.: Impact of using legacy software on nasa spacecraft. Technical report, NASA IV&V Facility, Fairmont, WV, USA (June 2006)
24. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science 220(4598), 671–680 (1983)
25. Menzies, T., Owen, D., Richardson, J.: The strangest thing about software. IEEE Computer (2007), http://menzies.us/pdf/07strange.pdf
26. Boehm, B.: Software Engineering Economics. Prentice-Hall, Englewood Cliffs (1981)
27. Baker, D.: A hybrid approach to expert and model-based effort estimation. Master's thesis, Lane Department of Computer Science and Electrical Engineering, West Virginia University (2007),
https://eidr.wvu.edu/etd/documentdata.eTD?documentid=5443
28. DeMarco, T., Lister, T.: Peopleware: productive projects and teams. Dorset House Publishing Co., Inc, New York (1987)

Investigating the Gap between Quantitative and Qualitative/Semi-quantitative Software Process Simulation Models: An Explorative Study

He Zhang

Lero - The Irish Software Engineering Research Centre
Department of Computer Science and Information Systems
University of Limerick, Ireland
he.zhang@lero.ie

Abstract. Software Process Simulation Modeling (SPSM) research has increased in the past two decades. However, most of process models for simulation are quantitative, which require detailed understanding and accurate measurement. As the follow-up work to the previous studies in qualitative/semi-quantitative modeling of software process, this paper aims to investigate the equivalence and gap between quantitative and qualitative/semi-quantitative process modeling, to compare the characteristics and performance of these approaches by modeling and simulating a software evolution process, and further to establish the substantial linkage between them. Following the enhanced model conversion scheme developed in this paper, the reference quantitative continuous model and its counterpart models become comparable. The results present their different capabilities and interesting perspectives.

1 Introduction

In the last two decades, Software Process Simulation Modeling (SPSM) has been emerging as an effective tool to help evaluate and manage changes made to software projects and organizations. However, most of software process models for simulation are purely quantitative, and require detailed understanding and accurate measurement of software processes, which relies on reliable and precise history data. Unfortunately, in many cases these data are not readily available, which limits its adoption in practice.

As the counterparts of quantitative modeling, qualitative/semi-quantitative approaches are able to cope with the lack of precise knowledge by modeling at a more abstract level than quantitative modeling. Our previous work explored qualitative/semi-quantitative modeling and simulation of software process at different scales [1,2,3], and justified their values in software process research. In [4], we developed a model conversion scheme for transforming a quantitative continuous process model into an equivalent qualitative model. This paper further enhances this scheme by introducing more quantitative characteristics in support of semi-quantitative model transformation, and reports an explorative study

Q. Wang et al. (Eds.): ICSP 2009, LNCS 5543, pp. 198–211, 2009.

of comparing the characteristics and performance between these approaches by modeling and simulating a software evolution process.

The model conversion scheme is introduced and enhanced in Section 2. Section 3 describes software evolution process and replicates a simplified SD evolution model. In Section 4, the counterpart models of software evolution are developed by applying the conversion scheme. A comparison between these approaches is presented in Section 5, which is followed by conclusions in Section 6.

2 Model Transformation

The selection of quantitative modeling paradigms for conversion and comparison is the first and important step in this study. According to the systematic review [5], system dynamics (SD) and discrete-event simulation (DES) are the two most widely used techniques out of ten simulation paradigms in SPSM, and both quantitative. On the other hand, qualitative/semi-quantitative approaches can be classified as one type of continuous modeling with the incomplete features of discrete transition. For an effective comparison, SD is selected as the quantitative approach due to its wide use in SPSM and continuous characteristics.

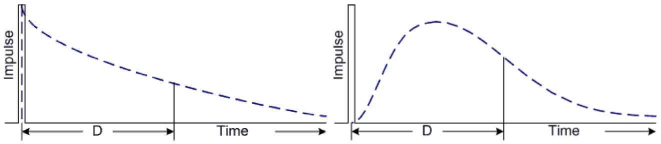
In [4], we discussed the structure equivalence between causal loop diagram (CLD) and abstract structure diagram (ASD), as well as the relationships between the elements of quantitative and qualitative modeling. An initial model conversion scheme was also proposed. Since *time* is treated qualitatively in qualitative modeling and simulation, there is no need to handle a delay in a '*qualitative*' length. Thus the scheme only transforms *level* and *rate* between the approaches in [4]. This section further discusses *delay* in continuous modeling and describes how to effectively transform it for semi-quantitative modeling.

2.1 Delay

Forrester identified two characteristics of a delay [6]. One is the length of time expressing the average delay D, which fully determines the "*steady-state*" effect of the delay. In steady state the flow rate multiplied by the average delay gives the quantity in transit in the delay. The other describes its "*transient response*", which tells how the time shape of the outflow is related to the time shape of the inflow when the inflow rate is changing over time. The delays with the same average delay time (D) can have quite different transient responses to changes in input rate (like plot a and b in Fig. 1).

Exponential delay is the most frequently used delay in SD. Fig. 1 shows two common types of exponential delay used in SPSM: first-order delay (a) and third-order delay (b). Mathematically speaking, an n^{th}-order delay is equivalent to n cascaded single-order delays, with each having a delay time of $\frac{\mathsf{D}}{n}$ [6].

Since *time* is treated qualitatively in qualitative simulation, QSIM algorithm [7] does not explicitly consider delay phenomenon, and neither include built-in *delay* function. However, as semi-quantitative simulation offers the capability



(a) first-order delay (b) third-order delay

Fig. 1. Exponential delay curves

of handling numeric values, the delay function is necessary to maintain the integrity in model transformation. Unfortunately, in terms of the author's knowledge so far, there is no such function built in any available qualitative simulation packages. This section intends to implement the (exponential) '*delay*' in QSIM framework.

First-Order Delay. First-order and third-order exponential delays are two of the most common delays used in the SD models of software process. Fig. 2 is a first-order delay presented in SD diagraming. Given an exogenous inflow rate (IN from other part of system), a first-order delay consists of a simple level (LEV) and a rate of outflow (OUT) that depends on the level and on the delay time (DEL). Table 1 shows the mathematical equations of fist-order delay. The outflow rate (OUT) is equal to the level (LEV) divided by the average delay (DEL).

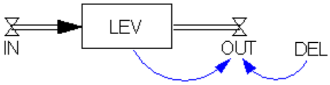
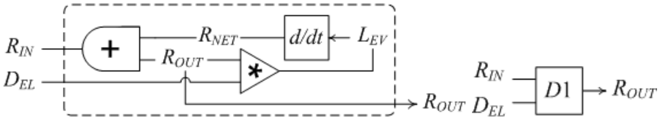


Fig. 2. First-order exponential delay in SD

Table 1. First-order exponential delay

	$\mathsf{OUT}_{[i, i+1]} = \mathsf{DELAY1}\ (\mathsf{IN}_{[i, i+1]}, \mathsf{DEL})$
	$\mathsf{OUT}_{[i, i+1]} = \mathsf{LEV}_{[i]}\ /\ \mathsf{DEL}$ $\mathsf{LEV}_{[i]} = \mathsf{LEV}_{[i-1]} + (\mathsf{DT})(\mathsf{IN}_{[i-1, i]} - \mathsf{OUT}_{[i-1, i]})$
$\mathsf{OUT}_{[i, i+1]}$	the outflow rate between time i and i+1
$\mathsf{LEV}_{[i]}$	the level stored for delay at time i
DEL	the average delay time
DT	the time step between successive evaluations of equation
IN	the inflow rate between time i-1 and i

A first-order delay is composed of four model elements (Fig. 2): one level, two rates, and one auxiliary variable (*delay*). Following the model element conversion described in [4], the structure of first-order delay is converted and represented with ASD notations (Fig. 3-a). A new ASD notation (Fig. 3-b), with two inputs (*inflow* and *delay*) and one output (*outflow*), is created to abstract this structure and avoid the redundant complexity of qualitative model. The detailed qualitative constraints implemented in QDE can be found in [8].



(a) implementation (b) notation

Fig. 3. Implementation of first-order delay in ASD

Third-Order Delay. A third-order delay is the equivalent of three first-order delays cascaded on one after another, so that the output of the first is the input to the second, and the output of the second is the input to the third. Fig. 4 illustrates the structure of a third-order delay in SD format. Table 2 shows the equations for calculating a third-order delay. The outflow rate (OUT) is equal to the level (LEV) divided by the average delay (DEL).

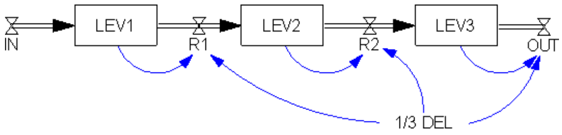
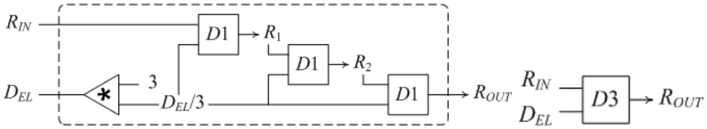


Fig. 4. Third-order exponential delay in SD

Table 2. Third-order exponential delay

$\mathsf{OUT}_{[i,\, i+1]} = \mathsf{DELAY3}\ (\mathsf{IN}_{[i,\, i+1]},\ \mathsf{DEL})$
$\mathsf{R1}_{[i,\, i+1]} = \mathsf{LEV1}_{[i]}\ /\ (\frac{1}{3}\mathsf{DEL})$ $\mathsf{LEV1}_{[i]} = \mathsf{LEV1}_{[i-1]} + (\mathsf{DT})(\mathsf{IN}_{[i-1,\, i]} - \mathsf{R1}_{[i-1,\, i]})$ $\mathsf{R2}_{[i,\, i+1]} = \mathsf{LEV2}_{[i]}\ /\ (\frac{1}{3}\mathsf{DEL})$ $\mathsf{LEV2}_{[i]} = \mathsf{LEV2}_{[i-1]} + (\mathsf{DT})(\mathsf{R1}_{[i-1,\, i]} - \mathsf{R2}_{[i-1,\, i]})$ $\mathsf{OUT}_{[i,\, i+1]} = \mathsf{LEV3}_{[i]}\ /\ (\frac{1}{3}\mathsf{DEL})$ $\mathsf{LEV3}_{[i]} = \mathsf{LEV3}_{[i-1]} + (\mathsf{DT})(\mathsf{R2}_{[i-1,\, i]} - \mathsf{OUT}_{[i-1,\, i]})$
$\mathsf{LEV}_{[i]} = \mathsf{LEV1}_{[i]} + \mathsf{LEV2}_{[i]} + \mathsf{LEV3}_{[i]}$

By applying the new ASD notation created for first-order delay (Fig. 3-b), the structure of a third-order delay can be represented in Fig. 5-a. Again, another new ASD notation (Fig. 5-b), with two inputs and one output, is created to abstract this more complicated structure. Here, the implemented third-order delay also demonstrates how to construct an n^{th}-order delay using the basic first-order delay in QDE (based on QSIM algorithm framework).



(a) implementation (b) notation

Fig. 5. Implementation of third-order delay in ASD

3 Reference Quantitative Model

This section introduces the selection and background of the reference software process modeled and replicates this evolution model for further comparison.

3.1 Reference Model Selection

In 2007, we undertook a systematic literature review of SPSM research in the past decade (1998-2007) [5,9]. This review found *phase*, *project*, and *product evolution* as the most modeled software process scales; as well as identified *generic development*, *software evolution*, *software process improvement*, and *incremental development* as the top four in the most interesting topics in SPSM research. As we previously investigated qualitative modeling of *generic development* [1,2] (software staffing process) and *incremental development* [3] at *project* and *phase* scales, this paper compares these modeling approaches with focus on *software evolution*. Moreover, the successful modeling and simulating software processes at all these scales and topics derived from the systematic review provide evidence for justifying their values in software process research.

The reference model also needs to contain most common elements and relations of the chosen evolution process. Nevertheless, the model's structure should be clear and simple enough to ensure the emphasis of this research on model transformation and comparison, rather than construction of a complicated model.

According to the above criteria of the reference quantitative model, an SD model of software evolution process is selected for this study. There are several candidate models published in the last decade, some of them appear in [10,11,12,13,14]. Among them, Wernick and Hall's model [14] consists of a single module, which is simpler than others'. Moreover, their model is the most recent SD model of evolution process found in the primary studies of the systematic review.

3.2 Software Evolution Process

Software evolution process is one of the most investigated software processes in SPSM [5]. The insights obtained from the previous studies indicated that software evolution could be systematically studied and exploited using SPSM approaches. They also suggested that to some extent software evolution is a disciplined phenomenon as illustrated, for example, by the regularity of functional growth patterns [15]. Models of such patterns permitted the forecasting of future overall system growth and growth rates. Moreover, the observed patterns of behavior appearing yielded common phenomenological interpretations.

Basically, four important feedback structures, identified by the previous related work, are used in model construction of software evolution processes:

Inertia-like (anti-regressive) effect due to system growth. The first hypothesis is that increasing the size of a software system and changes in unanticipated directions will over time reduce the enhancement and modification of that system [13]. These changes may result in a decay in software architecture. Meanwhile, new changes also have to be fitted into an existing system structure, and as the software grows, there are more existing components into which each new change needs to be fitted. Thereby, software developers are occupied on tasks specifically intended to maintain the system structure, and to compensate for the software aging effects, which are referred to as '*anti-regressive*' activities [12].

Effects of decreasing knowledge coverage. The increasing complexity of a software system also reduces the developer's ability to change the system because of a decrease in coverage of developer's knowledge of the system components, their composition and interactions [13]. As the software grows, the amount of knowledge needed to support future changes grows as well, but at a faster rate, as the implementation of each new component needs to be seen in the context of *all* of the existing system [16,17]. If the developer's knowledge does not grow at this rate, it may fall behind the knowledge needed to support further changes.

Generation (progressive effect) of new requirements. The release of upgraded software with new functionalities enables users to exploit opportunities for novel or extended system use, which in turn result in demand for further functionalities [11]. This positive feedback is recognized as '*progressive*' type of work, which represents the evolution activities that enhance software functionality by modification of or addition to the code and/or the documentation [12].

Correction of fault implementation. After the release and adoption of software, a small proportion of units (requirements) is gradually found not to be implemented as originally or correctly specified. They are eliminated from the specification, but may be replaced in the requirements by new or changed equivalents [10]. The rate of completion of successful implementation can be reflected by a *success* or *failure* percentage.

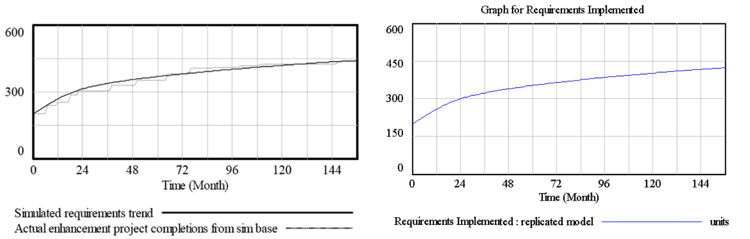
Some or all of these hypothetical drivers of specification change is reflected in (positive or negative) feedback loops in SD modeling, again calculated as portions of the successful implementation flow.

3.3 A Simplified Quantitative Model

Model Description and Calibration. The reference quantitative model (shown in Fig. 7) was developed using Vensim simulation environment (Ventana Systems, Inc.). Though it is a simplified model, it incorporates and reflects three of the typical feedback loops described above that are indicated by the loop numbers. They are feedback structures representing the inertia effect (anti-regressive activities, Loop 1), the generation of new requirements (progressive activities, Loop 2), and the correction of faults in previous implementation (Loop 3).

The '*size*' of software system has been abstracted into a number of arbitrary-sized '*units*' of requirements, since it is a more informative reflection of software evolution, which is more likely driven by changes in functionality than by low-level thinking with '*code*' [16]. Plus, it avoids issues related to specific metric.

The delay used in this model is a *third-order* delay, which fits the technical software process [10]. It represents the time delays caused by some entity passing through the phases of a process made up of a sequence of sub-processes, each of which depends for its input on the output of the previous one.



(a) system size by the reference model (b) system size by the replicated model

Fig. 6. Simulation of implemented requirements over time

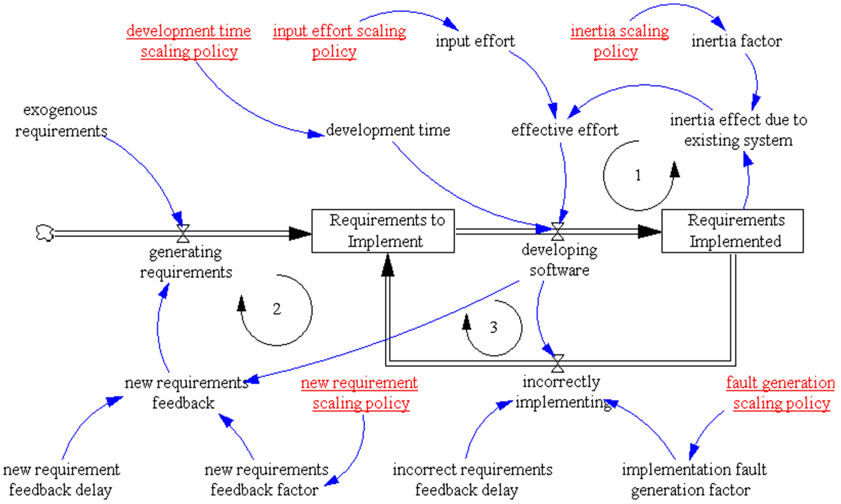
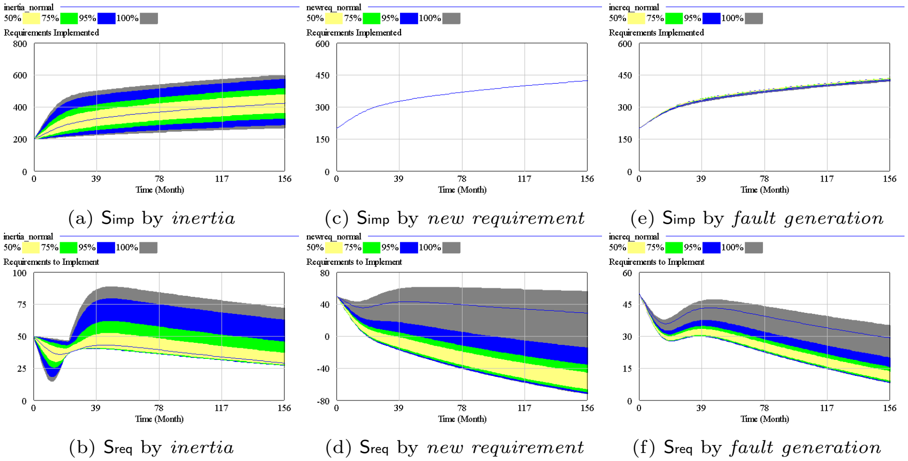

development time scaling policy
input effort scaling policy
input effort
inertia scaling policy
inertia factor
exogenous requirements
development time
effective effort
inertia effect due to existing system
1
Requirements to Implement
Requirements Implemented
generating requirements
developing software
2
3
new requirements feedback
new requirement scaling policy
incorrectly implementing
fault generation scaling policy
new requirement feedback delay
new requirements feedback factor
incorrect requirements feedback delay
implementation fault generation factor

Fig. 7. Reference Quantitative model of software evolution

The calibration inputs to the reference model are based on actual data for the evolution of the ICL VME mainframe system described in [11]. To guarantee the replication of the reference model, most of these variable inputs are kept here.

Note that the replicated reference model in this section is based on the SD flow graphs, inputs, outputs, and relation equations published in [11,12,13,14] (no full version models published). As result of this divergence, the output of the replicated model is slightly different from the originals. Fig. 6 shows the overall evolution trends are similar to each other, and the only difference on simulated system sizes, which can be regarded as the scaling effect of *inertia factor*.

Sensitivity to Policy Change. The reference model has been subjected to a further sensitivity analysis to investigate the effects of changes in policy inputs. Wernick and Hall introduced five policy factors to the reference model, which are underlined in Fig. 7. Each of the policy input varies from its default value of 1, using a normal distribution with a standard deviation of 0.25. For



(a) $\mathsf{S}_{\mathsf{imp}}$ by *inertia*

(c) $\mathsf{S}_{\mathsf{imp}}$ by *new requirement*

(e) $\mathsf{S}_{\mathsf{imp}}$ by *fault generation*

(b) $\mathsf{S}_{\mathsf{req}}$ by *inertia*

(d) $\mathsf{S}_{\mathsf{req}}$ by *new requirement*

(f) $\mathsf{S}_{\mathsf{req}}$ by *fault generation*

Fig. 8. Sensitivity of policy change for reference model

instance, the values greater than 1 of '*inertia scaling policy*' indicates the higher maintainability and evolvability of the system.

A similar sensitivity analysis design was applied to the replicated model. To simplify the discussion, three of their policy factors are selected to investigate the policy sensitivity of three feedback loops respectively. Fig. 8 shows the distributions of simulated system size growth and volume of requirements over time for 1000 runs for each parameter varied separately. The solid line in each plot indicates the mean result, and the regions either side of it contain 50%, 75%, 95%, and 100% of the simulated results respectively.

4 Corresponding Qualitative/Semi-quantitative Model

4.1 Qualitative Model

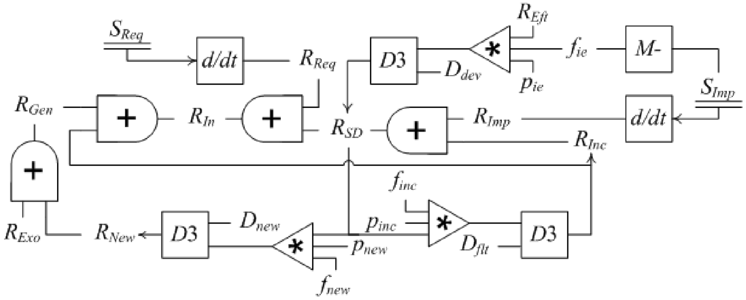
In qualitative modeling, the qualitative assumptions need to be abstracted from the real world system [18]. Here, the reference SD model should be converted into the corresponding qualitative model based on the discussion in [4]. The follows are the inherent qualitative assumptions extracted. Note that the qualitative model does not explicitly include the three delay relations in the reference model. More description of the qualitative model can be found in [8].

1. *Requirements to implement* ($\mathsf{S}_{\mathsf{Req}}$) come from *exogenous requirements*, *new requirements feedback* and *incorrect requirements feedback*;
2. $\mathsf{S}_{\mathsf{Req}}$ is transferred to *requirements implemented* ($\mathsf{S}_{\mathsf{Imp}}$) at *developing rate* ($\mathsf{R}_{\mathsf{SD}}$);
3. The *incorrectly implemented requirements*, as a small portion of $\mathsf{S}_{\mathsf{Imp}}$ is returned to $\mathsf{S}_{\mathsf{Req}}$ for rework;
4. Increasing existing system size ($\mathsf{S}_{\mathsf{Imp}}$) incurs more effort needed for '*anti-regressive activities*', and decreases R_{SD};

5. The *input effort* (R_{Eft}) has linear relationship with R_{SD};
6. The *new requirement feedback* (R_{New}) has linear relationship with R_{SD};
7. The *incorrectly implementation* (R_{Inc}) has linear relationship with R_{SD};
8. The development team size does not change (neither recruitment nor turnover) during the evolution process;
9. There is no *exogenous requirements* ($R_{Exo} = 0$) during the evolution process.

4.2 Semi-quantitative Model

Fig. 9 shows the semi-quantitative model based upon th graph of qualitative model and more constraints (*delays*) introduced. To be noticed, the newly introduced notations representing the exponential delays (Section 2.1) are included in semi-quantitative constraints.



S_{Req}: Requirements to implement
R_{Req}: Requirement generation rate
R_{New}: New requirement feedback rate
R_{SD}: Software development rate
R_{In}: Requirement input rate
R_{Eft}: Effective effort rate
f_{new}: new requirement feedback factor
D_{dev}: Development delay time
D_{inc}: Incorrect requirement feedback delay
p_{new}: new requirement feedback policy factor

S_{Imp}: Requirements implemented
R_{Imp}: Requirement implementation rate
R_{Inc}: Incorrect implementation rate
R_{Gen}: Requirement generation rate
R_{Exo}: Exogenous requirement rate
f_{ie}: inertia factor
f_{inc}: fault generation factor
D_{new}: New requirement feedback delay
p_{ie}: inertia scaling policy factor
p_{inc}: implementation fault feedback policy factor

Fig. 9. Corresponding semi-quantitative model of software evolution

One monotonic function (`M-`, between f_{ie} and S_{Imp}) is included in the qualitative model. This nonlinear relation needs to be quantified at this stage. Nevertheless, the inertia factor was quantified as a multiple of the inverse square [13] or inverse cube [14] of existing system size respectively. Therefore, an envelop function (Equation 1) is constructed for this case.

$$f_{ie} = \left[\left(\frac{\lambda}{S^3_{imp}} \right), \left(\frac{\lambda}{S^2_{imp}} \right) \right] \tag{1}$$

where λ is a suitable constant, and determined from historic data.

5 Simulation and Comparison

The qualitative simulation generates a diversity of behaviors of the evolution process, most of which are the combinations of varying patterns of important variables, including *requirements implemented*, *requirements implemented*, *requirement implementation rate*, and *requirement generation rate*. The oscillation phenomenon is observed and consistent with the qualitative behaviors reported by Ramil and Smith in [19], which constructed qualitative simulation model based on analytic models, instead of continuous casual model. More detailed results from qualitative simulation can be found in [4].

5.1 Single-Point Value Simulation

The comparison between quantitative simulation (SD) and semi-quantitative simulation (SQSIM) can be conducted from two aspects: simulation with single-point value and value range. Traditionally, purely quantitative simulation always assign a single-point (numeric) value for each input variable during each run of simulation. In contrast, SQSIM treats single-point values as value ranges without interval, i.e. the upper and lower bounds are set as the same. Hence, the inputs to semi-quantitative model are set with the same values as in the reference model (Section 3) for comparison.

On the other hand, the envelop function should be also replaced with the *exact* function to eliminate the uncertainty. Here the *exact* function (Equation 2) is used to replace the *envelop* function (Equation 1) for comparison.

$$f_{ie} = \left[\left(\frac{\lambda}{S_{imp}^{3}} \right), \left(\frac{\lambda}{S_{imp}^{3}} \right) \right] \tag{2}$$

Different from purely quantitative simulation, the SQSIM generates nine behaviors, even with single-point settings. Although the simulation is preset to terminate at the 156^{th} month [14], Q2 algorithm also includes some behaviors that may terminate in a range between [29.6, 156] months. It is because SQSIM is inherently based on value range, rather than single-point value. Any behaviors covering this condition are generated by algorithm. For comparison, this study is only interested in the behaviors exactly terminated at the preset time point. By removing the '*invalid*' behaviors, there are two behaviors consistent to the reference model. Fig. 10 depicts the varying trends of some important variables.

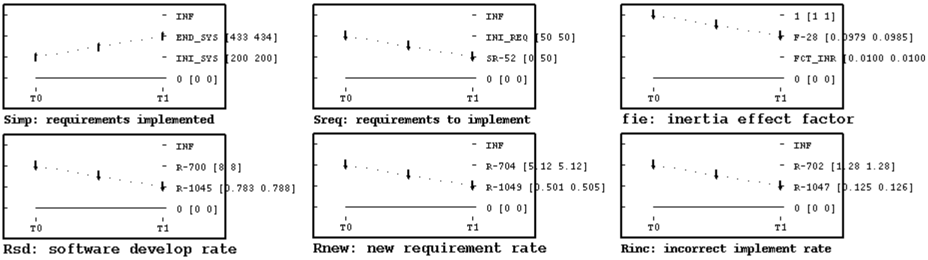


Fig. 10. Behavior of semi-quantitative simulation with single-point values

The only difference between these two behaviors is that $\mathsf{S_{Req}}$ may finish at 0 or in the rang [0, 50] units when the simulation terminates. The simulation in Section 3 predicts that the system size may grows up to 433.75 units at the 156^{th} month. Both the valid behaviors from SQSIM produce the close value range, [433, 434] units, for $\mathsf{S_{Imp}}$. Moreover, it is interesting that SQSIM presents other variables (e.g. f_{ie}, $\mathsf{R_{SD}}$, $\mathsf{R_{New}}$ and $\mathsf{R_{Inc}}$) in its inherent format (value range) as well. It is possibly caused by the slim deviation between the functions and their inverse versions, which are required for describing envelop functions.

Compared to the simulation result of system dynamics (presented in Fig. 6), the graphic result of semi-quantitative simulation only depicts the monotonic trend of $\mathsf{S_{Imp}}$, but lacks detailed shape, which depends on the number of *landmark* created in the course of simulation. It can be enriched by Q3 algorithm.

5.2 Value-Range Simulation

In Section 3, several policy factors are introduced for sensitivity analysis (Fig. 7). This subsection emphasizes on the value range comparison, between the Monte Carlo simulations of these inputs in terms of probability distributions and semi-quantitative simulations with the corresponding value ranges. Fig. 9 shows the semi-quantitative model with the policy factors (p_{ie}, p_{new}, and p_{inc}). Note that their probability distributions are not necessary in this form of simulation.

The results of SQSIM are summarized in Table 3. Although the value ranges generated by normal QSIM+Q2 algorithm (the second row) are consistent with SD results (covering the intervals on the first row), it is easy to observe that they remain very coarse. It is mainly because the evolution behaviors ($\mathsf{S_{Imp}}$) are monotonic and smooth, which, unlike the models reported in our previous studies, include no transition points and few *critical* time points. As a result, the *landmarks* inserted into qualitative intervals are not sufficient to generate finer ranges to reduce the uncertainty.

Table 3. Value range comparison

	$\mathsf{S_{Imp}}$ by f_{ie}	$\mathsf{S_{Imp}}$ by $\mathsf{R_{New}}$	$\mathsf{S_{Imp}}$ by $\mathsf{R_{Inc}}$
SD	[270, 601]	[434, 434]	[424, 443]
SQSIM (Q2)	[220, 627]	[423, 525]	[410, 527]
SQSIM (Q3)	[263, 610]	[430, 455]	[420, 471]

To improve the performance of SQSIM, Q3 algorithm, which extends Q2 with '*step-size refinement*' [18], is further employed to obtain finer-grained value ranges. The simulation results (with step-size of 10) are listed on the bottom row in Table 3. It demonstrates that the accuracy of SQSIM can be improved significantly by adaptively introducing additional *landmarks*. The remaining difference between these approaches is probably caused by *1*) the unique reasoning mechanism of SQSIM that is based on the behavior chattering instead of single-point calculation; *2*) the sampling and assumed probability distribution (in quantitative simulation) cause some missing points; *3*) step-size of Q3 has yet been completely optimized. Along with Q3 and other advanced refinement techniques (e.g. dynamic envelopes), SQSIM can smoothly span the gap from qualitative simulation on one hand to purely quantitative simulation on the other [18].

Overall, both qualitative/semi-quantitative simulation and purely quantitative simulation (SD) compared here have their strength and weakness. In modeling, compared with CLD, the qualitative approach starts at explicitly stated qualitative assumptions, and then converts them into more specific and clearer constraints. Thus, it provides a more rigorous approach. Moreover, a CLD model does not offer simulation capability, but an ASD model does. Both of them can be quantified to become their quantitative/semi-quantitative counterpart.

In simulation, both of SD and SQSIM can produce *similar* results in single-point simulation. SD can present the variable's varying trend with more details during the course. Whereas, the SQSIM approach reflects trends more qualitatively through QSIM, which shares the same plotting mechanism with the qualitative simulation engine. When dealing with uncertainty, the value range and its associated probability distribution is required for any stochastic (quantitative) simulation. In contrast, SQSIM handles uncertainty with value range and envelop function, which is able to omit their probability. In some cases, results of SQSIM are coarse and need to be further refined and optimized.

6 Conclusions

Our previous studies investigated the use of qualitative/semi-quantitative modeling in software engineering research, and constructed the process models at a variety of process scales. In [4], we initially developed a model conversion scheme between qualitative and quantitative process models based on a structure and element level mapping. As the follow-up work, this paper first enhances this scheme by introducing and handling *delay* in qualitative/semi-quantitative model, and then developed the corresponding semi-quantitative model of the reference quantitative (SD) model using the scheme. The characteristics and performance of these approaches are further compared. The major contributions of this paper are highlighted as the follows:

1. A model conversion scheme between quantitative (SD) and qualitative/semi-quantitative modeling is developed and enhanced by implementing the structure and element level mapping. From a given qualitative model and quantification, it is possible to transform and construct its equivalent SD model as well.
2. The n^{th}-order delay is introduced into semi-quantitative modeling and simulation, and implemented in QSIM algorithm framework.
3. The software evolution processes are revisited by using qualitative/semi-quantitative modeling and simulation.
4. The modeling characteristics are compared between system dynamics and qualitative/semi-quantitative modeling; and further the comparison of simulation capability and results between SD and SQSIM are presented.

The quantitative and qualitative/semi-quantitative modeling approaches compared in this paper offer a number of different capabilities and interesting perspectives in software process research. This research establishes a preliminary foundation for bridging these approaches, and provides the modelers an easily transformable alternative. However, as an explorative study, this paper cannot investigate all aspects of them at current stage. The future work in this direction can be continued to achieve a more holistic understanding.

References

1. Zhang, H., Huo, M., Kitchenham, B., Jeffery, R.: Qualitative simulation model for software engineering process. In: 17th Australian Software Engineering Conference (ASWEC 2006), Sydney, Australia, pp. 391–400. IEEE Computer Society Press, Los Alamitos (2006)
2. Zhang, H., Kitchenham, B.: Semi-quantitative simulation modeling of software engineering process. In: Wang, Q., Pfahl, D., Raffo, D.M., Wernick, P. (eds.) SPW 2006 and ProSim 2006. LNCS, vol. 3966, pp. 242–253. Springer, Heidelberg (2006)
3. Zhang, H., Keung, J., Kitchenham, B., Jeffery, R.: Semi-quantitative modeling for managing software development processes. In: 19th Australian Software Engineering Conference (ASWEC 2008), Perth, Australia, pp. 66–75. IEEE Computer Society Press, Los Alamitos (2008)
4. Zhang, H., Kitchenham, B., Jeffery, R.: Qualitative vs. quantitative software process simulation modeling: Conversion and comparison. In: 20th Australian Software Engineering Conference (ASWE 2009), Gold Coast, Australia. IEEE Computer Society Press, Los Alamitos (2009)
5. Zhang, H., Kitchenham, B., Pfahl, D.: Reflections on 10 years of software process simulation modeling: A systematic review. In: Wang, Q., Pfahl, D., Raffo, D.M. (eds.) ICSP 2008. LNCS, vol. 5007, pp. 345–356. Springer, Heidelberg (2008)
6. Forrester, J.W.: Industrial Dynamics. System Dynamics Series. Pegasus Communications (1969)
7. UT Qualitative Reasoning Group: Qsim version 4.0-alpha-4, University of Texas, http://www.cs.utexas.edu/users/qr/QR-software.html
8. Zhang, H.: Qualitative and Semi-quantitative Modelling and Simulation of Software Engineering Processes. PhD thesis, University of New South Wales, Australia (2008)
9. Zhang, H., Kitchenham, B., Pfahl, D.: Software process simulation modeling: Facts, trends, and directions. In: 15th Asia-Pacific Software Engineering Conference (APSEC 2008), Beijing, China, pp. 59–66. IEEE Computer Society, Los Alamitos (2008)
10. Wernick, P., Lehman, M.: Software process white box modelling for feast/1. Journal of Systems and Software 46(2-3) (1999)
11. Chatters, B.W., Lehman, M., Ramil, J.F., Wernick, P.: Modelling a software evolution process: A long-term case study. Software Process: Improvement and Practice 5(2-3) (2000)
12. Kahen, G., Lehman, M., Ramil, J.F., Wernick, P.: System dynamics modeling of software evolution processes for policy investigation: Approach and example. Journal of Systems and Software 59(3) (2001)

13. Wernick, P., Hall, T.: Simulating global software evolution processes by combining simple models: An initial study. Software Process: Improvement and Practice 7(3-4) (2002)
14. Wernick, P., Hall, T.: A policy investigation model for long-term software evolution processes. In: 5th International Workshop on Software Process Simulation Modeling (ProSim 2004), Edinburgh, Scotland, pp. 206–214 (2004)
15. Lehman, M., Ramil, J.F.: Software evolution - background, theory, practice. Information Processing Letters 88 (2003)
16. Turski, W.M.: The reference model for smooth growth of software systems. IEEE Transactions on Software Engineering 22(8) (1996)
17. Turski, W.M.: The reference model for smooth growth of software systems revisitied. IEEE Transactions on Software Engineering 28(8) (2002)
18. Kuipers, B.: Qualitative Reasoning: Modeling and Simulation with Incomplete Knowledge. MIT Press, Cambridge (1994)
19. Ramil, J.F., Smith, N.: Qualitative simulation of models of software evolution. Software Process: Improvement and Practice 7(3-4) (2002)

Bridge the Gap between Software Test Process and Business Value: A Case Study

Qi Li[1], Mingshu Li[2], Ye Yang[2], Qing Wang[2], Thomas Tan[1], Barry Boehm[1],
and Chenyong Hu[2]

[1] University of Southern California
{qli1,thomast,boehm}@usc.edu
[2] Institute of Software, Chinese Academy of Sciences
{mingshu,ye,wq,huchenyong}@itechs.iscas.ac.cn

Abstract. For a software project to succeed, acceptable quality must be achieved within an acceptable cost, providing business value to the customers, and keeping delivery time short. Software testing is a strenuous and expensive process and is often not organized to maximize business value. In this article, we propose a practical value based software testing method which aligns the internal test process with the value objectives coming from the customers and the market. Our case study in a real-life business project shows that this method helps manage testing process effectively and efficiently.

Keywords: value, business importance, risk, cost, testing, AHP.

1 Introduction

Cost, schedule and quality are highly correlated factors in trustworthy software development. They basically form three sides of an under-resourced triangle [1]. Companies are often faced with time and resource constraints that limit their ability to effectively complete testing efforts. This situation can often be improved by regarding the testing activity as an investment during the software life cycle [2]. Maximizing the value contribution of software testing can help us maximize the return on investment during the software testing phase. Managing software testing based on value considerations promises to deal with increasing testing costs and required effort [3].

Testing is one of the most labor-intensive activities in software development life cycle and consumes between 30% and 50% of total development costs according to many studies [3, 4]. Traditional testing methodologies such as: path, branch, instruction, mutation, scenario, or requirement testing usually treat all aspects of software as equally important [5], however, in practice 80% of the value often comes from 20% of the software [2, 6, 7], like a Pareto distribution. As Rudolf Ramler mentioned in [3], testing's indirect contribution to product value leads to a value-neutral perception of testing. The common separation of concerns between development and testing exacerbates the problem. Testing is often reduced to a purely technical issue leaving the close relationship between testing and business decisions unlinked and the potential value contribution of testing unexploited. He also points out that the key challenge in value-based testing is to integrate two dimensions: aligning the internal test process with the value objectives coming from the customers and the market [3].

Q. Wang et al. (Eds.): ICSP 2009, LNCS 5543, pp. 212–223, 2009.

The objective of this research is to apply value-based test management at the system level of trustworthy software development. We propose a value-based software testing method to better align investments with project objectives and business value. This method can provide decision support for test managers to deal with resource allocation, tradeoff and risk analysis, and time to market initiatives and software quality improvement and investment analysis. We also conducted a case study in a real-life project.

2 Related Work

Our work is primarily relevant to the research areas of Value Based Software Engineering, and Software Testing Methodology. The initial theory of Value Based Software Engineering unveils the fact that successful quality should be achieved at a level that makes all key stakeholders winners[8]. [8]introduces the “4+1” theory of VBSE. The center of the theory is the success-critical stakeholder (SCS) win-win Theory W [9, 10], which addresses what values are important and how success is assured for a given software engineering organization. The four supporting theories that it draws upon are utility theory, decision theory, dependency theory, and control theory, respectively dealing with how important are the values, how do stakeholders’ values determine decisions, how do dependencies affect value realization, and how to adapt to change and control value realization. The basic idea of value-based testing is a branch of value-based Verification & Validation (V&V) which is to treat each V&V activity (analysis, review, test) as a candidate investment in improving the software development process and ensuring that a software solution satisfies its value objectives [11].

Risk-based testing [12, 13], value-based testing [3] and value-based quality analysis [14, 15] are process strategies that are the most related with our research objective. Stale Amland introduces a risk-based testing approach [12] in which resources should be focused on those areas representing the highest risk exposure. However, this method doesn’t consider the testing cost which is also an essential factor during testing process. Besides we enhance its business importance calculation by Karl Wiegers’s method [16], incorporate risk analysis and improve it by introducing AHP (The Analytic Hierarchy Process) Method [17] to determine risk factors weights and reduce expert estimation bias. Rudolf Ramler outlines a framework for value-based test management [3], however, his outline doesn’t provide the implementation in details. Based on the COCOMO II cost-estimation model [18] and the COQUALMO quality estimation model [19], Liguo Huang proposes a quantitative risk analysis [20] which helps determine when to stop testing software and release the product. However, her method is more macroscopic and doesn’t consider any controlling and monitoring metrics of the testing process.

For adoption by industrial test teams, a prioritization technique with the following attributes would be beneficial [21]: 1) economical, not adding significant overhead or burden to the testing team; 2) improving customer-perceived software quality by decreasing field defects; 3) increasing business value for the customers. Our method is simple to follow and able to align software quality improvement with business value achievements from customers.

3 Method Overview

Fig 1 illustrates the whole process of this value-based software testing method. This method helps test manager enact the testing plan and adjust it during testing execution. This method has three main steps:

1). In Identifying SCSs and their win conditions, section 4 of this paper will consider three main factors for priority: 1). Key customers calculate the relative business importance of the features using a method first proposed by Karl Wiegers [16] (section 4.1.1). 2). Developers, the project manager and the test manager calculate the quality risk probability of each feature.(section 4.1.2). 3). The test manager estimates the testing cost of each feature (section 4.1.3).

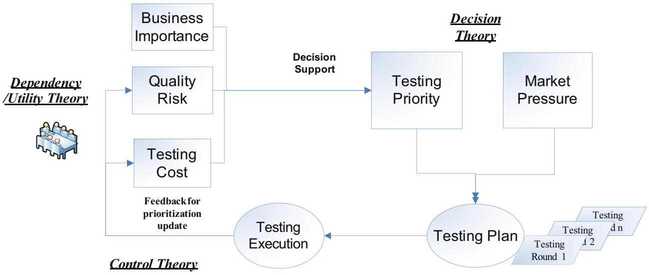


Fig. 1. Method Overview

This step brings the stakeholders together to consolidate their value models and to negotiate testing objectives. They are according to the Dependency and Utility Theory in VBSE which help to identify all of the SCSs and understand how the SCSs want to win.

2). We put value, risk and cost together and calculate a value priority number for each feature (section 4.2). This is like a multi-attribute decision and negotiation process which follows the Decision Theory in VBSE. Features' value priority helps us enact the testing plan, and resources should be focused on those areas representing the most important business value, the lowest testing cost and highest quality risk.

3). During the testing process, we adjust each feature's value priority according to the feedback of quality risk indicators, and updated testing cost estimation (section 4.3). This step assists to control progress toward SCS win-win realization which is according to the Control Theory of VBSE.

Finally, the case study result analysis in section 5 shows that this method is especially effective when the market pressure is very high.

4 Method with a Case Study

Our case study is based on a real-life business project in a software organization. This organization is a research and development organization in China which is appraised and rated at CMMI maturity level 4 in 2005. There is a research group in it focusing on the software process improvement and quantitative process management, which has established some quantitative management methods [22, 23]. The group also developed a toolkit called SoftPM which is used to manage software project and has been deployed to many software organizations in China. Our case study is based on one of its web-based system development projects with a whole size of 553 KLOC. It employs iterative development method and CMMI process management. The recent iterative development covers 9 new features with an overall size of 32.6 KLOC Java codes. The features are mostly independent amendments or patches of some existing modules. We use F1,F2…F9 to denote these 9 features for short. We apply this method in its system testing process, and we manage and control our process of testing with the help of SoftPM.

4.1 Identify SCSs and Win Conditions

To provide a more practical set of guidelines for prioritizing, the first thing is to identify SCSs and their win conditions. Direct stakeholders of testing are developers and project managers, who directly interact with the testing team. However, in the spirit of value-based software engineering important parties for testing are customers. Customers are the source of value objectives, which set the context and scope of testing. Marketing and product managers assist in testing for planning releases, pricing, promotion, and distribution[3].

We will look at three factors that must be considered when prioritizing the testing order of new features, and they represent SCSs's win conditions:

1).The business importance of having the features. It gives information as to what extent mutually agreed requirements are satisfied and to what extent the software meets their value propositions.

2).The quality risk of each feature. For project managers it supports risk management and progress estimation. The focus is on identifying and eliminating risks that are potential value breakers and inhibit value achievements.

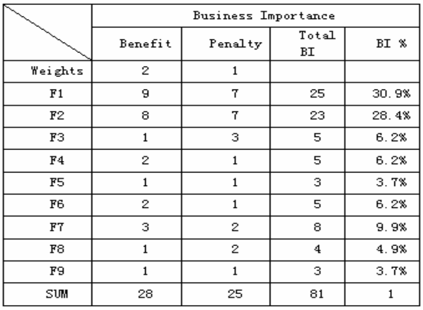
3).The cost estimation for testing each feature. Testing managers are interested in the identification of problems particularly the problem trends that helps to estimate and control testing process.

Because most projects are undertaken either to save or to make money, the first and third factors often dominate prioritization discussions. However, proper consideration of the influence of risk on the project is critical if we are to prioritize optimally. We will discuss them respectively in the subsequent three sections. They are according to the Dependency and Utility Theory in VBSE which help to identify all of the SCSs and understand how the SCSs want to win.

4.1.1 Business Importance

To determine business importance of each feature, we apply Karl Wiegers' approach [16] to our case study. This approach considers both the positive benefit of the presence of a feature and the negative impact of its absence and this approach relies on expert judgment by representative customers. Each feature is assessed in terms of the benefits it will bring if implemented, as well as the penalty that will be incurred if it is not implemented. The estimates of benefits and penalties are relative. A scale of 1 to 9 is used. The importance of including relative penalty should not be neglected. For example, failing to comply with a government regulation could incur a high penalty even if the customer benefit is low, as would omitting a feature that any reasonable customer would expect, whether or not they explicitly requested it. For each feature, the relative benefit and penalty are summed up and entered in the Total BI (Business Importance) column in Table 1. Fi's Total BI can be calculated by the following formula.

Table 1. Relative Business Importance Calculation



	Business Importance			
	Benefit	Penalty	Total BI	BI %
Weights	2	1		
F1	9	7	25	30.9%
F2	8	7	23	28.4%
F3	1	3	5	6.2%
F4	2	1	5	6.2%
F5	1	1	3	3.7%
F6	2	1	5	6.2%
F7	3	2	8	9.9%
F8	1	2	4	4.9%
F9	1	1	3	3.7%
SUM	28	25	81	1

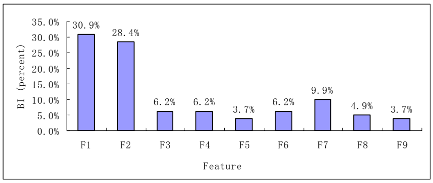


Fig. 2. Business Importance Distribution

$$\text{Total BI}_{\text{i}} = W_{Benefit} * Benefit_{i} + W_{Penalty} * Penalty_{i}$$

The sum of the Total BI column represents the total BI of delivering all features. To calculate the relative contribution of each feature, divide its total BI by the sum of the Total BI column.

In our case study, we asked our key customers to determine each feature's BI according to the above approach. They emphasize business benefit rather than penalty, and came to the agreement that weights the relative benefit as twice as important as the relative penalty. They gave the explanation that the 9 new features are amendments and updated functions for former version which has implemented the main functions ordered by customers and implementing these 9 features will bring more benefit to customers, while not implementing them will not greatly influence regular use. So in Table 1 they multiply the Relative Benefit by 2 before adding it to Relative Penalty to get the Total BI. Fig 2 shows the BI distribution of the 9 features. As we can see, there is an approximate Pareto distribution in which F1 and F2 contribute 22.2% of the features and 59.2% of the total BI.

4.1.2 Quality Risk

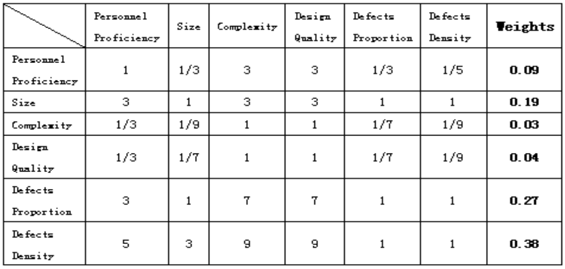
The risk analysis was performed prior to system test start, but was continuously updated during test execution. It aims to calculate the risk probability for each feature. There are three steps in quality risk analysis: the first is listing all risk factors based on history projects and experiences. Then use AHP (The Analytic Hierarchy Process) Method [17]to determine the weight for each risk factor and AHP consistency testing. The third is calculating the risk probability for each feature.

Step 1: Set up the n risks in the rows and columns of an $n*n$ matrix. In our case study, according to Company A's historical projects of the same type, we have six main quality risks: Personnel Proficiency, Size, Complexity, Design Quality, Defects Proportion, and Defects Density.

Step 2: The test manager collaborated with the developing manager determine the weights of each quality risk using AHP method. The Analytical Hierarchy Process (AHP) is a powerful and flexible multi-criteria decision-making method that has been applied to solve unstructured problems in a variety of decision-making situations, ranging from the simple personal decisions to the complex capital intensive decisions.

In this case study, the calculation of quality risks weights is illustrated in Table 2. The number in each cell represents the value pair-wise relative importance: number of 1, 3, 5, 7, or 9 in row *i* and column *j* stands for that the stakeholder value in row *i* is equally, moderately, strongly, very strongly, and extremely strongly more important than the stakeholder value in column *j*, respectively. In order to calculate weight, each cell is divided by the sum of its column, and then averaged by each row. The results of the final averaged weight are listed in the bolded Weights column in Table 2. The sum of weights equals 1.

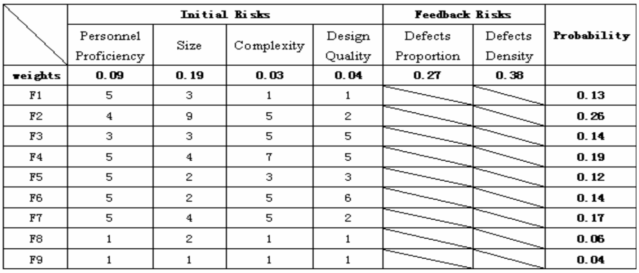
Table 2. Risk Factors' Weights Calculation-AHP



	Personnel Proficiency	Size	Complexity	Design Quality	Defects Proportion	Defects Density	**Weights**
Personnel Proficiency	1	1/3	3	3	1/3	1/5	**0.09**
Size	3	1	3	3	1	1	**0.19**
Complexity	1/3	1/9	1	1	1/7	1/9	**0.03**
Design Quality	1/3	1/7	1	1	1/7	1/9	**0.04**
Defects Proportion	3	1	7	7	1	1	**0.27**
Defects Density	5	3	9	9	1	1	**0.38**

If we were able to determine precisely the relative value of all risks, the values would be perfectly consistent. For instance, if we determine that *Risk1* is much more important than *Risk2*, *Risk2* is somewhat more important than *Risk3*, and *Risk3* is slightly more important than *Risk1*, an inconsistency has occurred and the result's accuracy is decreased. The redundancy of the pairwise comparisons makes the AHP much less sensitive to judgment errors; it also lets you measure judgment errors by calculating the consistency index (CI) of the comparison matrix, and then calculating the consistency ratio (CR). As a general rule, CR of 0.10 or less is considered acceptable[17]. In the case study, we calculated CR according to the steps in [17],the CR is 0.01, which means that our result is acceptable.

Table 3. Quality Risk Probability Calculation (Before System Test)



	Initial Risks				**Feedback Risks**		**Probability**
	Personnel Proficiency	Size	Complexity	Design Quality	Defects Proportion	Defects Density	
weights	**0.09**	**0.19**	**0.03**	**0.04**	**0.27**	**0.38**	
F1	5	3	1	1			**0.13**
F2	4	9	5	2			**0.26**
F3	3	3	5	5			**0.14**
F4	5	4	7	5			**0.19**
F5	5	2	3	3			**0.12**
F6	5	2	5	6			**0.14**
F7	5	4	5	2			**0.17**
F8	1	2	1	1			**0.06**
F9	1	1	1	1			**0.04**

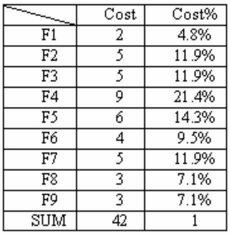
Step 3: The test manager's in collaboration with the developers, estimates the relative degree of risk factors associated with each feature on a scale from 1 to 9. An estimate of 1 means the risk factor influences the feature very little, while 9 indicates serious concerns should be paid about this risk factor. Initial Risks are risk factors we use to calculate the risk probability before the system testing and Feedback Risks such

as defects proportion and defects density are risk factors displayed during the testing process and serve to monitor and control the testing process. Qualitative risks such as Personnel Proficiency, Complexity, Design quality etc. are scored by the developing manager. Quantitative risks such as Size, Defects Proportion, Defects Density are scored based on project data, for example, if a feature's size is 6KLOC and the largest feature's size is 10KLOC, so the feature's size risk is scored as 9*(6/10)$\approx$5. The risk probability of Fi is $P_i = \frac{\sum_{j=1}^{n} R_{i,j} * W_j}{9} \in [0,1]$. $R_{i,j}$ is Fi's risk value of *jth* risk factor, W_j denotes the weight of *jth* risk factor. Table 3 will calculate the Probability of the total initial risks that comes from each feature before system test.

4.1.3 Testing Cost

The test manager estimates the relative cost of testing each feature, again on a scale ranging from a low of 1 to a high of 9. The test manager estimates the cost ratings based on factors such as the developing effort of the feature, the feature complexity, and the quality risks. Table 4 will calculate the percentage of total cost for each feature. In our case study, the estimating result is displayed in Table 4 and Fig 3 .

Table 4. Relative Testing Cost Estimation



	Cost	Cost%
F1	2	4.8%
F2	5	11.9%
F3	5	11.9%
F4	9	21.4%
F5	6	14.3%
F6	4	9.5%
F7	5	11.9%
F8	3	7.1%
F9	3	7.1%
SUM	42	1

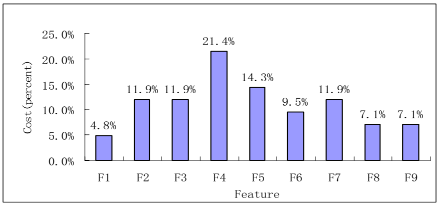


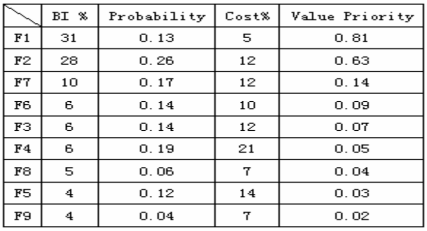
Fig. 3. Testing Cost Estimation Distribution

4.2 Put BI, Risk and Cost Together

This step is a multi-attribute decision and negotiation processing which is confirmed with the Decision Theory in VBSE. Once you enter the BI%, Probability and Cost% estimates into the spreadsheet (Table 5), it calculates a value priority number for each feature. The formula for the Value Priority column is:

$$Value\Pr iority = \frac{BI * \Pr obability}{Cost}.$$

Table 5. Value Priority Calculation



	BI %	Probability	Cost%	Value Priority
F1	31	0. 13	5	0. 81
F2	28	0. 26	12	0. 63
F7	10	0. 17	12	0. 14
F6	6	0. 14	10	0. 09
F3	6	0. 14	12	0. 07
F4	6	0. 19	21	0. 05
F8	5	0. 06	7	0. 04
F5	4	0. 12	14	0. 03
F9	4	0. 04	7	0. 02

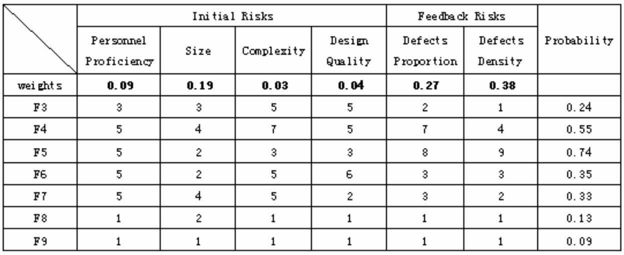
Sort the list of features in descending order by calculated priority. The features at the top of the list have the most favorable balance of BI, cost, and risk. Features with high BI, high quality risk and low cost have high testing priority and should be tested first. The key customer and developer representatives should review the completed spreadsheet to agree on the ratings and the resulting sequence. As we can see in Table

5, F1 and F2 have the highest Value Priority and should be tested first. They also have a stronger Pareto value, with 22% of the functions contributing 77% of the value.

4.3 Adjust Priority According to Feedback during Testing Process

Features' testing value priority provides the decision support for the test manager to enact the testing plan and adjust it according to the feedback of quality risk indicators, such as defects numbers and defects density and updated cost estimation. Usually, we collect defects data, re-estimate cost after one testing round and provide feedback to adjust the next testing round plan. Features with higher priority should be tested to satisfy the stop-test criteria first. When the features with highest priority satisfy the stop-test criteria, delete them from the spreadsheet, update feedback risk and cost estimation for features left and resort features according to the updated priority, continue this process when all features satisfy the stop-test criteria or it's the time to market. As the planning uncertainty is high for the first test round, the test manager decides to schedule the most valuable tests first so less important ones can easily be deferred if running out of time at the end of the cycle. This step helps to control progress toward SCS win-win realization which is according to the Control Theory of VBSE.

Table 6. Quality Risk Probability Calculation (After Round 1)



	Initial Risks				Feedback Risks		Probability
	Personnel Proficiency	Size	Complexity	Design Quality	Defects Proportion	Defects Density	
weights	**0.09**	**0.19**	**0.03**	**0.04**	**0.27**	**0.38**	
F3	3	3	5	5	2	1	0.24
F4	5	4	7	5	7	4	0.55
F5	5	2	3	3	8	9	0.74
F6	5	2	5	6	3	3	0.35
F7	5	4	5	2	3	2	0.33
F8	1	2	1	1	1	1	0.13
F9	1	1	1	1	1	1	0.09

Table 6 shows the quality probability calculation after testing round 1 when F1, F2 have satisfied stop-test criteria, so we delete them from the spreadsheet and re-sort the remaining features.

Each organization has its own stop-test criteria. In our case study, the company has 4 criteria, and they are: test cases coverage rate is 100%, all planned test cases executed and passed; no defects with impact equal to, or below severity level 3 are detected during at least one day of continuous testing, and to satisfy the quality goal of 0.2 defects/KLOC when released.

5 Performance Evaluation

In our case study, the test manger plans to execute 4 rounds of testing. During each round, test groups focus on 2-3 features with the highest current priority, and the other features are tested by automated tools. The testing result is when the first round is over, F1 and F2 satisfy the stop-test criteria, when the second round is over, F3, F6, F7 satisfied the stop-criteria, when the third round is over, F4, F8 satisfied the stop-test criteria, and the last round is F5 and F9. And initial estimating testing cost and actual testing cost comparison can be shown in Fig 4**.**

If we regard the testing activity as an investment, its value is realized when features satisfy the stop-test criteria. As we can see in Fig 5 and Fig 6 , when we finished the Round 1 testing, we earned 59.2% BI of all features, at a cost of only 19.8% of the all testing process, the ROI is as high as 1.99. During the Round 2, we earned 22.2% BI, cost 25.3% effort, and the ROI became negative as -0.12. We also can see, from Round 1 to Round 4, both the BI earned line and the ROI line is descending. Round 3 and Round 4 earn only 18.5% BI but cost 54.9% effort. This shows that the Round 1 testing is the most cost effective. Testing the features with higher value priority first is especially useful when the market pressure is very high. In such cases, one could stop testing after finding a negative ROI in Round 1. However, in some cases, continuing to test may be worthwhile in terms of customer-perceived quality.

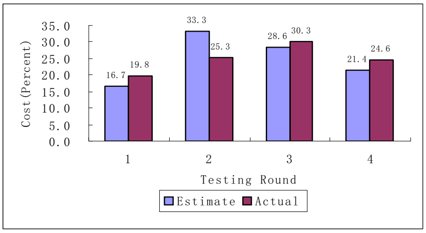
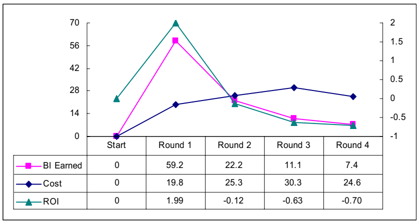


Fig. 4. Initial Estimating Testing Cost and Actual Testing Cost



	Start	Round 1	Round 2	Round 3	Round 4
BI Earned	0	59.2	22.2	11.1	7.4
Cost	0	19.8	25.3	30.3	24.6
ROI	0	1.99	-0.12	-0.63	-0.70

Fig. 5. BI, Cost and ROI between Testing Rounds

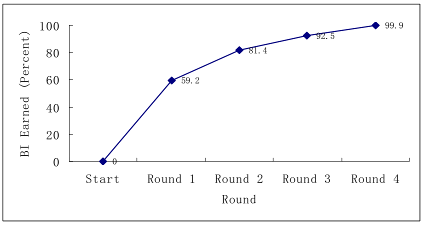
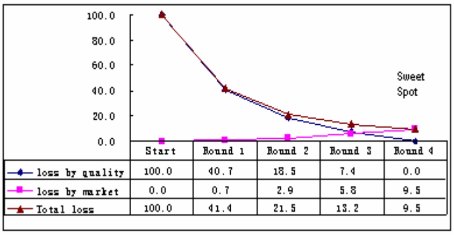


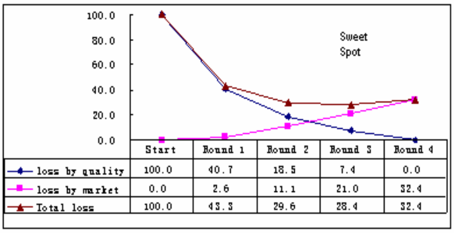
Fig. 6. Accumulated BI Earned During Testing Rounds

In our case study, we use a simple function as follows to display the market pressure's influence to BI: $\mathrm{Pr}\,esent\,BI = InitialBI/(1+\mathrm{Pr}\,essureRate)^{Time}$. Time represents the number of unit time cycle. A unit time cycle might be a year, a month, a week even a day. For simplicity, in our case study, the unit time cycle is a testing round. Pressure Rate is estimated and provided by market or product managers, with the help of customers. It represents during a unit time cycle, what is the percentage initial value of the software will depreciated. The more furious the market competition is, the larger the Pressure Rate is. As we can see from the formula above, the longer the time is, the larger the Pressure Rate is, the smaller is



	Start	Round 1	Round 2	Round 3	Round 4
loss by quality	100.0	40.7	18.5	7.4	0.0
loss by market	0.0	0.7	2.9	5.8	9.5
Total loss	100.0	41.4	21.5	13.2	9.5

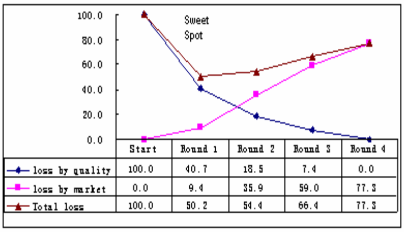
Fig. 7. BI Loss (Pressure Rate=1%)



	Start	Round 1	Round 2	Round 3	Round 4
loss by quality	100.0	40.7	18.5	7.4	0.0
loss by market	0.0	2.6	11.1	21.0	32.4
Total loss	100.0	43.3	29.6	28.4	32.4

Fig. 8. BI Loss (Pressure Rate=4%)

the present BI, and the larger the loss BI caused by market erosion. In our case study, Due to we calculate the relative business importance, the initial total BI is 100(%). When the Round n testing is over, the loss BI caused by market erosion is $100-100/(1+\Pr essureRate)^{n}$. On the other hand, the earlier the product enters the market, the larger the loss caused by poor quality. Finally, we can find a sweet spot (the minimum) from the combined risk exposure due to both unacceptable software quality and market erosion.



	Start	Round 1	Round 2	Round 3	Round 4
◆ loss by quality	100.0	40.7	18.5	7.4	0.0
■ loss by market	0.0	9.4	35.9	59.0	77.3
▲ Total loss	100.0	50.2	54.4	66.4	77.3

Fig. 9. BI Loss (Pressure Rate=16%)

We assume three Pressure Rates 1%, 4% and 16% standing for low, median and high market pressure respectively in Fig 7,8,9. When market pressure is as low as 1% in Fig7, the total loss caused by quality and market erosion reaches the lowest point (sweet spot) at the end of the Round 4.When the Pressure Rate is 4%, the lowest point of total loss is at the end of Round 3 in Fig 8, which means we should stop testing and release this product even F5 and F9 haven't reached the stop-test criteria at the end of Round 3; this would ensure the minimum loss. When the market pressure rate is as high as 16% in Fig 9, we should stop testing at the end of Round 1.

6 Discussion of Limitation

- This method is based on the assumption that the features for prioritization are independent and the absentness of any features doesn't influence the testing and releasing of others. If the prioritization has already been done in the implementation stage, testing is just a process that ensures the software satisfies customers' needs, instead of a process of trade-offs between quality and time-to-market pressures, and our method becomes meaningless for this situation.

- This method is applicable for business critical projects, not suitable for safety critical domains. In safety critical domains, software organizations follow strict guidelines for testing. Sacrifice of quality to satisfy the business goal might be unethical and unprofessional in such situations.
- This method depends highly on accurate cost estimation to enact testing plans and testing rounds, however, as practical experiences show, it is often not possible to anticipate all influence factors at the beginning of a test cycle. Re-estimation during the testing process gives a solution to this problem. Updated testing cost estimation during the testing process produces more accurate testing priorities.

- About the method validation, we used a simple function to display the market pressure's influence to BI in our case study; however, this form of market pressure functions as well[20]. The choice of which function to use depends on different project business cases.

7 Conclusion and Future Work

Software testing is a very resource-intensive activity in software development and not always organized to maximize business value. This paper has demonstrated a value-based approach for prioritizing features for testing which aligns the internal test process with the value objectives coming from the customers and the market. This involves prioritizing features based on their business importance, quality risk, and testing cost of each feature; adjusting feature's value priority during the testing process; and providing stop-testing decision criteria based on the market pressure. In the organization's case study, this method helps the test manager to identify features with high business importance, high quality risk and low cost, to focus testing effort on these features and to control and adjust testing plan toward SCSs win-win realization. The result shows that this method can help to improve ROI of testing investment at the early stage, especially when the market pressure is high.

In 2009, we plan to apply this testing approach to a set of e-Services projects at USC to get more empirical validation.

Acknowledgements

This work is supported by the National Natural Science Foundation of China under Grant Nos. 60573082, 60873072, and 60803023; the National Hi-Tech R&D Plan of China under Grant Nos. 2006AA01Z182 and 2007AA010303; the National Basic Research Program (973 program) under Grant No. 2007CB310802.

References

1. Boehm, B., Basili, V.R.: Software Defect Reduction Top10 List. IEEE Computer 34(1), 135–137 (2001)
2. Boehm, B.: Value-Based Software Engineering: Overview and Agenda. In: Value-Based Software Engineering. Springer, Heidelberg (2005)
3. Ramler, R., Biffl, S., Grunbacher, P.: Value-Based Management of Software Testing. In: Value-Based Software Engineering, pp. 226–244. Springer, Heidelberg (2005)
4. Beizer, B.: Software Testing Techniques, 2nd edn. International Thomson Computer Press, New York (1990)
5. Boehm, B.: Value-Based Software Engineering. ACM Software Engineering Notes, 28(2) (2003)
6. Bullock, J.: Calculating the Value of Testing. Software Testing and Quality Engineering 2(3), 56–62 (2000)
7. Pyster, A.B., Thayer, R.H.: Software Engineering Project Management 20 Years Later. IEEE Software 22(5), 18–21 (2005)
8. Boehm, B., Jain, A.: An Initial Theory of Value-Based Software Engineering. In: Value-Based Software Engineering, pp. 16–37. Springer, Heidelberg (2005)
9. Boehm, B., et al.: Using the WinWin spiral model: a case study. IEEE Computer 31(7), 33–44 (1998)
10. Boehm, B.: A Spiral Model of Software Development and Enhancement. IEEE Computer 21(5), 61–72 (1988)

11. Boehm, B., Huang, L.G.: Value-Based Software Engineering: A Case Study. IEEE Computer 36(3), 33–41 (2003)
12. Amland, S.: Risk Based Testing and Metrics. In: 5th International Conference EuroSTAR 1999, Barcelona, Spain (1999)
13. Raz, O., Shaw, M.: Software Risk Management and Insurance. In: Proceedings of the 23rd International Conference on Software Engineering (Workshop on Economics-Driven Software Engineering Research) (2001)
14. Lee, K., Boehm, B.: Empirical Results from an Experiment on Value-Based Review (VBR) Processes. In: International Symposium on Empirical Software Engineering (2005)
15. Boehm, B., et al.: The ROI of Software Dependability: The iDAVE Model. IEEE Software 21(3), 54–61 (2004)
16. Wiegers, E.K.: First Things First: Prioritizing Requirements. Software Development 7(10), 24–30 (1999)
17. Saaty, T.L.: The Analytic Hierarchy Process. McGraw-Hill, New York (1980)
18. Boehm, B., et al.: Software Cost Estimation with COCOMOII, Har/Cdr th edn. Prentice-Hall, Englewood Cliffs (2000)
19. Chulani, S., Boehm, B.: Modeling Software Defect Introduction and Removal: CO-QUALMO (COnstructive QUALity MOdel),Technical Report, University of Southern California (2002)
20. Huang, L., Boehm, B.: How Much Software Quality Investment Is Enough: A Value-Based Approach. IEEE Software 23(5), 88–95 (2006)
21. Srikanth, H., Williams, L.: On the Economics of Requirements-Based Test Case Prioritization. In: EDSER 2005, St. Louis, Missouri, USA (2005)
22. Wang, Q., et al.: BSR: A Statistic-based Approach for Establishing and Refining Software Process Performance Baseline. In: Proceedings of the 28th International Conference on Software Engineering, Shanghai, China (2006)
23. Wang, Q., et al.: An Empirical Study on Establishing Quantitative Management Model for Testing Process. In: Wang, Q., Pfahl, D., Raffo, D.M. (eds.) ICSP 2007. LNCS, vol. 4470, pp. 233–245. Springer, Heidelberg (2007)

Subcontracting Processes in Software Service Organisations - An Experience Report

Jakub Rudzki[1], Tarja Systä[2], and Karri Mustonen[1]

[1] Solita Oy, Satakunnankatu 18 A, 33210 Tampere, Finland
firstname.lastname@solita.fi
[2] Tampere University of Technology, Korkeakoulunkatu 1, 33101 Tampere, Finland
tarja.systa@tut.fi

Abstract. Software systems and projects have become more and more distributed. This emphasises the need for ensured software quality, which impacts e.g. customer satisfaction, development costs, and delivery schedules. Concerns about the software quality become even more important in the case of subcontracted software projects, and in the case of multi-site projects in particular. In this paper we describe a practise-derived process to assess the potential subcontractors at their selection stage and later to evaluate the cooperation with them. For both purposes, particular criteria suites and frameworks are proposed. The criteria include software-specific and productivity metrics, but also more qualitative criteria. We report initial observations from usage of the process in industry. The assessment results are expected to help in making decisions about subcontractor selection, and later on assigning specific types of projects to specific subcontractors based on their suitability.

1 Introduction

The usage of subcontractors in software development is not a new concept. Companies decide to introduce subcontracting in their projects for various reasons. Often the main reason is the cheaper labour, also a lack of qualified specialists may be a reason. Additionally, plans for expansion in another country can start from establishing an off-shore centre [1,2]. Subcontracting partners are often used in product development, but rarely in software companies that offer software services themselves. In such an organisation the subcontracting partner must be extremely reliable and able to provide high quality services, so that the software service company can provide services transparent for the end customer, who in most of the cases, is not interested in internal organisation of the offered projects. A correctly selected subcontracting partner can increase chances of successful relationship with software service customer, which is often based on trust [3]. In this paper we are presenting a case study of subcontracting selection and evaluation process that is based on the case of software service provider company Solita [1].

[1] www.solita.fi

Q. Wang et al. (Eds.): ICSP 2009, LNCS 5543, pp. 224–235, 2009.

Solita is a software service company offering its customers customised solutions. Those solutions are developed in projects where to some extent subcontracting partners are used. In order to find the most suitable subcontracting partners we have developed and successfully used a process for selection of subcontractors. The process as such is based on best practises of subcontracting. However, it is adjusted to specific needs of software service company. We also present a cooperation process that we have used for projects developed together with the subcontracting partners. Together the selection and cooperation processes create a general process of subcontracting in software service companies.

The rest of this paper is structures as follows, in Sect. 2 we discuss the most relevant research related to presented process. In Sect. 3 we present the overview of the cooperation process with subcontracting partners. In Sect. 4 selection process is presented. Ways of everyday cooperation with subcontracting partners are presented in Sect. 5. Then in Sect. 6 we present our experiences in Solita with using the processes. Finally, we present our conclusions in Sect. 7.

2 Related Works

Software development subcontracting has been discussed for some time already in the literature. There are different strategies presented, including outsourcing based on cooperation and competition of subcontracting partners [4]. The strategy based on a good relationship with the subcontracting partners is one of the aims of the subcontracting process discussed in this paper. Also the process to be discussed is in line with good practises of outsourcing [5,6]. For example, the outsourcing process can be assessed using the Outsourcing Management Maturity Model [5]. Also most of the seven outsourcing practises presented by Reifer [6] can be found in the subcontracting process discussed in this paper. One part of the subcontractor cooperation process is the subcontractor selection. Subcontractor selection methods have been analysed by Assmann and Punter, which resulted in formulating Method for Assessing Software Subcontractors (MASS) [7]. MASS has elements of subcontractor selection that are also utilised at the general level in the described subcontractor selection process. However, while MASS focuses on particular projects, subcontracting selection process we discuss focuses on long-term cooperation that should result in multiple projects. This different goal results in different assessment criteria that are more general and technology-related, rather then specific for a particular project.

Also high level frameworks for acquisition of services CMMI [8,9], CMMI for Acquisition [10], EuroMethod [11] / ISPL [12], or models focusing on social and cultural aspects of subcontracting eSCM-SP / SQM-CODE [13] are a good base for company-specific processes. The process described in this paper contains main elements of those frameworks (i.e. selection, monitoring, termination); however, we present details of a process specific to software service organisation. Also we do not focus explicitly on risk factors, e.g. legal risks, as those should be addressed in contracts, while quality related risks are addressed by careful selection and cooperation processes that are described in the following sections.

3 Process Overview

We see the cooperation with subcontractors as a process that starts from the search of a suitable company and then evolves into a long-lasting cooperation. The concept of this process is depicted in Fig. 1

In the selection phase, which consists of the active search through different means and the selection of the companies that will be chosen as potential subcontractors, we use a number of criteria that are aiming at selecting the most suitable subcontracting partner. The criteria are to reveal the general 'health' of the company, e.g., from the financial point of view [14], but also their competence in a given technological field. The subcontractor quality can be understood in many ways. Here we will focus on attributes that help to identify whether a particular company will be a suitable subcontracting partner or not. Such assessment is not easy. Therefore the process does not aim at providing a definite answer in the style 'Yes, that company will be a good partner for us' or 'No, that company certainly will fail in cooperation with us'. The process should be used as additional help when decisions are made. There are many other factors that are used to select a new business partner [14], including location, culture, or political context. However, we concentrate on the quality aspects of such cooperation as for a software service company software quality is a very important factor in maintaining a good relationship with end customers. The selection phase results in a list of subcontractors that have been selected for cooperation. The subcontractors can be categorised based on their technical capabilities, prices, size etc. Categorisation may help especially in the case of large number of subcontractors.

In the evaluation phase, the work done with certain subcontractors is constantly evaluated. In order to develop good cooperation and mutual understanding with the subcontractor the projects using subcontractors should be continuously evaluated, which is in line with general good practises in outsourcing [6,5] and aiming at partnership with the subcontractor [15]. The criteria used for evaluation are reduced to minimum in order to reduce the costs of the measurements in the business settings. The evaluation results can be used to give feedback to the subcontractor and adjust project arrangements in the ordering company. Projects utilising subcontractors can be arranged in various ways; the project can be carried out entirely by the subcontractor, only with supervision from the ordering company, or with mixed teams of subcontractors and ordering company employees. The work evaluation should help both sides to better meet their expectations. Also based on the classification the subcontractors can be used more efficiently in the future projects.

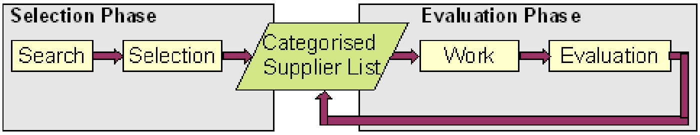


Fig. 1. Subcontractor cooperation process

4 Selection Phase

The aim of the selection phase is to find as many potential subcontractors as possible, then through multi-step selection process choose subcontractors the most suitable for cooperation. This phase results in a list of subcontractors who can be categorised based on their suitability for software service projects. The details of the selection phase are depicted in Fig. 2. Many potential companies are rejected, but it only means that they are not suitable at given time for specific needs. It is possible that they can be suitable in the future if the cooperation context changes, e.g., their size or competence change.

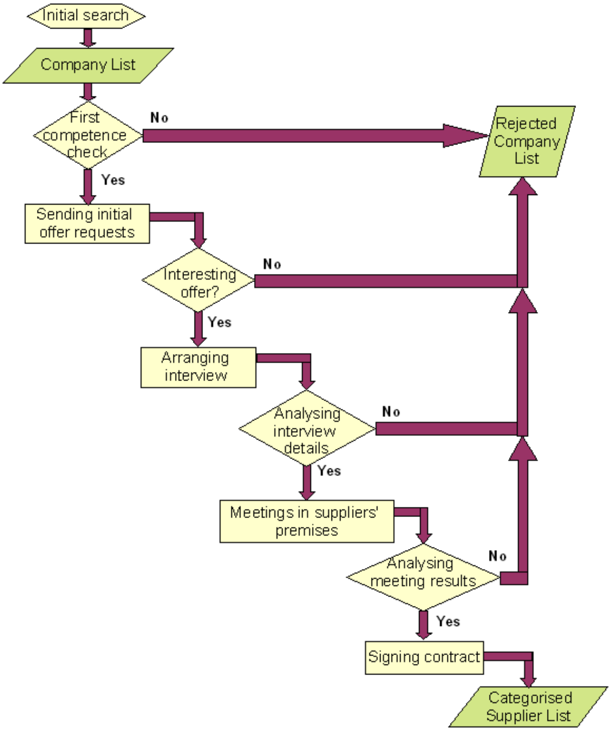


Fig. 2. Subcontractor selection phase

4.1 Selection Phase Walkthrough

The selection phase starts from *initial search*, which can be done through Internet search engines, personal recommendations, social network portals and any other means allowing to find potential suppliers of subcontracting services. Next, in *first competence check* the companies are checked initially for their competences. The check is done mostly based on the information provided by the company in their publicly available promotional materials, particularly web-pages. That check allows, for example, to filter out recommended companies that changed their profile, for instance, from software supplier to hardware. After the company list is narrowed down, *sending initial offer requests* can start. In the offer request the company is expected to state whether they are interested in providing subcontracting services, give basic information on their operations and provide price range for their services. Companies that express their interest in cooperation on acceptable conditions are selected for further negotiations.

The next step is *arranging interview* when details of possible cooperation are discussed. The interview is typically organised as a teleconference, because of the distance between the parties. As a preparation for the interview the company is asked to fill in a questionnaire with questions allowing to gather detailed information on their competences and assess their quality. The details of the questions are discusses in Sect. 4.2. During the teleconference main objectives of the subcontracting are discussed and both parties can gather more information about each other. Subsequently the *interview analysis* is performed. The analysis is done based on the material gathered before and during the interview. The answers collected from the questionnaire are analysed and used as a supportive tool to further select the companies.

Finally, *meetings in suppliers' premises* are organised. Those meetings are used to get to know better by the parties as well as negotiate the details of cooperation. Subsequently *analysis of meetings* are performed to select final companies with whom *contracts are signed*. Signing a contract ends the selection phase and starts the evaluation phase with new subcontractors.

4.2 Criteria

The questions used to assess the potential subcontractor quality are based on information that should be easily available, at least at the stage of considering the given company as a subcontractor. No information is asked that would be too business-specific. Thus, obtaining that details should not be a problem. In fact, if the potential subcontractor was reluctant to provide such information, that would also provide some picture of the company policies and culture. However, we assume that the potential subcontractor is willing to provide answers to the questions collected in the questionnaire.

Assessing the subcontractor quality can be done based on well established standards, e.g., CMMI-ACQ [10], MASS [7], eSCM-SP / SQM-CODE [13], or others as described in Sect. 2. The use of any quality standards by the potential subcontractor also allows assessing the potential quality of their work, as specific quality attributes relate more to certain processes [16]. For assessing a general quality level of the subcontractor, before any project is carried out by that subcontractor, we suggest the following selected criteria:

1. *Number of qualified engineers*, which provides information of the potential talent pool available for the ordering company. That number can also be used to make other attributes relative, which helps to compare results of companies of different sizes.
2. *Median number of years of experience*, which provides general information on the experience level that can be expected from the available engineers.

3. *Number of customers*, which allows for assessing how active is the company with winning contracts from customers.
4. *Percentage of long-lasting customers*, which allows for finding out whether the subcontractor aims at good relationship with their customers.
5. *Number of projects done per year to the number of specialists*, which provides information on typical projects done in the company concerned. A high level of this metric indicates that the company has many small projects. That, in turn, can indicate flexibility in changing the projects but shows that the potential subcontractor does not have experience with long-lasting projects, e.g., a new product development. On the other hand low value of the metric indicates the opposite. The actual values and their distinction between high and low values depends on what the ordering company expects.
6. *Average length of a project*, which also provides information on the type of projects done in the subcontractor company. That information must be evaluated in the context of the prosperous projects that are planned to be subcontracted.
7. *Average size of the project*, which allows finding out what kind of size projects the company is able to carry out. Also that measure may indicate in what kind of teams the professionals concerned are typically working.
8. *Expertise in evaluated technology*, which provides directly information on the experience in given technology. That category should be adjusted and possibly fine-grained to the level suitable for the project(s) that are to be subcontracted.
9. *Processes used for software development*, which allows to asses the awareness of the prosperous subcontractor about software development methodologies. If a company follows certain processes, it indicates that they thought through their practises and want to improve them. The process maturity can also be characterised at different levels [17]. In that case complete lack of any processes can be seen as a negative factor during the selection phase.
10. *Number of projects fully coordinated*, which provides information on the project specialisation of the given company. That metric shows whether the company provides people to work in teams (maybe virtual teams) coordinated by the customer ordering the project, or the potential subcontractor is able to deliver a product based on requirements given by the customer. Also this metric should be evaluated in the context of the prosperous projects to be subcontracted.
11. *Change management practises*, which allows to assess whether the company has internal processes defined and whether they are consistent with the development processes used by the potential subcontractor.

5 Evaluation Phase

When a relationship with a subcontracting company (or a group of companies) is established and projects are done by those companies, it should be possible to assess the actual quality of those subcontractors based on their work. Having in mind that we discuss the subcontracting of software service projects, we should define software-specific criteria for subcontractor quality measurement. The work evaluation process is depicted in Fig. 3.

The list of subcontractors that are cooperating with the contractor results from the selection process. The subcontractors can be also categorised based on their competences, prices etc. But later during the cooperation, the subcontractors can additionally be categorised based on the quality of their work. That information can be used by the managers to select a better suitable subcontractor for specific projects. Also it can happen that the actual performance of a subcontractor is found not sufficient and the cooperation with such company will be terminated.

Before a new project is started, a team is selected from personnel available from certain subcontractor. That stage is similar to any project team selection with the difference that some or all team members are from the subcontractor organisation. When the project is ongoing the project team can internally evaluate project progress and adjust internal practises to improve the project performance. The internal project feedback can be easily done for projects following the Scrum/Agile methodology [18], where such feedback sessions are part of the process. Finally, after the project has completed its results can be evaluated. The project can be evaluated in multiple dimensions. Naturally software quality metrics can be used to assess the quality of the produced software. However, additionally other criteria can be used to determine whether project was successful and to what extent the subcontractor work influenced the success or failure.

The most general and straightforward metric of the project performance is the profit. It is difficult to measure the project performance based on the number of lines of code, as it can promote lengthy difficult to maintain code (as pointed out by Martin Fowler [19]). But the profitability of the project does not necessary tell much about the quality of the produced software. To measure the software quality it is possible to select a few software metrics [20] and process metrics [21]. Also specific set of metrics for outsourcing has been proposed, for example POMADO [22]. However, POMADO consists very detailed level of

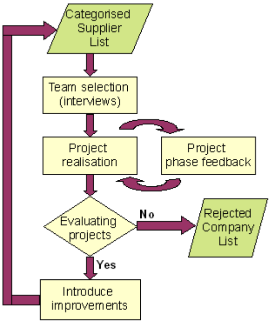


Fig. 3. Subcontractor evaluation phase

metrics, mostly focusing on communication, which as such is very important, however in this work we approach the subcontractor evaluation from different angles. We have selected metrics that can be easily gathered from the produced code, faults statistics, cost of the project (measured in hours or actual amount of money). The measures must be generic for any type of the projects and subcontractors, they also do not require any input from the subcontractor as the data is readily available in the ordering company. Also more qualitative criteria should be used reflecting, for example, customer satisfaction.

The primarily objective of these metrics is to compare the work of different subcontractors and categorise them better. That information should be used to more consciously choose subcontractors for specific projects. The measurements are not meant to create a ranking of the subcontractors as their specific competences may differ greatly and comparing them directly would not be reasonable. In addition to finding out the quality of work of particular subcontractors such measurements can provide additional information on specific projects. For example, comparison of the results for projects that were done entirely by one subcontractor (single-site projects) with multi-site projects can provide information on the impact of the multi-site setting on the project quality. That information will in turn allow for more conscious choice of the project setting, or the influence of the setting on the project in the case when the setting cannot be altered. It is possible to imagine that based on comparative data it can be recognised that projects with profitability margins below certain threshold should not be taken in certain settings. For example, if multi-site projects turn out to have much different profitability comparing to one-site projects, some projects should not be done in multi-site arrangements.

5.1 Quality Monitoring Metrics

There are numbers of software and productivity metrics that can be used for measuring different aspects of the software quality, however, in the context of commercial projects only most useful metrics should be calculated. Therefore we suggest a set of metrics that we see as preferred and they should be used for all projects. But we also specify optional metrics that can be used in order to gather more detailed information on specific projects or subcontractors. The preferred metrics that we suggest to measure project quality include:

1. Profit margin of a project measures the value of a project from the business point of view, regardless of the project technical quality. Also this metric can be used to determine well performing projects and possibly identify project specifics, e.g., size, number of sites, length, etc., that influence projects profitability.
2. The ratio of hours spent for communication to the total work hours. That metric should indicate differences in communication patterns between projects. Especially multi-site projects are assumed to require much more communication than single-site projects [23,24]. By measuring the number of hours spent for communication it should be possible to compare results

between single- and multi-site projects, as well as well performing projects and badly performing ones. For example, if a multi-site project starts performing badly and it is also seen that there is not much communication in the project, preventive measures can be taken. Also it should be noted that communication is understood as any forms of communication, including face-to-face meetings, teleconferences, and also virtual chats in any instant messengers (IM).

3. The percentage of project team members from a subcontractor company. That metric reflects the team organisation and extend to which the subcontractor had contributed in the work.

The optional metrics that we suggest include:

1. Number of implemented required features to the total number of required features for specified time. Each project has a number of features which are recognised as required, so features that if not implemented, the final product will be significantly less valuable for the end customer. This metric allows for measuring what is the percentage of implemented features. The number indicates how well the team managed to fulfil the agreed requirements.
2. Percentage of the agreed performance target achieved. The performance target differs from product to product, but for a web application it can be the number of users served in specified time period. The percentage of the achieved performance target tells how well the project managed to realise important goal from the user point of view. Additionally, as the performance goal should be agreed beforehand, this metric also tells how good is the team in the goals estimation.
3. Number of faults per 1000 lines of code; where faults are gathered through the whole life-cycle of the software development. Providing that compared projects use the same technology (programming language), the metric will give a general picture whether the created code follows the defined requirements.
4. Number of fault fixes delivered. That metric can be used to monitor fault handling pattern in the project, i.e., to see whether the newly reported faults are resolved promptly.

Additional qualitative criteria include:

1. Project methodology, which may allow to link correct practises with certain types of projects.
2. Customer satisfaction, which in software service project is very important, yet difficult to measure. The satisfaction level usually can be obtained from the customer when the project work is done and the end customer is asked for feedback.

6 Process Applicability - Experiences to Date

The described process derives from our real-life experiences of subcontractor selection and later cooperation with them. The process version presented in this work refines areas, which should be addressed during the selection phase. The subcontractor cooperation evaluation is an ongoing process and the set of used criteria may change in the future.

The selection process has been used for two years now since its initial version. We are able to distinguish two different cycles of using the selection process with different business and technological requirements. One requirement valid in all cycles was the location of the potential subcontractor partner. From our point of view the partner should be in a 'nearshore' [25] location understood as in distance allowing business trips in one day, similar time zone and culture. In the first selection cycle we initially selected 26 companies from about 200, from which we contacted 12 companies, then for final visits were selected 5 companies. That process resulted in cooperation with a few partners. The second selection cycle is still ongoing, but so far we have initially selected 77 out of about 150 companies, then contacted 38 companies, then interviewed 8 companies and visited 6 different companies. As it can be seen the selection process greatly reduces the number of companies at each stage. The selection is based on the collected data as well as experience and judgement of the decision makers, that's why we don't use a point system where potential subcontractors collect points. That approach does not seem feasible in a selection process that is more qualitative than quantitative. The process is used whenever there is a need for new subcontracting partners, also already collected company profiles are re-evaluated if the cooperation context changes

The evaluation of cooperation with existing subcontracting partners is ongoing and data is collected. Currently the collected data is not enough to draw any general conclusions or recognise new patterns. We believe that the selected criteria are enough for evaluating the cooperation with subcontractors and improve the cooperation when needed. So far there have not been noticed any significant correlation in particular criteria levels and project success, however, we also have not obtained data for any extreme project cases of failure or significant success. Such extreme cases should provide more information and possible correlation with particular criteria. Even with initial results we are able to observe some patterns in specific criteria. For example, the time spent for communication reflects the changes in the project arrangement. In one project, the ratio of time used for communication to the total project time was typically about 10%. However, the ratio increased (to about 13%) when the team was dislocated to different sites.

Additionally, the results show that during absence of the project manager the communication related to the coordination has lowered (to about 5%). It is not possible to draw any general conclusions based on this data, but it shows some correlation to the events in that particular project. Also differences between the few analysed projects have been observed. The communication needs in projects differed from 9% of overall time, to 25%. The lower communication effort was noticed in smaller projects, while the higher in the bigger ones and more distributed. However, all of those projects were otherwise moderately successful, so no general conclusions can be drown at this point. When data for many projects and many subcontractors is collected, then it should be possible to conclude more generally and possibly recognise certain patterns.

7 Conclusions

We have presented a process and criteria allowing for assessing subcontractor performance through the whole cooperation cycle. Starting from selection of the subcontractor where particular criteria can be used to perform better selection, to end with continuous evaluation of cooperation with a subcontractor. The presented processes are aiming at the needs of software service companies that particularly strive to establish trustful relationships with their customers regardless of usage of the subcontractors in projects. Therefore processes helping to build quality-based cooperation with subcontracting partners directly reflect in relationships with the end customers of the software service company. The presented processes use multiple criteria of different kinds in order to cover different aspects of selection and cooperation with subcontracting partners.

Based on the experiences from subcontractor selection we refined previously-used metrics and created a set that will be used in the future selection processes. The evaluation process using the defined mandatory criteria is currently being done. The initial results indicate their relevance to monitoring the project work and when generalised also the subcontractor work. As the cooperation with subcontracting partners has been successful in several completed projects, we believe that the selection and cooperation process is correct. But still the selected criteria and processes should improve this cooperation even more. As future work, we will analyse results obtained from industrial usage of the process and possibly improve it. Additionally, we would consider creating a tool providing automatically the needed reports using data from different IT systems of the company.

References

1. Bardhan, A.: Managing globalization of R&D: Organizing for offshoring innovation. Human Systems Management 25(2) (2006)
2. Harvey, M.G., Griffith, D.A.: The role of globalization, time acceleration, and virtual global teams in fostering successful global product launches. Journal of Product Innovation Management 24(5), 486–501 (2007)
3. Hoch, D.J., Roeding, C.R., Purkert, G., Lindner, S.K.: Secrets of Software Success: Management Insights from 100 Software Firms Around the World. Harvard Business Press (2000)
4. Timothy, M.: Laster. Balanced Sourcing – Cooperation and Competition in Supplier Relationships. Jossey-Bass (1998)
5. Power, M.J., Desouza, K.C., Bonifazi, C.: Developing superior outsourcing programs. IT Professional 7(4), 32–38 (2005)
6. Reifer, D.J.: Seven hot outsourcing practices. IEEE Softw. 21(1), 14–16 (2004)
7. Assmann, D., Punter, T.: Towards partnership in software subcontracting. Comput. Ind. 54(2), 137–150 (2004)

8. Software Engineering Institute, Carnegie Mellon University. Capability maturity model integration (CMMI) (2007), http://www.sei.cmu.edu/cmmi/index.html
9. Chrissis, M.B., Konrad, M., Shrum, S.: CMMI Guidelines for Process Integration and Product Improvement. Addison-Wesley Longman Publishing Co., Inc., Boston (2003)
10. Software Engineering Institute, Carnegie Mellon University. Cmmi for acquisition, version 1.2. Technical Report CMU/SEI-2007-TR-017, SEI (2007)
11. Euromethod Project. Euromethod version 1 reference manual (July 1996), http://projekte.fast.de/Euromethod
12. ISPL Project. Ispl - information services procurement library, http://projekte.fast.de/ISPL/
13. Siakas, K.V., Balstrup, B.: Software outsourcing quality achieved by global virtual collaboration. Software Process: Improvement and Practice 11(3), 319–328 (2006)
14. Östring, P.: Profit-Focused Supplier Management: How to Identify Risks and Recognize Opportunities. Amacom (2003)
15. Lee, J.-N., Huynh, M.Q., Kwok, R.C.-W., Pi, S.-M.: It outsourcing evolution—: past, present, and future. Commun. ACM 46(5), 84–89 (2003)
16. Ashrafi, N.: The impact of software process improvement on quality: in theory and practice. Inf. Manage. 40(7), 677–690 (2003)
17. Humphrey, W.S.: Characterizing the software process: A maturity framework. IEEE Softw. 5(2), 73–79 (1988)
18. Paasivaara, M., Durasiewicz, S., Lassenius, C.: Using scrum in a globally distributed project: a case study. Software Process: Improvement and Practice 13(6), 527–544 (2008)
19. Fowler, M.: Cannot measure productivity (August 2003), http://martinfowler.com/bliki/CannotMeasureProductivity.html
20. Kan, S.H.: Metrics and Models in Software Quality Engineering. Addison-Wesley Longman Publishing Co., Inc., Boston (2002)
21. Humphrey, W.S.: The software quality profile, http://www.sei.cmu.edu/publications/articles/quality-profile (accessed in 02.2008)
22. Radoiu, D., Vajda, A.: Process-oriented metrics for application development outsourcing. a practitioner's approach. Studia Univ. Babes-Bolyai, Informatica XLIX(1) (2004)
23. Paasivaara, M.: Communication needs, practices, and supporting structures in global inter-organizational software development projects. In: Proceedings of the IWGSD at the 25th ICSE, Portland, Oregon, pp. 59–63 (2003)
24. Mockus, A., Herbsleb, J.: Challenges of global software development. In: METRICS 2001, Washington, DC, USA, p. 182. IEEE Computer Society Press, Los Alamitos (2001)
25. Carmel, E., Abbott, P.: Why 'nearshore' means that distance matters. Commun. ACM 50(10), 40–46 (2007)

On Reducing the Pre-release Failures of Web Plug-In on Social Networking Site

Xingliang Yu, Jing Li, and Hua Zhong

Institute of Software, Chinese Academy of Sciences
Graduate University of Chinese Academy of Sciences
P.O.Box 8718, Beijing 100190, China
yuxl@otcaix.iscas.ac.cn

Abstract. In recent years, web plug-ins have been flourishing social networking sites. Web plug-in is successful since it results in unique user experience, and promotes the fast-pace innovation of web technologies. However, the plug-ins developed by end users also introduces many new problems to both networking and software engineering fields. One of the key problems is pre-release failure. In other words, the failures that we can avoid before software release are usually found after the release. However, existing methods fail to avoid the pre-release failures of web plug-ins. To do this, this paper introduces an experimental technology, namely release-waiting farm. It not only maintains the free and creative environment of end user development, encouraging them to deliver plug-ins, but effectively formalizes their development process, thus provide long-term benefit to both end users and social networking sites.

Keywords: end user development, pre-release failure, social networking site, web plug-in.

1 Introduction

Social networking site (SNS) is virtual community that connects people with friends and others who work, study, and live around them. In very recent years, SNS has experienced periods of growth with participation expanding at rates 20% a month [16], and thus become an exciting member among all web sites. A typical example is Facebook [6], which allows millions of users to create online profiles and share personal information with vast networks of friends. A step further, a growing number of SNSs, including Facebook, also encourage end users to develop web plug-ins, and shares these plug-ins across the whole site. For example, in Xiaonei [7], Friend Trade is a web plug-in developed by five college students, which has been installed by more than three millions of Chinese users in 42 days.

Web plug-in attaches tremendous end users since they result in extremely unique user experience, and promote user's networking circles. However, they also introduce many new problems to both networking and software engineering fields. One key problem is pre-release failure. In general, software failure refers to an observable error in the program behavior [1]. Pre-release failures occur before software release, typically during software testing, while, post-release failures are found by users after the software is release.

Q. Wang et al. (Eds.): ICSP 2009, LNCS 5543, pp. 236–245, 2009.

Unlike software engineers in companies, web plug-in developers are free to release their plug-ins at any time without required to pass formal testing. As a result, most of the failures we found are post-release failures, instead of pre-release ones. This in turn leads to a vast number of performance, security, and privacy problems. For example, a careless developer may release a plug-in that fetches Bill's user profiles in an endless loop. As users install and run this plug-in, web server would significantly slow down, or even leads to DDoS. In another example, a controversial plug-in may collect e-mail addresses from a user's profile to forward spam.

To ensure the quality of plug-in, both industry and research communities have presented various methods. Facebook outlines a list of prohibited plug-in categories and seek for law protection [6]. John introduced ISN technique that only allows a user's friends to view his privacy information [8]. Meanwhile, Leon proposed a complex access-matrix, as well as a long list of checking rules [9]. Along another thread, Ohlsson et al. suggests CVS-based source code verification, and presented an automatic method [10]. While Ostrand et al. have been trying to introducing a formal quality assurance methodology customized for end user development [11].

However, all these methods fail to avoid the pre-release failures of plug-ins, or mix pre-release failures with post-release ones. In this paper, we present a release-waiting farm method, which extends the state of the art in three points. First, it highly ensures the freedom and creativity of plug-in developers. Second, it allows a plug-in to release only after systematic testing in SNS snapshot environment. Third, it formalizes the web plug-in development process, thus provides benefit to both SNS and developers.

The remainder of this paper is organized as follows. Section 2 summaries the related work on pre-releasing failure detection, as well as end user development. Section 3 defines the release-waiting farm, and then Section 4 presents the details design and our system realization. Finally, Section 5 discusses our findings with the analysis from a three-month experiment, and Section 6 concludes the paper.

2 Related Works

In this paper, we use the term *defect* to refer to an error in the source code, and the term *failure* to refer to an observable error in the program behavior. In other words, every failure can be traced back to some *defects*, but a *defect* need not result in a failure [2]. *Pre-release failures* generally occur before a software release, in the course of testing; *post-release failures* are found by users after the release. Therefore, pre-release failures, if undetected in time, will definitely become post-release failures.

To avoid post-release failures, a number of software metrics and techniques have been proposed [11]. In particular, Chidamber and Kemerer suggested eight metrics to cover the properties of object-oriented programs; Mockus used them in an industrial software development process to explore what driving software quality [8]. After that, Nachiappan [3] were among the first to validate these metrics, and found that these metrics appeared to be useful for predicting defect density. But these metrics and methods all assume that formal test phase is the key step to discover pre-release failures, and can't be as casual as the one in SNS.

Participation in SNS has dramatically increased in recent years, and thus has attracted the attention of the media and researchers. The latter have often built upon the existing literature on social network theory like [11] to discuss its online incarnations, or focus on profile to explore privacy and security problems [9], or collect data from SNS site like Flickr to study the phenomenon of social networks [4], or even predict the future trends through complicated survey [5].

In very recent years, as more and more SNSs enable users to develop web plug-ins themselves, and share the plug-ins inside the whole web site. Moreover, industrial companies have also paid much attention to this, e.g., Borland ported its applications to Facebook [6]. Starting from the perspective of end user development (EUD), and experimental system on exploring the test methodology on SNS, we proposed a release-waiting farm technology [14]. With release-waiting farm, SNS users are able to join formal, while customized for SNS, test activities, figuring out pre-release failures, and thus improve the quality of the web plug-ins they develop.

3 Release-Waiting Farm

The process of developing a web plug-in on SNS is significantly different from traditional software process. As shown in Fig. 1, the key difference of web plug-in development process lies in testing phase.

Unlike industrial software development, end user tends to test his plug-in by using it himself, but find no way to systematically test the plug-in. After using it himself, end user quickly believes that the plug-in can work. What he can do next is to release it, and wait for bug feedbacks from his friends. If there is bug feedback, in most cases there must be, he then iterates through the development process, re-enter requirement analysis, or design, or construction to fix a bug. SNS simply fails to provide effective support to end user developers, thus suffers from many failure-prone plug-ins.

To reduce the number of pre-release failures, we provide release-waiting farm technology, which is built on a modified iterative development process (Fig. 1). Here, after testing the plug-in himself, developer passes his newly constructed plug-in to a release-waiting farm. The farm is a small but compact web site provided by SNS, and supports four kinds of functions.

First, the farm is a mini world. The world not only consists of a medium-size, cohesive set of users that are taken from the latest snapshot of the database of SNS, but also owns the latest set of stable plug-ins that have already become popular in the SNS. Therefore, plug-ins that fully tested in the farm are safe to release.

Second, brainstorm usage and testing. SNS strongly encourages all end user developers to test new plug-ins. The scenario seems like many unknown users sit down around each plug-in, using it and trying to figure out the possible failures with their experience.

Third, invited senior users and developers. When a new plug-in enters the farm, it is tagged with a category. Generally, the senior users or developers who have extensive usage of that category of plug-ins can serve as field experts and perform valuable testing. Like brainstorm testing, SNS has established attaching bonus to compensate the time of the senior.

Finally, SNS needs debugging toolset. For example, SNS must provide API watcher for developers to view the formal output of its API. And it should run online bug-tracing system to record a bug opened by others, and fixed by corresponding developers. Moreover, it also needs to provide friendly functions for users to view and manipulate failure-related information like image, short description, and bonus.

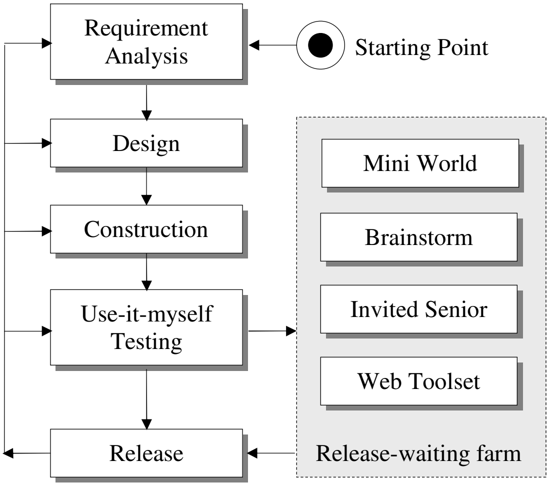


Fig. 1. A modified iterative development process for high quality web plug-in applications

4 Experiment

To verify the usability of release-waiting farm, we carried out an experiment in our software engineering course from Feb. 15, 2008 to Jul. 10, 2008. First, we constructed a web plug-in develop environment, including REST-like Java APIs, user manuals, sample code, and online bug-tracing system. Second, we set up two web servers, one for SNS, and the other for release-waiting farm. Third, we advertised our SNS exp.gucas.ac.cn (EXP for short) among the graduate students of Chinese Academy of Sciences. Until Jun. 12, 2008, EXP has attached 547 users, 36 active users are selected as senior ones, totally 73 plug-ins were released until Jul. 2008.

Like many other SNSs, EXP provides JSP develop environment, which is composed of three key parts: Java API, URL callback mechanism, and Apache Tomcat 5.5. As a popular web development technology, JSP enables users to develop various plug-ins with enormous support. In particular, our Java API borrows the same interfaces from Facebook.com, and is enough to build diverse plug-ins [7]. Note that the APIs have evolved several versions and many small improvements, so Table I lists only a partial snapshot.

Unlike other SNSs, we don't expect EXP to attach too many users since that requires much budget to support. Basically, EXP is an experimental SNS with simple UI, and lots of web plug-ins released by college students. In the course of experiment, the number of registered users raised up steadily. In particular, most users are Master candidates since EXP was mostly oriented from a graduate lesson. Finally, over 55% percent users are involved in developing at lest one plug-in.

As shown in Fig. 1, the plug-in development process in EXP is simple. End user first develops a beta plug-in on his local web server, and access the plug-in through URL callback mechanism. He tests the plug-in himself, and then submits it to release-waiting farm. After fixing the bugs submitted by other users in the farm, owner can then release the plug-in to main EXP server.

To enable release-waiting farm, we set up another web server (rwf.exp.gucas.ac.cn), and run the following program at 3:00 PM everyday, as shown in Fig. 2. Here, we arbitrarily choose top 10% active users and top 50% plug-ins, so as to build a snapshot of EXP. Note that there would be better tradeoff than ours. Finding satisfied tradeoffs under various conditions is regarded as our future work. But now, this is enough for us to build a medium-size snapshot.

Table 1. A snapshot of EXP APIs, taken at Jun. 12, 2008

Class	Function	Description
auth	startAppSession	start an app after system checking
	appLogin	only register app can login
user	getInfo	get open info of a user with his ID
	isAppInstalled	check if the user install this app
friend	getAllFriendID	get all friends' ID of a user
	getOnlineFriends	get all online friends' ID of a user
	areFriends	are two users friends?
	getAppUsers	all users who install this app
invite	addOutsite	invite outside friends to SNS
	addBonus	give bonus to a user for invitation
notify	sendMsg	send notification to a user

Table 2. Registered users summarized at Jun. 12, 2008

Type	Sub-type	No.	Senior
CS-related major student	Master	210	12
	PhD	38	2
	Post-Doc	9	0
other major student	Master	80	8
	PhD	23	2
	Post-Doc	3	0
campus staff	Age 23-26	70	5
	Age 27-30	45	1
	Age 30-	23	1
outside campus friend	Age 23-26	18	3
	Age 27-30	22	1
	Age 30+	6	1
All		**547**	**36**

Table 3. Top 10 EXP plug-ins at Jun. 12, 2008

Rank	Plug-In	Description
1	Buy-Sell Friends	Buy a friend, and punish him
2	Pick Music to Friend	Pick a music for you
3	Vote	You like football, tennis, or…
4	Footmark	I have been to Shanghai…
5	Name-Matching	Our name match?
6	Touch You	Give you a kiss, embrace…
7	Give You a Gift	Give my friend a gift
8	Movie Review	I have watched this movie…
9	Friend Impression	Bill is a strong handsome boy
10	Daily Account	I pay 10$ for breakfast…

During this five-month experiment, we have been strongly impressed by the enthusiasm of end users. To open more effective functions for end user developers, our APIs have been pushed to evolve several versions, and hundreds of improvements. Finally, 42 plug-ins are released by student teams in the software engineering course; 25 plug-ins are released by individual students for fun; six plug-ins developed by staff and friends outside campus. As a result, our small SNS surprisingly attached more than 500 users; many of them even require us continue to run both the main server and farm server after experiment and course.

```
//stop receiving new pre-release plug-ins
stop_receive_newcomer();
//store existing plug-ins and bug info into database
store_beta_plugin();

// top 10% active users
// users who install top 50% plug-ins
users = top10pActiveUsers() &&  usersInstall50pApp()
loadInfo(users);
loadPlugin(users);

//since not all users will be in farm,
//delete bad connections from to other users
DelBadConnection(users);

load_beta_plugin();
start_receive_newcomer();
```

Fig. 2. Building the mini world of farm

5 Analysis

We analyze the data collected during the experiment, and try to answer four questions. Does the farm big enough to test plug-ins? Do brainstorm and invited senior methods work? Is it worthy to build the farm? Can plug-in developers and SNS benefit from release-waiting farm?

From Table 4, we notice that almost all failures found in the farm can be revisited in main server if we don’t have the farm, i.e. restore the server status to the point when plug-in enters the farm. It is encouraging that for the top 10 plug-ins, 752 out of 772 failures found in the farm re-appear in the main server. So it is safe to claim that release-waiting farm technology can significantly uncover the pre-release failures.

However, there is still 20 failures we found in farm don’t reappear in the main server. Failures found in farm, but can’t appear in main server, simply means wasted effort. To explore the reason, we check the log file of plug-in Vote, and find that timing is the critical factor. Since a vote opens and ends in a fixed interval, typically several days. But in the farm, testing users simply want to finish the testing in seconds. This result in a significant difference usage scenarios, thus leads to 10 failures not found in main server. This indicates that the farm needs more effort to ensure that every failure found here, must appear in main server. We regard this as one of our future work.

What we do not show in Table 4 is the number of failures found in main server, but not in the farm. We can't do this since SNS can only encourage, instead of forcing a user to submit bug report.

A step further, it is satisfying that most failures found and fixed in the farm, won't appear in the main server. Of course, solving failures found in the farm but not in main server will delay the release time. We perform a online questionnaire to check this, and find that developers agree to test their plug-ins in the farm, and deem it worthy since it indeed improves the quality of plug-in. We also find from the questionnaire that, our farm is big enough to uncover enough pre-release failures due to poor design and programming skills.

Table 4. The failure analysis of top 10 plug-ins

Plug-In	Found in farm	Found in main server	Not found in main Server
Buy-Sell Friends	122	120	2
Pick Music to Friend	104	100	4
Vote	133	123	10
Footmark	67	67	0
Name-Matching	45	45	0
Touch You	44	44	0
Give You a Gift	59	56	3
Movie Review	65	65	0
Friend Impression	77	76	1
Daily Account	56	56	0
All	**772**	**752**	**20**

In the experiment, we also evaluate the testing methods, brainstorm and invited senior, through the analysis of the bug records in detail. Here, we regard a failure as effective failure (EF) only if it results in code change. Therefore, the number of EFs must be smaller than that of all open bugs. As reported in Table 5, it is easy to note that both brainstorm and invited senior methods give a solid number of failures, which result in code change.

Table 5. Bug status of top 10 plug-ins

Plug-in	Brainstorm		Invited Senior	
	Open	EF	Open	EF
Buy-Sell Friends	190	50	97	72
Pick Music to Friend	145	55	99	49
Vote	233	65	131	67
Footmark	80	17	69	50
Name-Matching	90	35	41	20
Touch You	66	18	37	26
Give You a Gift	98	35	31	24
Movie Review	79	34	30	21
Friend Impression	104	22	87	55
Daily Account	120	30	29	26
All	**1205**	**308**	**651**	**410**

Brainstorm and invited senior methods are different in that failures opened by the latter, are more likely to trigger code changes. As a result, data collected in bug-tracing system reveal that developer tends to solve the failure found by invited senior first. This tendency is reasonable since most failures (or bugs) opened by brainstorm are redundancy. In brainstorm, each tester opens a bug when he finds a failure, without reviewing the existing bug list. Finally, it is also safe to summarize that both brainstorm and invited senior methods work effectively.

To apply the release-waiting farm, we need to run an additional server, which introduces extra cost. So it is natural to ask "Is it worthy to use farm?" As shown in Table 6, the cost of running a stable and fast main server, is typically 3 times of running its farm. From hardware requirement, monthly bandwidth cost, to the needed number of maintenance staff, main server all present a tight demand on cost. While the farm costs 25% of the main server since it operates on a small-scale size of data set. We also claim that the percentage of farm cost tends to lower down as the size of main server grows.

Table 6. Comparison of main server and the farm

Parameter	Main Server	Farm
CPU	Intel Core 2 E4600	Intel P4 2.4 GHz
Memory	2G, 800MHz	512M, 400MHz
Hard Disk	SCSI 72G	SCSI 8G
NIC	Intel 8492 1000Mbps	Intel 8391 100Mbps
Peak Bandwidth	58.9Mbps	22.3Mbps
Network Cost	$150/m	$30/m
Maintainer Needed	3	1

A step further, we found the extra cost is strongly tight to the number of users loaded from main server. Note that in our experiment, the users automatically selected from the main server by our program are mostly come from the software engineering course. As homework, they are required to login the farm and test plug-in. As a result, commercial SNS must adapt the user-selecting method to make sure that users in the farm also like to contribute.

The source code of 30 teams in our class is managed through a CVS repository. Each time when a team releases a plug-in, they are required to add a version tag to the source. And code changes due to bugs found in the farm are finished by many small commits. We randomly selected five teams, and analyzed their CVS records with the technique presented in [8].

Fig. 3 shows the modification requests of these five teams in CVS. Note that before starting the course, 83% students haven't version control experience showed by a named questionnaire. After the course, we conduct another questionnaire, as well as coffee interview, and found that all involved students have already tend to use CVS-like systems to control their code, and like to release their software after other users' test. This shows that, although, more than half of the users participate in the development of plug-ins come from the non-computer professional[14], programming and submitting plug-ins is purely for fun, but for the pursuit of better results, they can accept the software engineering disciplines, and benefit from it. Or we can say that our Release Waiting Farm technology can help users to study and master the software

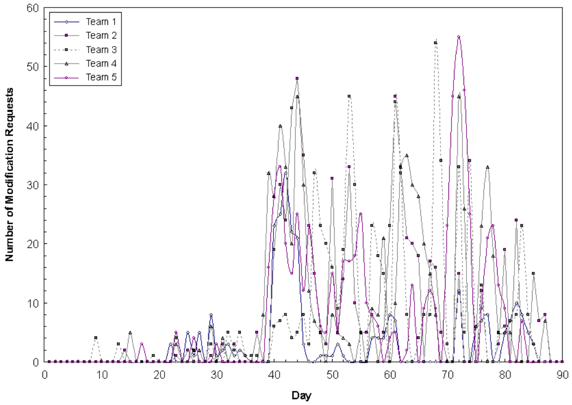


Fig. 3. Modification request of teams

engineering methods, standardize the development process of end-user, thereby enhancing the end-user developed software quality.

6 Conclusions

As a surprisingly growing number of web plug-ins have been installed by users in social networking site, many software failures, especially the pre-releasing ones, lead to significant performance, security, privacy problems, as well as research directions.

On the journey of reducing the pre-release failures, we present a release-waiting farm method, which extends the state of the art in three points. Our experiment reveals that the farm is big enough to test plug-ins, brainstorm and invited senior methods work effectively, the farm costs low when compared with the main server, and it is worthy to build a farm. We also found from both developers and general users that, SNS armed with this method can formalize the development process of end user, raise plug-in quality, and thus gain long-term benefit.

Acknowledgments

This work was supported by the National Natural Science Foundation of China under Grant No. 60773028; and the National Hi-Tech Research and Development 863 Program of China under Grant No. 2006AA01Z180.

References

1. Humphrey, W.S.: The Personal Software Process. Technical Report, CMU/SEI-2000-TR-022 (2000)
2. Li, P.L., Herbsleb, J., Shaw, M.: Finding Predictors of Field Failure for Open Source Software Systems in Commonly Available Data Sources: A Case Study of OpenBSD. In: Proceedings of the 11th IEEE International Software Metrics Symposium (METRICS 2005), pp. 32–52 (2005)

3. Nagappan, N., Ball, T., Zeller, A.: Mining metrics to predict component failures. In: Proceedings of the 28th international conference on Software engineering, pp. 452–461 (2006)
4. Subramanyam, R., Krishnan, M.S.: Empirical analysis of ck metrics for object-oriented design complexity: Implications for software failure. IEEE Trans. Software Eng. 29(4), 297–310 (2003)
5. Zimmermann, T., WeiBgerber, P.: Preprocessing CVS data for fine-grained analysis. In: Proceedings of International Workshop on Mining Software Repositories, pp. 2–6 (2004)
6. Facebook Inc., `http://www.facebook.com`
7. Xiaonei Inc., `http://www.xiaonei.com`
8. Mockus, A., Zhang, P., Li, P.: Drivers for customer perceived software quality. In: Proceedings of International Conference on Software Engineering (ICSE), St. Louis, MO, pp. 225–233 (2005)
9. Nagappan, N., Ball, T.: Use of Relative Code Churn Measures to Predict System Defect Density. In: Proceedings of International Conference on Software Engineering (ICSE), St. Louis, MO, pp. 284–292 (2005)
10. Ohlsson, N., Alberg, H.: Predicting fault-prone software modules in telephone switches. IEEE Transactions in Software Engineering 22(12), 886–894 (1996)
11. Ostrand, T., Weyuker, E., Bell, R.M.: Predicting the location and number of faults in large software systems. IEEE Transactions in Software Engineering 31(4), 340–355 (2005)
12. Sliwerski, J., Zimmermann, T., Zeller, A.: When Do Changes Induce Fixes? In: Proceedings of Mining Software Repositories (MSR) Workshop (2005)
13. Subramanyam, R., Krishnan, M.S.: Empirical Analysis of CK Metrics for Object-Oriented Design Complexity: Implications for Software Defects. IEEE Transactions on Software Engineering 29(4), 297–310 (2003)
14. Yu, X., Li, J., Zhong, H.: Release-waiting Farm: An Original Framework for Reducing the Pre-release Failures of Web Plug-in on Social Networking Site. In: Proceedings of CSIE 2009 (2009)
15. Zimmermann, T., Weigerber, P., Diehl, S., Zeller, A.: Mining Version Histories to Guide Software Changes. IEEE Transactions in Software Engineering 31(6), 429–445 (2005)

Technical Software Development Process in the XML Domain

Liming Zhu, Tu Tak Tran, Mark Staples, and Ross Jeffery

NICTA, Australian Technology Park, Eveleigh, NSW, Australia
School of Computer Science and Engineering, University of New South Wales, Australia
{Liming.Zhu,Tutak.Tran,Mark.Staples,Ross.Jeffery}@nicta.com.au

Abstract. Background: A Technical Development Process (TDP) is a development process for a particular technology, such as XML, service orientation, object orientation or a programming language. Unlike software development life-cycle processes, TDPs provide concrete and detailed guidance to software engineers working in a particular technology domain. TDPs are currently not well understood in terms of description, modelling and interactions with life-cycle processes. **Aim**: In this paper, we investigate what are TDPs in the XML domain and how can TDPs be modelled using existing development process modelling notations and tools. **Method**: We extracted XML specific TDPs from literatures, interviews and internal documentation within software development organizations and conducted systematic verifications and validations. **Results:** We identify different types of TDPs in the XML domain and propose mechanisms to model TDPs using Software Process Engineering Meta-models (SPEM) in the Eclipse Modelling Framework (EPF). **Conclusion:** The results demonstrate the feasibility of explicitly identifying and modelling of TDPs in the context of software process modelling and how they are used in software development. The results help further bridge the gap between macro-processes (life-cycle and management-centred processes) and micro-processes (e.g. developer-centred TDPs).

1 Introduction

A technical development process is a development process for a particular technology, such as XML, service orientation, object orientation or a programming language. Technical development processes are micro-processes [1] composed of technical steps, best practices, and checklists for different types of technology-specific components at different stages[2, 3]. These processes aim to provide concrete and detailed guidance to software engineers during development. This is in contrast to macro-level software development lifecycles such as waterfall, RUP and the V-model which provide high-level guidance for general software development. Micro-processes are important in two aspects:

- They help understand the causal relationships better in software development through looking into precise specification of the detailed processes.
- They are more amenable to software engineers who do not find macro-processes useful for their immediate needs at a micro-level during day to day development.

Q. Wang et al. (Eds.): ICSP 2009, LNCS 5543, pp. 246–255, 2009.

There are calls from both industry [4]and academia [1, 2, 5] for better understanding of technical development processes. This paper answers two research questions:

- RQ1: What are the technical development processes within an XML development context?
- RQ2: How can XML technical development processes be represented in the Eclipse Process Framework (a SPEM implementation)?

We explored different sources and discovered a range of technical processes within the XML domain. We identified different types of technical development processes and verified them with developers. We further validated the technical development processes against different intended uses including development, training and management. These technical development processes are initially captured as textual process descriptions. We then investigated representational approaches of technical development processes. The Software Process Engineering Meta-Model (SPEM) [6]and the Eclipse Process Framework (EPF)[7] were used to model and store these technical development processes. We propose several mechanisms of process modelling. The contribution of work is as follows:

- It helps better understand what technical development processes are from the view point of practitioners and process engineering. This enables further systematic use of technical development processes for planning, cost estimation training and management in addition to development.
- It proposes representational mechanisms for modelling technical development processes using existing process meta-models and modelling tools. This enables technical development processes to be used and analysed along with other types of process models including life cycle models (e.g. RUP or Spiral models) and practice driven models (e.g. Agile methods).

In the rest of the paper, section 2 discusses the background and existing work. Section 3 illustrates the methodology and data collection. Section 4 presents the results. Section 5 provides a discussion of the results, and lastly, section 6 concludes the paper.

2 Related Work

Osterweil [1] considers macro-process research to be an examination of external behaviours of process, researching topics such as speed of execution, characteristics of produced software products and the processes themselves. These are often carried out through empirical studies as we have limited understanding of the internals. There is a need to further investigate the internals of the macro-process activities (micro-processes) and better understand the causal relationships between micro-processes and other entities, such as life cycle models, product quality, cost estimation and project management. Current connections between macro-processes and micro-processes are usually created through organization- or project-specific process tailoring [8-11]. The factors investigated are limited to project context rather than technology-driven fire-grained processes. Micro-process research should be an examination of the "precise specification of the details of software processes, for the purpose of inferring how those details affect the external behaviours of the processes" [1]. One way of shifting from macro-process to micro-process research is to move from statistics-based investigations to ones based on process models.

Life-cycle driven process models are fairly abstract and can be represented in many different process modelling notations. They are more suitable for high level generic processes that apply to software development in general. Technical development processes lack this generality and straight-forward sequential flows (with only coarse-grained iterations), a life-cycle model style representation may not be appropriate. In this research, we adapt some of the ideas in life-cycle model representations. For example, the iterative representation of the spiral model may be applicable to modelling repeated refinement of XML schemas.

Various meta-models exist for systematic model representations. The meta-models generally fits into a 3 layered view of process engineering [12, 13]. Examples of meta-modelling languages include Software Process Engineering Metamodel (SPEM) , OPEN Process Framework (OPF), OOSPICE, LiveNet and the Standard Meta-model for Software Development Methodologies (SMSDM) [12]. We choose SPEM in this research because its relative better tool support through Eclipse Process Framework (EPF), which enables the possibility of interaction analysis with other types of processes through existing EPF method plug-ins (e.g. Open RUP, XP and Scrum). EPF implements the SPEM model.

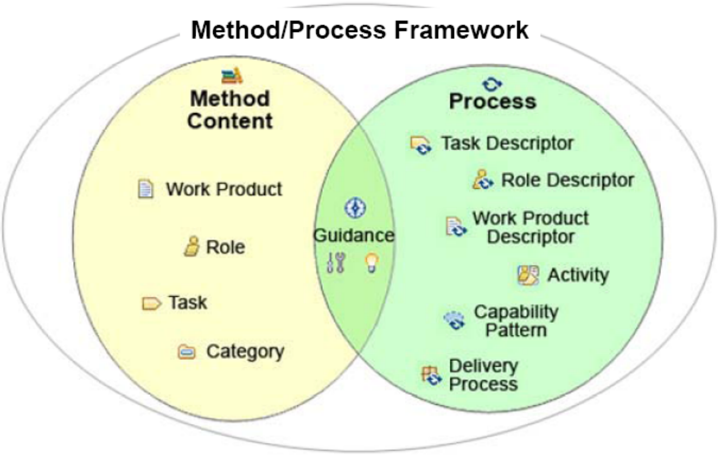


Fig. 1. Structure of EPF

As shown in Figure 1, method content refers to the content in processes, including tasks, roles, work products and guidance. The process section of the framework adds a temporal element, allowing the creation of ordered sequences of methods or processes. There are two main types of process in EPF: capability patterns and delivery processes. Capability patterns represent short term reusable clusters of processes, while delivery processes represent to end lifecycle processes.

An electronic process guides (EPG) is another example of using process models for day to day development. EPGs are "web applications structured according to the process with process descriptions, navigation and searching tools and electronic links to extra information like templates, examples, tools and project databases" [14] [15]. They allow process models to be tailored and instantiated into project models, whereby software improvement techniques can be applied to improve the process model layer. However, the process models used, including project-specific models, are still high-level models that do not take technology-driven practices into more systematic process consideration.

Technical development processes have been identified as one of the main internal factors affecting micro-processes [2]. In this paper, we investigate what technical development processes are in the XML domain and how to represent them in process models. The reason for choosing the XML domain is due to the observation that certain technologies (such as XML) have bigger impact on development processes at the micro-level [2].

3 Methodologies and Data Collection

The organisation involved is a small to medium sized software development company focused on building and integrating publishing systems which create, manage and disseminate information. They also conduct training courses in using XML technologies such as XSD, XSLT and XPath. The organisation was selected due to their extensive experience in the XML domain. XML technical development processes were gathered from the organization using semi-structured interviews and mining internal documents, described in detail below. From these sources a list of candidate XML TDPs were compiled and briefly described. Fifteen candidate XML technical development processes were found. We conducted semi-structured interviews, mined internal documentation and cross-verified with other sources.

Semi-structured interview

Senior developers were selected to do an initial semi-structured open-ended interview. They were chosen because of their extensive experience in XML development. This questionnaire was piloted with a research engineer to estimate the time required to complete the questionnaire, and find any initial problems with the questions. On average, the interview with developers took about two and half hours. Notes were taken during the interview, and interviews were recorded and transcribed for review. Upon completion of the initial interview, seven candidate XML technical development processes were found. A second-round interviews were conducted with the developers to gather further details to clarify details for some of the other candidate processes. More technical development processes were identified.

Internal documentation

Internal documents within the company were also used to extract XML technical development processes. Internal documents include email conversations and internal development documents. The authors had unrestricted access to the intranet for general information and project related information.

Other sources

Other sources include a number of XML development books, online resources and developer blogs. They were mainly used to cross-verify the identified processes.

A similar approach was taken the representation (modelling) which was used to confirm that SPEM-based EPF representations are sufficiently accurate. Verification also ascertained whether any improvements could be made if there were deficiencies in the mechanisms of the representation.

We also conducted validation against two purposes in addition to development purposes. The purposes used in the validation include:

- Educational benefits: whether the model is helpful for learning XML development processes
- Usefulness to management: whether the model is useful in supporting managerial tasks

4 Results

4.1 Documenting Technical Development Processes

Table 1 provides an overview of the candidate XML technical development processes extracted from the company.

There were several general ways of mapping technical development processes to an EPF repository:

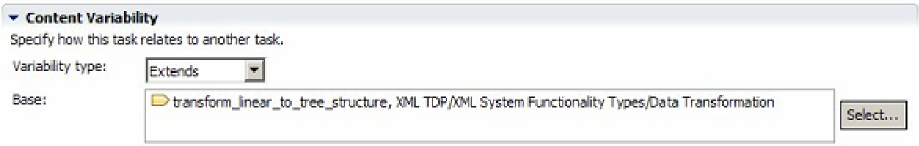
- Task extend
- Task contribute
- Extra tasks
- Disciplines and capability patterns
- Extension of life-cycle or practice driven development processes

Table 1. Example Technical Development Processes

ID	Category	Short description
Use a standard schema	Schema design	When using a standard schema, no schema design is required but additional activities have to be performed.
Use a three-tiered transformation architecture	Data transformation	Build an additional layer between input and output to buffer changes that occur with the input and output structures
Make a choice of data validation techniques	Technology selection	Grammar-based or rule based validation should be selected
Merge different data structures	Data transformation	Combine and reconcile two or more different XML schema structures
Transform a linear structure to a tree structure	Data transformation	Develop a data transformation engine that transforms a linear structure XML (flat hierarchy) to a tree structure (deep hierarchy)

Using task extend

In some technical processes, the generic process was represented as a task, while the more specific section was added using “task extend” as shown in Figure 2.



Content Variability

Specify how this task relates to another task.

Variability type: Extends

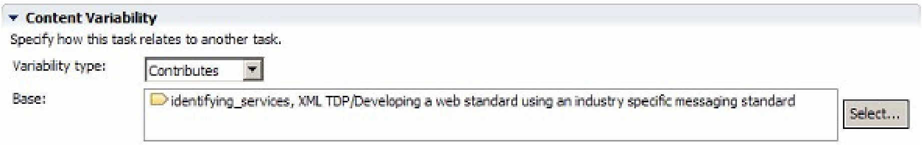
Base: transform_linear_to_tree_structure, XML TDP/XML System Functionality Types/Data Transformation

Fig. 2. Task Extend

Extended tasks inherit all the sections of their base classes except for the sections that are filled in the extension task. In which case, the extended task section completely replaces the base task’s section. This can be seen in where the purpose is the only section that gets replaced in the project specific task. However, a disadvantage of this is that if there are parts of the generic process that you want to use in the descriptions of the project specific task, there needs to be manual duplication of the content.

Using task contribute

In “task contribute”, the contributing task appends to the content as shown in Figure 3. Also both contributing and contributed-to tasks become one task in the final composition. Hence, any links that refer to the generic task or the specific task refer to the same task, with the base content and contributed content appended. This is a disadvantage if there is a need to show generic and project specific tasks separately.



Content Variability

Specify how this task relates to another task.

Variability type: Contributes

Base: identifying_services, XML TDP/Developing a web standard using an industry specific messaging standard

Fig. 3. Task Contribute

Extra tasks

Adding extra tasks is another technique used to represent more specific processes. The process is represented as a separate task because there was no real generic task that could be extended or contributed to, to represent it. However, if there was a generic task that was relevant, creating an extra task would entail needless duplication, where task extensions or contributions could otherwise be used.

Disciplines and capability patterns

Using a discipline and a capability pattern was another way to model generic and project specific technical processes. This was applied to some processes, as an alternative to the “task contribute” approach. Several disciplines were created whereby the generic tasks were added along with the extra tasks with a capability pattern as a reference workflow. Within the capability patterns, generic tasks were added, and extended using the process properties documentation. Using the process documentation, the method content task could be replaced with new information. Replacing the information from the process documentation was exactly like extending a task, in that if a section was empty, the base task content would be used. There were instances that replacing the section was appropriate, for example the key considerations in a data validation process needed to be replaced completely, as the list of potential considerations of the generic task became irrelevant in some cases.

Linking back to the generic process was also used for more contextual information about the key considerations. An advantage of having a capability pattern for the project specific processes was that it gave an indication of the flow of tasks, and allow for an activity diagram to be created. A disadvantage of extending the task using process documentation is that it is not reusable. Only method content can be reused within EPF.

Extension of life cycle or practice driven development processes

The extension of development processes is very similar to representing generic and specific processes. In our case, we extended OpenUP (a tailored version of RUP) for the XML domain in some instances.

4.2 Representing Technical Development Processes

Representing technical development processes using workflow-based process models poses challenges as workflow-based approach limits the way of modelling. These limitations have been identified [16]and the alternative is to use a more flexible and advanced process modelling language, such as Little-JIL [17]. However, as one of the primary goals of this research is to model technical development processes along with life-cycle and practice driven development processes, we chose EPF and SPEM as the platform and notation. The authors are currently exploring the features of Little-JIL for modelling.

There were several general ways of modelling technical development processes in EPF as shown in Figure 4 and Figure 5:

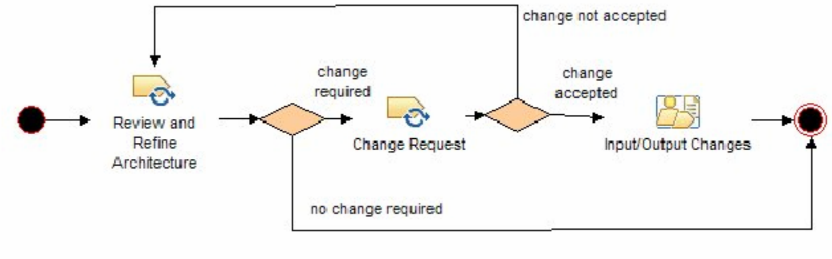


Fig. 4. Use a three-tired transformation architecture

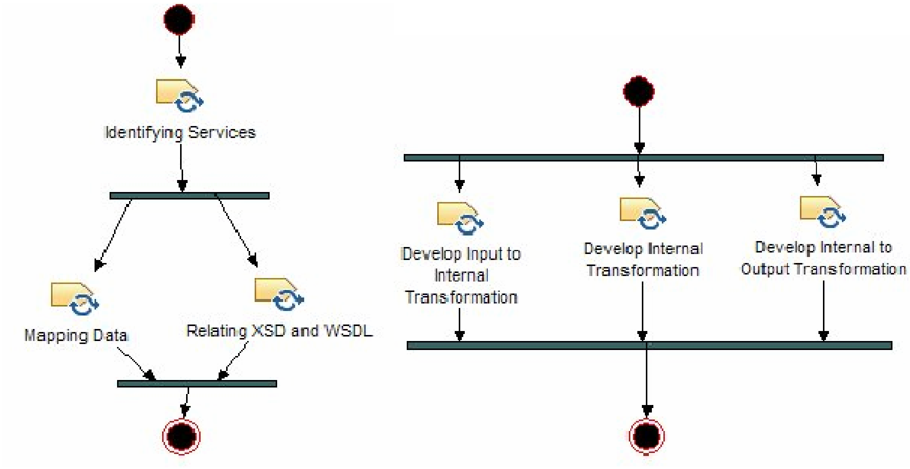

Identifying Services
Mapping Data
Relating XSD and WSDL
Develop Input to Internal Transformation
Develop Internal Transformation
Develop Internal to Output Transformation

Fig. 5. Develop Web services using industry schema standards

Iterative tasks

Iterative tasks or tasks that constantly repeat were represented in an activity diagram with synchronisation bars. This allowed the tasks represented in parallel to be shown to repeat amongst each other, without having to specify the order of the repeats.

Using steps to represent various outcomes

Steps were not limited to just representing a sequence of events. We used steps to indicate the various outcomes of a technical process. The list form feature in EPF was useful as it allows each outcome to be separated without having to create a separate task. The list of steps is also extended using “task contribute” when there was new techniques involved.

Representing technical examples

Technical examples (e.g. how to use Schematron in an Ant task) were represented using a tool mentor and a template. The tool mentor presents a starting point in how to use a particular process. Rather than copying the documentation provided into the contents of the template, the documentation file is able to be included in EPF and linked to from the template file. Both the tool mentor and the template are forms of guidance in EPF, which can be reused by other tasks and processes.

5 Discussion

Project specific processes

During the interview, some practices are project specific and can not be easily generalized. These are documented as project practices with their specific contexts. Generic processes provide a baseline for comparison amongst all the XML technical development processes. Also, the project specific aspects were helpful in the interviews as they provided a context for the development process and assisted the developer in remembering the intricacies of the process. In terms of using the process, project specific representations also help by providing examples of how the process can be applied.

Technology/Technique selection

We found technology/technique selection is an important aspect of XML technical development processes. Most of these include information such as a range of technologies to choose from, situations in which the technology would be used, and advantages and disadvantages in their use and the associated technical processes and activities. The chosen techniques can drastically influence the software development processes that occur. For example, among data validation techniques, choosing a rule based schema and validation means completely different processes in the requirements elicitation stage to acquire business rules that could be represented in the schema.

Validity of the results

One external validity threat of this research is the number of developers being interviewed and the fact that one company is the primary source. This is remedied through cross validating cross other sources including external sources.

These technical development processes are also contributed back to the company in the form of EFP repositories. They will be applied in more projects to validate and improve the existing processes.

6 Conclusion

In this research, we extracted technical development processes for the XML domain from a development organisation and cross-verified from other external sources. These technical development processes are documented in SPEM using EPF. We propose a number of mechanisms for documenting such processes both in textual formats and workflow based representations. These representations were further validated against their use through interviews.

The research further bridges the gap between macro-processes and micro-processes through better understanding of what technical development processes are from the view point of practitioners and process engineering. This may allow further systematic use of technical development processes for planning, cost estimation training and management in addition to development. The representational mechanisms for modelling technical development using SPEM and EPF also enables technical development processes to be used and analysed along with other types of process models including life cycle models (e.g. RUP or Spiral models) and practice driven models (e.g. Agile methods).

We are currently analysing process interactions among different type of processes and exploring the possibilities of using a more advanced process modelling language for technical processes.

Acknowledgements

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

References

1. Osterweil, L.J.: Unifying Microprocess and Macroprocess Research. In: Li, M., Boehm, B., Osterweil, L.J. (eds.) SPW 2005. LNCS, vol. 3840, pp. 68–74. Springer, Heidelberg (2006)
2. Zhu, L., Jeffery, R., Huo, M., Tran, T.T.: Effects of Architecture and Technical Development Process on Micro-Process. In: Wang, Q., Pfahl, D., Raffo, D.M. (eds.) ICSP 2007. LNCS, vol. 4470, pp. 49–60. Springer, Heidelberg (2007)
3. Zhu, L., Staples, M., Jeffery, R.: Scaling Up Software Architecture Evaluation Processes. In: Wang, Q., Pfahl, D., Raffo, D.M. (eds.) ICSP 2008. LNCS, vol. 5007, pp. 112–122. Springer, Heidelberg (2008)
4. Waldt, D.: The extensibility manifesto: A blueprint for XML implementation. In: XML (2005)
5. Bhuta, J., Boehm, B., Meyers, S.: Process Elements: Components of Software Process Architectures (2005)
6. OMG: Software Process Engineering Metamodel (SPEM) v2.0 Draft (2005)
7. Eclipse Process Framework (EPF), `http://www.eclipse.org/epf/`
8. Münch, J.: Transformation-based Creation of Custom-tailored Software Process Models. In: International Workshop on Software Process Simulation and Modeling (ProSim), pp. 50–56. Institution of Electrical Engineers (IEE) (2004)
9. Johansson, E., Nedstam, J., Wartenberg, F., Host, M.: A Qualitative Methodology for Tailoring SPE Activities in Embedded Platform Development. In: Bomarius, F., Komi-Sirviö, S. (eds.) PROFES 2005. LNCS, vol. 3547, pp. 39–53. Springer, Heidelberg (2005)
10. Jaufman, O., Munch, J.: Acquisition of a Project-Specific Process. In: Bomarius, F., Komi-Sirviö, S. (eds.) PROFES 2005. LNCS, vol. 3547, pp. 328–342. Springer, Heidelberg (2005)
11. Hanssen, G.K., Westerheim, H., Bjornson, F.O.: Tailoring RUP to a Defined Project Type: A Case Study. In: Bomarius, F., Komi-Sirviö, S. (eds.) PROFES 2005. LNCS, vol. 3547, pp. 314–327. Springer, Heidelberg (2005)
12. Henderson-Sellers, B., Gonzalez-Perez, C.: A comparison of four process metamodels and the creation of a new generic standard. Information and Software Technology 47, 49–65 (2005)
13. Gonzalez-Perez, C., Henderson-Sellers, B.: Modelling Software Development Methodologies: A Conceptual Foundation. Journal of Systems and Software 18, 1778–1796 (2007)
14. Kurniawati, F., Jeffery, R.: The use and effects of an electronic process guide and experience repository: a longitudinal study. Information and Software Technology 48, 566–577 (2006)
15. Scott, L., Carvalho, L., Jeffery, R., Ambra, J., Becher-Kornstaedt, U.: Understanding the use of an electronic process guide. Information and Software Technology 44, 601–616 (2002)
16. Zhu, L., Osterweil, L., Staples, M., Kannengiesser, U., Simidchieva, B.I.: Desiderata for Languages to be Used in the Definition of Reference Business Processes. International Journal of Software and Informatics 1, 37–66 (2008)
17. Wise, A.: Little-JIL 1.5 Language Report. Department of Computer Science, University of Massachusetts, Amherst, MA (2006)

Software Product Quality: Ensuring a Common Goal

Sebastian Barney and Claes Wohlin

School of Engineering
Blekinge Institute of Technology
sebastian.barney@bth.se, claes.wohlin@bth.se

Abstract. Software qualities are in many cases tacit and hard to measure. Thus, there is a potential risk that they get lower priority than deadlines, cost and functionality. Yet software qualities impact customers, profits and even developer efficiency. This paper presents a method to evaluate the priority of software qualities in an industrial context. The method is applied in a case study, where the ISO 9126 model for software quality is combined with Theory-W to create a process for evaluating the alignment between success-critical stakeholder groups in the area of software product quality. The results of an exploratory case study using this tool is then presented and discussed. It is shown that the method provides valuable information about software qualities.

1 Introduction

Software quality forms an important part of a product offering. But what qualities are valuable is a very context dependant problem, changing with both the product and perspective you bring. Maximising the value of a product's quality involves reconciling any conflicts between the key stakeholder groups – including customer, business and technical perspectives – so that these groups can work together effectively towards a common goal. However, there is always a risk that qualities get a lower priority than delivery date, cost and functionality. This risk comes from the fact the qualities are most difficult to measure in relation to delivery date, cost and functionality. The balance between delivery time, cost, scope and quality is discussed as part of XP [1].

This obvious risk forms the starting point of the research presented in this paper. The main objective is to understand priorities of software qualities in an industrial context so that any improvements can be focused on the most important aspects of product development. It is also key that any improvements can be made in the context of the development environment. The paper makes two main contributions; first, a method for analysing qualities in an industrial organisation is presented; second, the method is applied and the paper presents an industrial study exploring the alignment priority given to various software qualities between the groups involved in the software development process.

Asking what defines an adequate level of quality in a software system is a highly context dependent question [2]. Software quality affects more than just the user of the software and each group involved with a software product brings its own perspective on quality [2]. Customers, developers, product managers, project managers and testers can all value the same qualities of the same product in different ways. Looking at the

Q. Wang et al. (Eds.): ICSP 2009, LNCS 5543, pp. 256–267, 2009.

software supporting a social networking site, one could reasonably expect that customers would value usability higher than the other groups, developers value maintainability due to the dynamic nature of the product and management values efficiency due to the scale and resources required of the application. But ultimately these value stances need to be reconciled.

The qualities each group values will also change depending on the product. Functionality is becoming increasingly important for mobile phones, reliability is more important in financial and medical domains, and portability for web-based applications.

Understanding both (a) the groups impacted by a software product and (b) the value provided to each group by the various software product qualities is useful information for companies developing software. As any development project will have time, resource and financial constraints, this information will allow the development effort to be focused in the most critical areas.

The remainder of the paper is outlined as follows. Section 2 presents some background in terms of related work, objectives and the context of the case study. In Section 3, the method for studying qualities with respect to priorities and management in an industrial context is presented. The results of the case study are presented in Section 4. In Section 5, the applicability of the method and the findings from the case study are discussed. Finally, Section 6 presents the conclusions.

2 Background

In the software development process there are four variables that need to be controlled – cost, time, quality and scope [1]. Further there is an axiom that states that external forces can set at most three of these variables with the remainder being set by the development team.

Time, cost and scope can all operate within acceptable ranges, but quality is a terrible control variable as it only allows very short term gains at a very high cost to all parties involved [1]. That said, quality does not need to be perfect [3] but the development process is much simpler when the success-critical stakeholders agree on what action should be taken [4].

This section examines different perspectives of quality, processes to manage quality in a software engineering context, and the research objectives.

2.1 What Is Quality?

The definitions of quality are both many and conflicting, even when only examining the topic in relation to software engineering. Looking across different disciplines it is possible to see a complex multifaceted concept of quality that can be described from five different perspectives [5]:

- The *transcendental perspective* defines quality as something that can be recognized but not defined in advance.
- The *user perspective* defines quality as fit for purpose.
- The *manufacturing perspective* defines quality as conformance to specification.
- The *product view* defines quality in terms of essential characteristics of the product in question.

- The *value-based view* defines quality in terms of the amount a customer is willing to pay for it.

By far the most common perspectives taken in the software development industry are that of the user and manufacturer. [2,6]. However, there is an increasing body of literature that recognises the importance of taking advantage of all of the perspectives involved in software development. Theory-W states that success requires all of the success-critical stakeholders to compromise [4], while requirement specification reading techniques that take advantage of different perspectives have been found to catch 35% more defects than non-directed alternatives [7,8], and value-based software engineering now recognises the value brought by different perspectives into the development process [2].

Software quality is not only defined by the relevant perspectives, but also by the context in which it exists [2]. Just as each line of cars has a target market, software quality must be planned to allow a development company to meet its business objectives. Less than perfect software quality can in fact be ideal [3], but deciding how much less than perfect can only be decided in a given business context [2].

2.2 Quality Models for Software Development

Numerous models have been developed to support software quality. Examples of these models include McCall's quality model, Boehm's quality model, Dromey's quality model and ISO 9126.

McCall's quality model is the first of the modern software product quality models [2]. The model uses a hierarchy of factors, criteria and metrics to address internal and external product quality. Eleven factors define an external or user perspective of quality. Each of these factors is linked to between two and five of 23 criteria that define an internal or development perspective of quality. Further metrics are associated with the factors allowing quality to be measured and managed.

McCall's quality model was followed by Boehm's quality model [2]. Like McCall's model, Boehm's model presents product quality in a hierarchy with three high level characteristics linked to seven intermediate factors, which are in turn linked to 15 primitive characteristics. Boehm's model has a wider scope than that of McCall's, with more emphasis on the cost-effectiveness of maintenance [9].

More recently work has been done to create an international standard for software product quality measurement – ISO 9126 [10]. This standard is again organised in a hierarchy with six characteristics at the top level and 20 sub-characteristics with indicators used to measure the sub-characteristics. In addition to aspects of internal and external quality, covered by McCall and Boehm's models, ISO 9126 includes quality characteristics of functionality [9]. Internal, external and functional qualities are also mixed at all levels of the hierarchy. However, ISO 9126 does not clearly state how quality should be measured [2].

None of these three models present a rationale for the selection of characteristics to be included in the quality model and it is not possible to tell if a model presents a complete or consistent definition of quality [2]. Further the placement of items appears arbitrary in ISO 9126, with no justification as to why Interoperability is not related to Portability.

Dromey presents a different type of model that attempts to address some of the issues presented and support developers build product quality [11]. Dromey believes that it is impossible to build high-level quality attributes like reliability or maintainability into a product, but developers must instead build properties that manifest in achieving these goals. The distinction this model makes is important, as using it will verify that it allows the quality required to be achieved [2]. Before Dromey's model can be successfully applied, the various groups involved in the development of a software product must agree on what quality attributes should be achieved and to what level. This process can be supported using other models.

2.3 Merging Perspectives on Software Quality

Software product quality can easily become an area of problems and conflict, as each stakeholder group has its own perspective on what is important. A number of methods can be applied to help reconcile this situation and select the best way forward. These methods include expert judgement, the NFR Framework, Quality Functional Deployment and Theory-W.

Expert judgement involves one or more experienced professionals using their experiences and knowledge to make a decision on an issue. The decisions are not necessarily supported by modelling or numerical assessment.

The NFR Framework uses diagrams to relate non-functional requirement goals with different decisions that can be made in the design and operation of a system that affect it positively or negatively, allowing trade-offs to be identified and made [12]. While this method makes the results of a choice to be made more explicit, it requires a set of common priorities to be identified to allow effective decisions to be made.

Quality function deployment (QFD) considers the priority of customer and technical requirements in achieving the goals of the system to help prioritize the requirements [13]. However, the other perspectives involved in the development of the software product are not considered.

Value-based software engineering (VBSE) recognises the problems created by conflicting perspectives in the software development process [14]. Central to resolving conflict in VBSE is Theory-W, which requires [4]:

1. Success-critical stakeholder groups to be identified;
2. The requirements of these groups to be elicited;
3. Negotiation between the groups to create a win-win situation; and
4. A control process to support success-critical stakeholder win-win realisation and adaption to a changing environment.

The key advantage of Theory-W is that it explicitly brings all of the parties on whom success lies together to understand each other's needs, compromise and agree. But in order to be successful Theory-W must be managed to ensure the plans are achieved and any deviations from the plans are corrected [4]. Management requires an understanding of why the goals are being pursued, what is the required result, who is responsible for the result, how the result will be achieved and at what cost the result can be achieved. The answer to these questions will be specific to the context in which they are answered.

2.4 Research Objectives

The objective of the research presented in this paper is to create and validate a method capable of determining the level of alignment between the internal success-critical stakeholder groups. The method should be able to identify the degree to which the groups are aligned in how they perceive operations today with respect to quality.

This method should be evaluated in an industrial case, answering the research questions presented in this section.

RQ1: Is the method proposed in this paper capable of identifying the degree to which the internal success-critical stakeholder groups are aligned in how they perceive the priorities on software product quality today?

However, alignment itself only ensures that the success-critical stakeholder groups have a common understanding of what is happening today, it does not mean the groups agree this is what should be happening today. As each group represents a different, and potentially conflicting, perspective on software product quality it is important to discover what each of these groups perceive should be happening the situation today, in a hypothesised *ideal* situation. This is addressed by the second research question:

RQ2: Is the method proposed in this paper capable of identifying what the different internal success-critical stakeholder groups perceive as the ideal set of priorities on software product qualities in the situation today? And to what degree are the groups aligned?

3 Methodology

To address the research questions, a method, using Theory-W as a starting point, is developed. By exploiting the early phases of Theory-W it is possible to determine the level of alignment between the internal success-critical stakeholder groups. This involves identifying the internal success-critical stakeholder groups and eliciting their value propositions with respect to quality.

The results should support the continued application of Theory-W, to negotiate between the success-critical stakeholders to achieve a better situation and realize this goal through clearer management.

3.1 Quality Model

The literature on software product quality recognizes that quality depends both on the perspective of the observer and the actual software product in question. As such, using any model as it appears in the literature risks not adequately defining quality in the context being studied. To use one of the quality models briefly introduced in Section 2 is a good starting, but company specific needs have to be taken into account, as illustrated in the case study in Section 4.

3.2 Questionnaire

This method proposes the cumulative voting (CV) [15] technique to elicit (a) how important each quality is today, and then repeated the exercise to show (b) how important they perceived each quality should be today in a perceived ideal situation. CV asks participants to spend 1000 points across all of the qualities previously identified, to represent their relative influence. For example, if a participant thought testability does not at all matter today and security was twice as important as scalability they might award these qualities zero, 200 and 100 respectively.

3.3 Analysis

CV allows participants' responses to be grouped logically for analysis – for this method into the success-critical stakeholder groups. The results of each participant in the group can be averaged for each quality, ultimately producing a list that shows each quality and the averaged notion of its importance.

From here it is possible to rank the qualities from most to least influential for each success-critical stakeholder group. The degree to which the groups are aligned can then be calculated pairwise using a Spearman rank correlation matrix.

4 Case Study

The case study was conducted during Autumn 2007 for two products at Ericsson. Ericsson is a world leading company in telecommunication, providing a wide range of products and solutions. Products are developed and sold as generic solutions offered to an open market, although customized versions of the products are also developed for key customers.

4.1 Success-Critical Stakeholder Groups

High-level R&D management supported the authors to identify internal success-critical stakeholder groups for this case study. Participants in the case study represent:

- *Strategic Product Management (SPM):* This group has the strategic product responsibility and decides the overall product development direction.
- *Project Management (PM):* This group is responsible for planning and executing projects aligned with the priorities of the strategic product management.
- *Tactical Product Management (TPM):* This group supports the strategic product management with expert knowledge of the systems and their architecture. It is also responsible for providing analysis of pre-project requirements in the form of feasibility, impact and technical dependencies.
- *Development and Testing (R&D):* These groups are responsible for the implementation, verification and validation of requirements.

The high-level management further recommended that the results of SPM and PM be combined when determining the priorities given to the software product qualities, the first research question. These groups work closely together; with SPM prioritising the development activities that PM is responsible for planning.

A description of this case study was sent out to the managers of the identified success-critical stakeholder groups requesting volunteers from their teams to take part in the case study.

In total 44 potential participants were identified to take part in this case study, with 31 usable results being obtained. A breakdown of the participants can be seen in Table 1. Two of the participants identified felt they were not appropriate and identified other people in their team to replace themselves, two people declined to participate, one questionnaire result was lost in an Excel crash and nine people could not find time to complete the questionnaire.

Table 1. Study response rate

Group	Candidates	Replacements	Complete Responses
Strategic Product Management	15	1	6
Project Management	6		4
Tactical Product Management	9	0	9
Development and Testing	14	1	12
Total	*44*	*2*	*31*

The low response rate for Strategic Product Managers was anticipated, so extra participants for this role were selected to ensure a sufficient number of responses.

The questionnaire was conducted as a one-on-one structured interview, which each participant taking between 30 and 75 minutes. The interviews were conducted over two-month period.

4.2 Software Product Qualities

The process of defining a model of software product qualities was a collaborative exercise involving the academic and industrial perspectives. The list of qualities was defined specifically for the products studied at Ericsson, maximizing the relevance for this industrial partner and possibilities for using the results to support improvements within the company.

However, just as some qualities can be more important than others, there are other aspects of the development process that compete with the implementation of software product quality. To understand the important of software product quality it must be placed in the context of all aspects of software product development that are controlled. These are *time*, *cost*, *quality* and *scope* [1].

These four control variables have been complemented with ISO 9126, the international standard for software product quality, providing more detail on the components of quality and scope. The authors then wrote preliminary definitions for these terms.

A workshop was held within Ericsson to review and refine the terms defining software product quality. The aim was to ensure the final list of terms and definitions would be complete, meaningful and useful to Ericsson. The model was split into three categories – the ISO 9126 qualities relating to *functionality*, the ISO 9126 qualities relating to *system properties* and *project management* to cover *time* and *cost*. Moreover, *security* was moved from *functionality* to *system properties*. Two new qualities were identified and added to *system properties*; these are *scalability* and *performance management/statistics*. Finally, five of the quality terms were complemented with alternative names used in Ericsson.

The terms and definitions used in the case study presented in this paper are available online [16].

4.3 Pilot Study

A questionnaire was developed using the methodology described in Section 3 and piloted. The participants in the pilot had trouble making comparisons between qualities related to *features*, *system properties* and aspects of *project management*. An example of such a comparison could include accuracy of features, resource behaviour of the system and development cost. In order to address this issue the authors modified the questionnaire to use a hierarchical cumulative voting (HCV) method as described by Berander and Jönsson [17]. This effectively split the questionnaire up into four independent CV exercises; one parent list that included the three category terms – *features*, *system properties* and *project management* – and one list for each of the categories, each containing the relevant qualities. A pilot of the new questionnaire found the participants' capacity to respond much improved.

In order to conduct the analysis each participant's response needs to be changed from HCV to CV, converting the four cumulating voting lists into one that covers all of the qualities.

Remember that one of the four lists includes the categories *features*, *system properties* and *project management*, while the remaining lists each detail the qualities that make up one of the categories. This allows the number of points awarded to each quality to be multiplied with the category from which it came [17]. It is also necessary to muliply each of these results by the number of qualities from the same category as the resultant value. Finally the set of number for each quality can be scaled so that the sum is 1000.

For example, if *200* points are awarded to *project management* and *600* points are awarded to *time*, then *category* × *quality* × *number of qualities* = *200×600×2* = *240,000*. The scaling of this result then depends on the other values, but if the other values were to sum to *4,800,000* then the number would be scaled to *result* ÷ *total sum* × *1000* = *240,000* ÷ *4,800,000* × *1000* = *50*.

It is necessary to multiply each value by the number of qualities in the same category to ensure that qualities with many categories are not underrepresented and that categories with few qualities are not overrepresented.

The individual responses can now be grouped and averaged, allowing ranks to be determined and the Spearman rank correlation can then be calculated.

The final version of the questionnaire is available online [16].

4.4 Software Product Quality Priorities

The first objective of this case study is to determine the degree to which the key stakeholders are aligned regarding how they see software product quality today. The results show the groups are very aligned, with Spearman's rank correlation values between 0.80 and 0.90 indicating each group ranked the qualities in a very similar order. The full results are presented in Table 2.

Similarly the key stakeholder groups are aligned how they ranked the software qualities should be today, in their perceived ideal situation. The results in Table 3 show correlation values between 0.65 and 0.74 between the groups.

Table 2. Correlation matrix showing the degree to which the groups are aligned in how they perceive the priorities today

	SPM & PM	TPM	R&D
SPM & PM	1.00	0.80	0.90
TPM		1.00	0.86
R&D			1.00

Table 3. Correlation matrix showing the degree to which the groups are aligned in how they perceive the priorities should be today (ideal)

	SPM & PM	TPM	R&D
SPM & PM	1.00	0.74	0.71
TPM		1.00	0.65
R&D			1.00

The similarities between the perceived situation today and the perceived ideal situation were striking. Looking at all responses the correlation between the two situations is 0.82. However, each group individually saw the need for more changes. The results in Table 4 show, for example, a correlation of 0.62 comparing what R&D perceived as the priorities today against what they thought the priorities should ideally be today.

Table 4. Correlation between perceived situation today and perceived ideal situation today

Group	Correlation
All groups	0.82
SPM & PM	0.72
TPM	0.77
R&D	0.62

While there was variation between participants of the same perspective, there was no individual that stood out as being consistently different in their results to the other members of their group.

Looking at the underlying data it is possible to further understand the differences and similarities between the groups. The remainder of this section highlights key aspects of similarity and difference.

The qualities studied have been grouped into three categories – *features*, *project management* and *system properties*. All of the groups today ranked these groupings in the same order, with *features* as the most important category, followed by *project management* and finally *system properties*. Interestingly all groups would like to see *system properties* overtake *project management* in their perception of the ideal situation. This helps explain the high correlation values attained.

However, looking at the individual qualities it is possible to explain why the collective results from all participants shows less need for change than the result of any of the groups individually. While confidentiality does not allow the ranked qualities to be published in this paper, some groups are more affected by some of the qualities than other groups; so they perceive these qualities as more important, while the other groups perceive the same quality as less important. For example, *Time Behaviour* is ranked seventh today, with all groups placing it in the seventh or eighth position today. In the perceived ideal situation the overall rank is only increased one place to sixth, but the same criterion is ranked second most important by R&D, sixth most important by TPM and tenth most important by SPM & PM. This situation acts to reduce the correlation coefficients for the individual groups, but still keeps it high when examining all results together.

There was a high level of agreement in the ranks given to qualities relating to features and project management both today and in the ideal situation. The area of greatest contention between the groups concerns qualities relating to *system properties*. SPM differs the most when examining the results of the three groups. The results highlight which qualities the groups agree on as important – such as *Scaleability*, *Time Behaviour*, *Robustness/Stability*, *Configurability/Product Customisability/Adaptability* and *Resource Behaviour* – and which qualities for which there are differing priorities – *Recoverability*, *Operability*, *Performance Management/Statistics*, *Upgradeability/Replaceability*, *Analysability*, *Testability*, *Containment/ ISP/Fault Tolerance* and *Security*.

5 Discussion

The methodology proposed in Section 3 has been applied to identify the level of alignment between internal success-critical stakeholders with the modification made after the pilot study described in Section 4.3. While the results from the case presented in this paper are not generalisable, it highlights what situations can be detected by the method and acts as a reference point for future applications of the method.

While in the case study some changes to the priorities given to the software product qualities would be perceived beneficial by each of the internal success-critical stakeholder groups, the extent of the changes required is reduced when considering all perspectives together. This can be seen most clearly with a number of qualities where the groups agree on their importance today, but some of the groups think some should be more important while other groups think they should be less important and it ends up in almost the same place. This result shows Theory-W in action, with the organisation having to balance conflicting stakeholder perspectives in order to achieve the optimal balance.

While the current processes seem to have done a reasonable job to balance the various concerns of software product quality, this is not explicitly visible to all of the stakeholders who felt that their needs were not being adequately addressed. One of the ongoing aims within Ericsson is to use these results to foster a greater understanding and dialogue between the internal success-critical stakeholders in terms of each other's needs.

6 Conclusion

This paper presents a methodology and results of a case study for examining the alignment between the internal success-critical stakeholder groups in software product quality. The results obtained by the method were interesting, valuable and very positive from the perspective of the industrial partner, Ericsson, with:

- The groups found to be aligned in perceived the priorities placed on different software product quality today; and
- Overall the participants in the case study perceive few changes necessary to improve the current situation.

The case study results highlight that different stakeholder groups have different priorities, and companies must be able to balance these differing opinions in order to achieve an optimal outcome. Key to achieving this outcome appears to be open and transparent dialogue and cross group communication and understanding. The results also provide an understanding of the context of software product quality for future work within the case setting.

The case study presented in this paper may not be representative of the software development industry, only involving two products from one company. Still, it provides some insights into how qualities are handled in an industrial context. Furthermore, the method can be applied in other situations to support the alignment of success-critical stakeholders in issues of software product quality. In turn, these additional results can help determine which of the results, if any, can be generalised.

This research will be used in three ways:

- This work is the first in a series of studies examining, with the intention to help improve, the alignment of company strategy, product strategy, product management and development efforts.
- This work is also the first in another series that is looking at different investment options and trade-offs in software development – like features, quality and staff training. Going forward the aim of this work is to support organisations improve the investment choices they make.
- The authors are also hoping to replicate parts of this study at different organisations to help achieve greater alignment in issues of software product quality and draw more general conclusions in this topic area.

Acknowledgments

We would like to thank Ericsson for their active involvement in and support of this research.

This work was partly funded by The Knowledge Foundation in Sweden under a research grant for the project Blekinge Engineering Software Qualities (BESQ) (http://www.bth. se/besq).

References

1. Beck, K.: Extreme Programming Explained: Embrace Change. Addison-Wesley, Reading (2000)
2. Kitchenham, B., Pfleeger, S.: Software quality: The elusive target. IEEE Software 13(1), 12–21 (1996)
3. Yourdon, E.: When good enough software is best. IEEE Software 12(3), 79–81 (1995)
4. Boehm, B., Ross, R.: Theory-w software project management principles and examples. IEEE Transactions on Software Engineering 15(7), 902–916 (1989)

5. Garvin, D.A.: What does “product quality” really mean? Sloan Management Review 26(1), 25–43 (1984)
6. Hoyer, R.W., Hoyer, B.B.Y.: What is quality? Quality Progress 34(7), 53–62 (2001)
7. Basili, V.R.: Evolving and packaging reading technologies. Journal of Systems and Software, Achieving Quality in Software 38(1), 3–12 (1997)
8. Boehm, B., Basili, V.: Software defect reduction top 10 list. Computer 34(1), 135–137 (2001)
9. Milicic, D.: Software Quality Models and Philosophies. In: Software Quality Attributes and Trade-Offs, pp. 3–19. Blekinge Institute of Technology (2005)
10. ISO9126: Software engineering – product quality – part 1: Quality model. International Standards Organization (2001)
11. Dromey, R.: Concerning the chimera. Software, IEEE 13(1), 33–43 (1996)
12. Chung, L., Nixon, B., Yu, E., Mylopoulos, J.: Non-Functional Requirements in Software Engineering. Kluwer Academic, Dordrecht (2000)
13. Herzwurm, G., Schockert, S., Pietsch, W.: Qfd for customer-focused requirements engineering. In: 11th IEEE International Requirements Engineering Conference, 2003, pp. 330–338 (September 2003)
14. Boehm, B., Jain, A.: An initial theory of value-based software engineering. Value-Based Software Engineering, 15–37 (2006)
15. Leffingwell, D., Widrig, D.: Managing software requirements: a unified approach. Addison-Wesley, Reading (1999)
16. Barney, S., Wohlin, C.: Software product quality questionnaire (2008), `http://www.bth.se/tek/aps/sba`
17. Berander, P., Jönsson, P.: A goal question metric based approach for efficient measurement framework definition. In: Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering (ISESE 2006), pp. 316–325. ACM, New York (2006)

Predicting Upgrade Project Defects Based on Enhancement Requirements: An Empirical Study

Lei He[1,2], Juan Li[1], Qing Wang[1], and Ye Yang[1]

[1] Institute of Software, Chinese Academy of Sciences
[2] Graduate University of Chinese Academy of Sciences
{helei,lijuan,wq,ye}@ itechs.iscas.ac.cn

Abstract. In upgrade project development, Enhancement Requirements (ER, e.g. requirement additions and modifications) introduce new defects to the project. We need to evaluate this impact to help plan later project schedule and resources. Typically, many of the existing prediction technologies estimate defects based on software size or process performance baselines. However, they are limited in estimating the impact of ER on product quality. This paper proposes a novel ER-based defect prediction method using information retrieval (IR) technique and support vector machines (SVM). We analyze the historical data of defects and requirement specifications of actual upgrade projects to establish multiple prediction models to estimate new defects introduced by ER. Then we design two experiments to validate the method and report some preliminary results. The results indicate that our method can provide useful support for impact analysis of requirement evolution in upgrade projects.

Keywords: Defect Prediction, Enhancement Requirement, Information Retrieval, Support Vector Machines.

1 Introduction

Software upgrade project is responsible for delivering an updated release or version of the software product. In these upgrade projects, changes to the existing software are requested, including modifications to some current functionality and additions of new functionality, in this paper, these changes are named as Enhancement Requirements (ER). Studies have shown that these functional enhancement activities have a close bearing on the reliability of the software product [1]. Typically, ER injects new defects to the upgrade project [1]. It is necessary to analyze the impacts of ER to evaluate and guide our software enhancement activities. One way is to estimate characteristics of the defects introduced by ER, for example, the type of the defects (user interface or functional defects), and the workload to fix these defects. Based on the estimated information the project manager can plan project schedule and resources more effectively.

Technologies for the identification and prediction of defects have been developed rapidly. These technologies have been applied successfully in actual software projects. Traditional prediction methods are usually based on the design or developing period of the software lifecycle. For example, several approaches predict defects by

Q. Wang et al. (Eds.): ICSP 2009, LNCS 5543, pp. 268–279, 2009.

analyzing the characteristics of the source code [2, 3]. And several researches solve the prediction problem by the metrics of software architecting [4, 5]. But there are still difficulties if we want to know the influences and consequences caused by ER.

This paper proposes a defect prediction method based on ER. ER is modified requirement or newly added requirement in the upgrade software project. We did an empirical study to investigate this predicting problem. Our basic idea is shown in figure 1. We associate historical defects with requirements. And when ER happens, we analyze the new requirement to find out the most similar old requirement. So according to the defects related to the old requirement, we can predict the most possible defects that will be introduced by this specific ER.

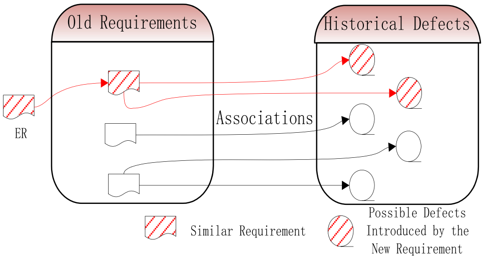


Fig. 1. Basic Idea of Our Approach

Actually, we classify the old requirements based on the properties of defects related to find out the similar requirements first. In our study, we consider four different properties of defects: number, workload, priority and type. The prediction results contain these four kinds of defect properties. IR technique is applied to establish associations of historical requirements and defects, and SVM is applied to establish prediction models. We did experiments to evaluate the performance of our methodology. The results show that this methodology can provide significant defect-predicting information to analyze the impact of requirement changes in the upgrade projects, and help the project manager to allocate the resources to reduce the costs and risks at the early stages of the software process.

The rest of this paper is organized as follows. Section 2 introduces related work. Section 3 describes the detail steps of our prediction method. In section 4, we present the process and results of applying our method to an actual project. We analyze the limitations and drawbacks of the predicting model in section 5. Finally, we give our conclusion in section 6.

2 Related Work

Software defect prediction techniques have been developed rapidly, which can be divided into dynamic and static approaches [6]. Basically, our defect prediction model is a static one which is based on defect-related data metrics. Static defect prediction approaches are based on different phases of software process. Malaiya et al. proposed a mathematic methodology on calculating defect density due to requirements volatility during different time of software process [8]. COQUALMO (COnstructive QUALity MOdel) [4] introduced by Boehm is a quality model extension to COCOMO II [4], which can be used to estimate defects injected in different activities, and defects removed by defect removal activities. Gou et al. proposed the BiDefect (process-performance Baselines based iteration Defect management) method [5] to support quantitative defect management in iterative development. Our defect prediction method is focusing on the requirement phase of the software lifecycle.

Static predicting approaches using machine learning (ML) techniques have been applied to empirical studies recently. Fenton at al. did an experiment to develop a causal model based on Bayesian net for predicting the number of residual defects that are likely to be found during independent testing or operational usage [7]. Their model incorporates a set of quantitative and qualitative factors describing a project and its development process, and it can be applied very early in the software lifecycle. SVM is a supervised ML model for classification and regression. It has shown its capability in solving quality-analyze and defect-predict problems. Xing proposed a technique to predict software quality by adopting SVM in the classification of software modules based on complexity metrics [9]. And Elish did an empirical study in predicting defect-prone software modules using SVM and compared its prediction performance against eight statistical and machine learning models [12]. The results indicate that the prediction performance of SVM is generally better than the compared models. SVM also performs well in our defect prediction methodology.

3 Defect Prediction Method Based on Enhancement Requirements

The prediction process of our method includes three steps: associating requirements and defects, classifying requirements and extracting features, modeling and predicting using SVM. Figure 2 shows the process of this approach. These three steps are discussed in sections 3.1 through 3.3.

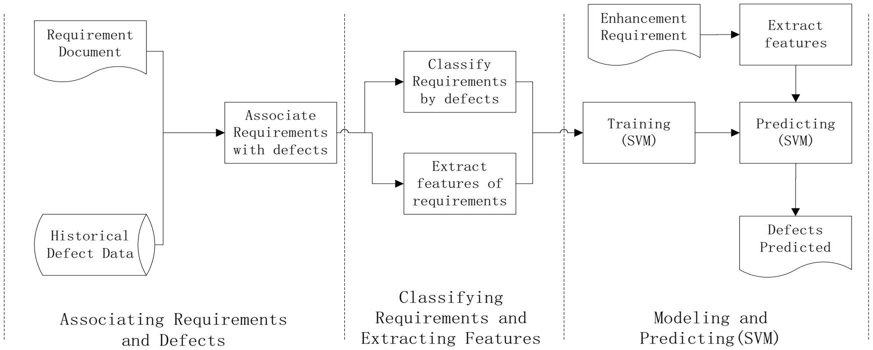

Requirement Document
Historical Defect Data
Associate Requirements with defects
Classify Requirements by defects
Extract features of requirements
Enhancement Requirement
Extract features
Training (SVM)
Predicting (SVM)
Defects Predicted
Associating Requirements and Defects
Classifying Requirements and Extracting Features
Modeling and Predicting(SVM)

Fig. 2. Overview of the Prediction Process

3.1 Associating Requirements and Defects

First, we establish association between defects and requirements. In our study, the defects and the requirements are both described in natural language, so we use IR techniques to associate the two parts. The association process is described in figure 3.

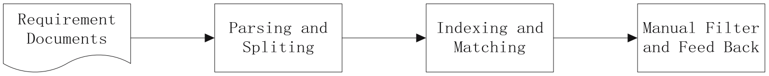


Fig. 3. The Associating Process

Defect descriptions are usually brief instructions, for example: "*System crashes when clicked the 'submit' button*". Requirements are comparatively complete specifications. In our study, we generally analyze the requirement specifications and split them into requirement items of use cases. These use cases are of the same format, for example:

·*Name: Submit Work Plan;*

·*Description: To save and submit work plan in the work space;*

·*Pre-conditions: User logged in;*

·*Basic Work Flow: 1. Enter the work space, open the work plan table. 2. Input work plan. 3. Click 'submit' Button; Additional Work Flow: None;*

·*Post-conditions: A new plan added in project window.* ……

We convert the associating problem into a text searching problem, by considering the defect description as the query, and the requirement item as the target document. And we parse the requirement documents by splitting them into requirement items, which are use cases. We build up an inverted index and use the formula below to get the matching scores of every defect to requirement items [10]:

$$\sum_{t\,in\,d} TF(t\ in\ r) * IDF(t) * Boost(t.field\ in\ r) * LengthNorm(t.field\ in\ r) \tag{1}$$

In the formula, "r" stands for requirement item, "d" stands for the defect description, and "t" is the term of the context. Table 1 lists instructions of the factors in the formula.

Table 1. Factors in Scoring Formula

Factors	Description
TF(t in r)	Term frequency factor for the term t in the requirement r.
IDF (t)	Inverse document frequency of the term t.
Boost (t.field in r)	Field boost, as set during indexing, default value is 1.0.
LengthNorm(t.field in r)	Normalization value of a field, given the number of terms within the field.

We use this scoring formula to calculate the relevance of every defect to requirement item. By setting up threshold value for the resulting scores, the associated couples whose scores are lower than the threshold are abandoned. Usually the threshold is adjusted for a few times to get the appropriate result set.

We also need manual intervention to raise the accuracy of the association. For example, it is helpful to ask the requirement engineer to filter the result set of the association. Then, feed-back work like re-weighting some factors in the formula is adopted after we filter the associated result.

At the end of this step, we can get a comparatively accurate association set with related requirement items and defects.

3.2 Classifying Requirements and Extracting Features

It is important to classify the original requirement items to get our SVM training data. We have to make our classification significant, meaning that the requirement items in the same class must have common features. These features will be the dimensions in the vectors of the SVM. However, it is nontrivial to discover and extract features of the requirement. Here is the method we adopt in this empirical study. Firstly, we try to classify the requirements by the standards we are interested in, that is, the properties of defects we are hoping to get in the prediction results. For example, the number and the work load of the defects related to one requirement item. Secondly, after dividing the requirement items into classes, we examine items in every class to analyze whether the classification is reasonable and whether the items have implicated features. Based on the analysis, we check and modify the classification standards repeatedly until we have satisfactory classification sets. Thirdly, we analyze the features of the requirements and determine their dimensions by experimental methods. Here the goal is to identify which factors influence the classification, which are the peculiar factors of requirement items inside every class, and their weights of influence.

We can divide the features into two types by their different effects for classification: the common ones and the special ones. The common features are the main parameters for all classification standards. And they are the basic and natural properties of the requirements, such as the context statistic features of the requirement item, including the “TF”, “IDF” and “DF” values of terms, which can be calculated to CHI values (a common feature of text property) [11]. The special features are the ones that act distinguishingly on different classification standards. For example, the text length of one requirement item is a special feature, because it influences the number of defects the item brings in, but it has no business of the type of the defects. Common features and special features used in our study are listed below.

Common Features: TF, IDF, DF. (calculated to CHI values of terms)

Special Features: Module of the requirement (MoR), text length of the requirement item (LoR), Number of items in event flow (NEF), Description in UI design (DUI), pre-conditions of the requirement (PCR), required data properties (RDP).

We may use different combinations of features according to different classification standards and characteristics of the projects.

Following the three steps mentioned earlier in this section, we get several classification sets according to the defect properties. After we analyze and select features of every classification set, we need to quantify the features of each requirement item to generate inputs to Support Vector Machines. Some features are naturally numeric, such as the TF, IDF values of the terms. We just need a unitary processing on these features for later use. However, some features are not initially quantitative, so we must map them by the definitions. For example, the module of the requirement, if there are 10 modules in the project, we mark the item according to the module it belongs to, from 1 to 10. Then normalize these numbers by 10 to get the feature data.

Now we have the classifications and the features with numeric values of the historic data, so we can format them to generate input files of Support Vector Machines.

3.3 Modeling and Predicting Using SVM

Support Vector Machines are a set of related supervised learning methods used for classification and regression [13]. SVM has the characteristics below to make it perform well in solving classification problems [12]. These characteristics matches quite well with our requirements data.

I. It can be generalized well even in high-dimensional spaces under small training sample conditions. This means that the ability of SVM to learn can be independent of the feature space dimensionality.

II. It gives a global optimum solution, since SVM is formulated as a quadratic programming problem.

III. It is robust to outliers. It prevents the effect of outliers by using the margin parameter to control the misclassification error.

IV. It can model nonlinear functional relationships that are difficult to model with other techniques.

Here, we adopt classifications and features of requirement items generated in section 3.2 as the input to SVM. For each classification set, the features build up the dimensions of SVM classification hyperspace. We scale dimensionality of each feature to represent its weight. Each requirement item is one point in the hyperspace. We train SVM models according to the input data to get classification margins. When new requirement comes, by extracting the same kinds of features as the input data and importing them into the trained SVM model, we can find out what class exactly the requirement belongs to. According to the property of the class, we get the property of the defects predicted.

SVM models need to be amended according to the characteristics of the targeting project before actual prediction work. We modify parameters of the SVM model, types and weights of requirement features by going through testing processes, which are analysis between prediction results and testing data. Testing data are acquired from parts of the original classification data.

4 Experiment

4.1 Background

Our study is based on the “SoftPM” project. This “SoftPM” project is a software quality management platform, targeting to facilitate the process improvement initiatives in many small and medium software organizations. The first version was released in 2002 and 9 versions have been released until now. We have items of defect records of the 9 versions stored in the project database. These defects are all introduced or related by the ER. Table 2 shows main characteristics of the defects in the database.

The requirements of every version of project “SoftPM” are stored in software requirements specification including Use Case specification and non functional requirements specification. In our study, we mainly use Use Case specification. The specification is of the standard format, including name, description, actor, event flow pre-condition and post-condition of use cases.

Table 2. Detailed Information of Historical Defects

Name	Description
Title	Title of the defect.
Description	Brief description of the defect.
Submitter	Submitter of the defect.
Module	Module of project the defect belongs to
Workload	Workload to fix the defect (man-hour).
Priority	Priority of the defect: Normal, Serious and Critical.
Type	Type of the defect: UI, Internal or Integrated.
Step	Steps to re-appear the defect.

4.2 Experiment Design

We design two experiments on the historical data of "SoftPM" project to establish and evaluate the performance of our predicting model. We acquire 4,893 defects and 581 use cases of the project. Then we get classification data of these defects and use cases through step 1 and 2 of our methodology. The first experiment is to evaluate the accuracy of prediction in the four separated defect properties (as table 3 lists). We use 80% of the classification data as the training data and the rest as the test data to calculate the prediction accuracy. The second experiment is to predict defects introduced by a specific upgrade version of the project. By comparing the predicted defects with the actual ones recorded in the project database, we evaluate the performance of our prediction model.

We predict properties of defects introduced by ER as table 3 shows.

Table 3. Defect Properties

Property of Defects	Description
Number	The number of the defects introduced by the requirement
Average Workload	The average workload (man-hour) to fix the defects
Priority	The priority of the defects, based on the majority defects related to the specific requirement item.
Type	The type of the defects, based on the majority defects related to the specific requirement item.

Based on the above four kinds of properties to predict, we define four classification standards and their class boundaries to classify the original requirement items. Class boundaries and requirement features, including their weights, are modified through the prediction process of experiments to match the characteristics of the "SoftPM" data. Table 4 shows the final class boundaries and features of each requirement class. The numbers in the brackets of the features are the final dimensionalities of features.

Based on the table above, we can divide the 581 items of requirements into classes by four standards. So with the features, we can train four different models of SVM to predict four kinds of defect properties introduced by ER.

Table 4. Classification Information of Original Requirements

Classification Standard	Classes	Boundary (Range)	Requirement Features
By Number of Related Defects	Extra Few	0~5	CHI (1000) MoR (50) LoR (100) NEF (100) PCR (50)
	Few	6~10	
	Medium	11~29	
	High	30~49	
	Extra High	More than 50	
By Average Workload of Related Defects	Extra Low	0~24 (man-hour)	CHI (1000) MoR (100) LoR (50) NEF (150)
	Low	25~48	
	Medium	49~144	
	High	145~288	
	Extra High	More than 289	
By Priority of related Defects	Normal	Regarding to the defect properties recorded in the database	CHI (1000) MoR (200) NEF (100) PCR (50)
	Serious		
	Critical		
By Type of related Defects	UI	Regarding to the defect properties recorded in the database	CHI (1000) MoR (100) NEF (100) DUI (50) RDP (50)
	Internal		
	Integrated		

4.3 Experiment 1

In the first experiment, we randomly select 80% of the requirements with their features as the training data. These requirements are divided into classes according to the four standards listed in table 4. Then we input the training data to SVM to train four prediction models. After these SVM models have been trained, the rest 20% of the requirement items (116 items) are input as the test data. We calculate the accuracy by divide the hits of the predicted requirement classifications by the total ones. Recall rate is calculated as: considering each classification standard, divide the sum of the smaller one of the predicted and actual classification number by the total testing number. Table 5 shows the input data distribution and the classification results of Experiment 1.

We can tell from the predicting results that we get higher accuracy when the class-number is smaller. The accuracy of three-class predicting is higher than the five-class one. This makes sense, because the more class number we have, the more detailed information we will get, and that will cause decrease of accuracy (Prediction based only one classification will sure be 100% accuracy but brings no information). We should find a balance of information provided and accuracy of results during the actual predicting process.

Table 5. Predicting Results of Experiment 1

Standard	Classes	Overall Requirements Distribution	Test Requirements Distribution		Recall (predicts/total)	Accuracy (hits/total)
			Predicted Class.	Actual Class.		
Number	Ex-Few	113	25	21	86.2% (100/116)	65.5% (76/116)
	Few	91	20	23		
	Medium	183	47	35		
	High	134	15	26		
	Ex-High	60	9	11		
Average Workload	Ex-Low	76	25	33	82.8% (96/116)	62.1% (72/116)
	Low	146	40	21		
	Medium	157	31	30		
	High	114	12	13		
	Ex-High	88	8	19		
Priority	Normal	252	51	58	94.0% (109/116)	73.3% (85/116)
	Serious	187	29	27		
	Critical	142	36	31		
Type	UI	170	30	38	93.1% (108/116)	78.4% (91/116)
	Internal	133	36	34		
	Integrated	278	50	44		

4.4 Experiment 2

In the second experiment, we analyze the data by project versions. Prediction processes and results of the upgrade versions are similar. Here we just analyze the 7th version of the “SoftPM” project as an example. According to the release instructions of version 7, it has 20% functional enhancements upon version 6. Version 7 is a typical upgrade project. We analyze requirements specification of version 7 by comparing it with the specification of version 6. ER in version 7 can be divided into two parts: the “newly added” requirements and the “modified” requirements. For those “modified” ones, we need to calculate the correlation of the requirement item in version 7 and the former one in version 6 to get the coefficient “C” (0<C<1, if C=1 there is no difference between two items). Classification process of version 6 requirement items is same as Experiment 1. We import the version 6 requirement items with their features into SVM as the training data, and the ER items introduced by version 7 as the test data. The predicted classification results of ER items are listed in table 6. There are totally 48 ER items in version 7 of “SoftPM” project.

Table 6. Predicted Classification Results of Experiment 2

Classification Standard	Number of Requirement Items In Each Class
By Number of Related Defects	Extra Few:19; Few:21; Medium:7; High: 1; Extra High: 0
By Average Workload of Related Defects	Extra Low: 7; Low: 16; Medium: 15; High: 8; Extra High: 2
By Priority of Related Defects	Normal: 29; Serious: 13; Critical: 6
By Type of Related Defects	UI: 17; Internal: 13; Integrated: 18

Now we quantify these classification results to get an intuitive predicting statement of defects properties. We assign actual value for each class in the first two classification standards according to the distribution of the training data. So we can calculate how many defects exactly introduced by the change of requirement, then the total workload and the distribution of defects' priorities and types. For these "modified" requirement items in experiment 2, when we estimate the number and workload of the predicted defects, the coefficient "1-C" is required to be multiple to get the final results. We analyze the actual statistic properties of defects which introduced by version 7 requirements, and compare them to the predicted ones to evaluate the results. Figure 4 shows the detail results of the compares.

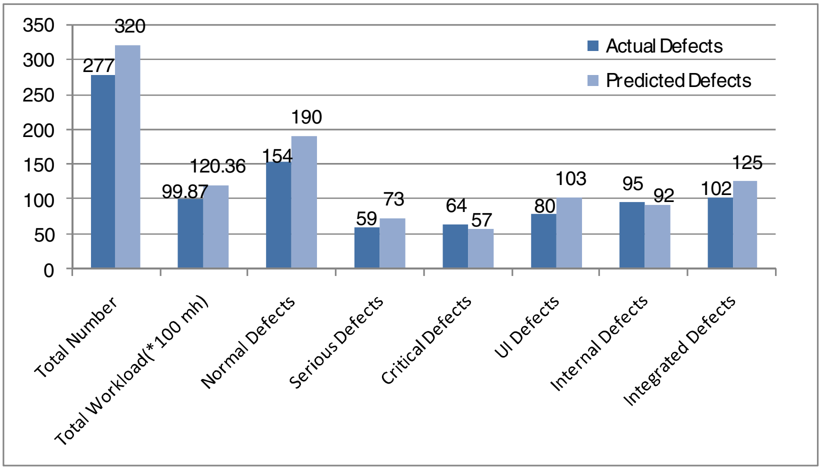


Fig. 4. Compare of Actual and Predicted Defects

We can tell that in experiment 2, the results are more meaningful and the gap between the actual and predicted data is smaller. This phenomenon is because that the errors of the predicting results are offset with each other so the holistic performance of the statistic results is much better.

We can also see from the results that most predicted numbers of defects are larger than the actual ones. This is because when we convert the predicted classification results into the quantified prediction reports, we calculating the numbers of each class by assigning a specific value standing for the class using the average value of the input data. Obviously this value assignment is larger than the actual one in the experiment. Deeper statistic analysis is required on the input data to reduce the error.

From the results and analysis of the two experiments, our approach shows that its predicting results are very helpful in analyzing impacts on requirement enhancement.

5 Threat to Validity

The results of the experiments prove that our defect prediction approach is meaningful in estimating impacts on ER. However, we can also find some limitations and threats.

(1)Our defect-predicting model can only be applied on upgrade projects and must have got sufficient and continuous historical information of defects and requirements. We can do nothing with newly started projects using this prediction method. And we have to go through the whole analysis and modeling process again when we apply the prediction method to a different project.

(2)Our model works well with requirement addition or modification, but the definition of the prediction process is lack for requirement deletion. The reason is that it is hard to connect historical defect information to the deleted requirement items. We tried to consider the deleting activities as the inverted process of the adding ones, but the predicting results are really unsatisfactory. Some new associating techniques must be brought in to solve this problem.

(3)Features of requirements are hard to distinguish and extract, and to evaluate the weight of each feature accurately is difficult. Now we just give them initial values and modify the values through the test experiments to get better prediction results. Future work is required to ensure the completeness of the features and to reevaluate the importance of each feature to the classification standard.

(4)The prediction accuracy is fine when we divide the original requirement items into 3 or 5 classes. But if we want more detailed characteristics of predicted defects, we have to divide the requirements into more classes. And this will decrease the accuracy a lot. There is space for us to optimize our prediction model to improve the accuracy.

6 Conclusion

In this paper, we propose an empirical methodology to predict defects based on enhancement requirements. A novel framework of defect prediction is put forward. It implements a new method for evaluating and analyzing the impact of requirement enhancement in upgrade project. It also provides considerable and convincing prediction results to estimate the costs and risks of requirement evolution.

We establish connections between defects and requirements to acquire basic standards of requirements classification. This association process provides the inspiration of our methodology. And in this prediction model, SVM has shown that it is adaptive to modeling nonlinear functional relationships of requirement data which are difficult to model with other techniques [9]. The results of experiments reveal the effectiveness of SVM in this specific prediction problem.

Although there are some limitations and drawbacks in the model, it provides a very promising and significant method for defect-predicting and requirement management. Our future work will focus on optimizing the algorithm of the methodology to enhance the accuracy of the results. We will conduct additional empirical studies with other datasets to further support the theories of this paper. We believe that this defect-predicting frame has a rather bright future and its value will be testified.

Acknowledgements

This research is supported by the National Natural Science Foundation of China under grant No. 90718042 and No. 60803023, the National Basic Research Program (973 program) under grant No. 2007CB310802, the Hi-Tech Research and Development Program of China (863 Program) under grant No. 2007AA010303.

Special thanks to Jian Gao, Fei Dong, Yin Li, Hao Jia, LingJun Kong, ZhongPeng Lin and Jing Du for their great support to our research work.

References

1. Lanning, D.L., Khoshgoftaar, T.M.: The Impact of Software Enhancement on Software Reliability. IEEE Transactions on Reliability 44(4), 677–682 (1995)
2. Koru, A.G., Liu, H.: Building Effective Defect-prediction Models in Practice. Software, IEEE 22(6), 23–29 (2005)
3. 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)
4. Boehm, B.W., Horowitz, E., Madachy, R., et al.: Software Cost Estimation with COCOMO II. Prentice Hall PTR, Upper Saddle River (2000)
5. Gou, L., Wang, Q., Yuan, J., Yang, Y., Li, M., Jiang, N.: Quantitative Managing Defects for Iterative Projects: An Industrial Experience Report in China. In: Wang, Q., Pfahl, D., Raffo, D.M. (eds.) ICSP 2008. LNCS, vol. 5007, pp. 369–380. Springer, Heidelberg (2008)
6. Wang, Q., Wu, S.J., Li, M.S.: Software Defect Prediction. Journal of Software 19(7), 1565–1580 (2008)
7. Fenton, N., Neil, M., Marsh, W., Hearty, P., Radlinski, Ł.: On the Effectiveness of Early Life Cycle Defect Prediction with Bayesian Nets. mpir Software Eng. 13, 499–537 (2008)
8. Malaiya, Y.K., Denton, J.: Requirements Volatility and Defect Density. In: 10th International Symposium on Software Reliability Engineering, p. 285 (1999)
9. Xing, F., Guo, P., Lyu, M.: A novel method for early software quality prediction based on support vector machine. In: Proc. of the 16th IEEE Int'l Symp. on Software Reliability Engineering (ISSRE 2005), pp. 213–222 (2005)
10. Gospodnetic, O., Hatcher, E.: Lucene in Action. Maning Publication (2006)
11. Qin, J., Lu, R.Z.: Feature Extraction in Text Categorization. Journal of Computer Applications 23(2), 45–46 (2003)
12. Elish, K.O., Elish, M.O.: Predicting Defect-prone Software Modules using Support Vector Machines. The Journal of Systems and Software 81, 49–660 (2008)
13. Gunn, S.R.: Support Vector Machines for Classification and Regression, Technical Report. Faculty of Engineering, Science and Mathematics School of Electronics and Computer Science, University of Southampton (1998)

Incremental Process Modeling through Stakeholder-Based Hybrid Process Simulation

Xu Bai[1], Liguo Huang[1], and Supannika Koolmanojwong[2]

[1] Southern Methodist University, Dallas TX 75205, USA
{bxu,lghuang}@engr.smu.edu
[2] Center for Systems and Software Engineering, University of Southern California, Los Angeles, CA 90089, USA
koolmanu@usc.edu

Abstract. Both the process modeling and process simulation are necessary components of process automation. A Process Modeling Language (PML) is a set of description tools that define processes attributes and constrains in a specific domain. Process modeling stakeholders may have different levels of dependencies on different types of PMLs. They also have various perspectives for modeling a process. Discrete and continuous PMLs are complimentary in modeling the process at different levels of abstraction and to address different stakeholders' perspectives. The hybrid process simulation combines micro-level discrete process models with the macro-level continuous process models to capture process dynamics and deploy process optimization. This paper proposes an incremental approach based on the hybrid simulation in modeling a software process at different levels of abstraction in order to address different stakeholders' perspectives. By addressing stakeholders' concerns in hybrid simulation at each process segment, this approach incrementally integrates internal process dynamics and modifications due to external changes into process model that cannot be easily achieved by individually using discrete or continuous modeling approaches.

Keywords: Process Modeling, Process Simulation, Process Automation, Process Modeling Language, Hybrid Process Simulation Model.

1 Introduction

1.1 Motivation

The process is defined as a logical ordering of people, procedures, technologies and work activities designed to transform information, materials and energy into a specific result [1]. Resources and activities are two basic elements of software process. Process Modeling Language (PML) is used to describe the process and capture the essential properties of process by linguistic abstraction. Most latest PMLs support the process visualization and formalization. Process formalization always leads to programmable languages and simulation/interpretation tools that automate the process execution.

Q. Wang et al. (Eds.): ICSP 2009, LNCS 5543, pp. 280–292, 2009.

One of the main goals of process automation is to provide computer-based real-time support and guidance for process enactment. The PMLs play key roles in the process automation. There are many PMLs proposed in the literatures, however, no single approach becomes dominant in practice. Various PMLs provide different process modeling capabilities, based on their design intuitive and linguistic properties. In addition, various stakeholders are involved in the process modeling activities, such as process managers and process engineers, etc. Different stakeholders may have different perspectives and concerns in process modeling. Based on these observations, we did a comprehensive survey and analysis of several widely used PMLs including Little-JIL [2,3], Petri Nets and its variants [4], System Dynamics [5], etc., and categorized them into discrete and continuous PMLs [6]. We found that the modeling capabilities of discrete and continuous PMLs are complementary at different levels of abstraction and in addressing different stakeholders' perspectives. Thus we propose a stakeholder-based approach to build the process model by incrementally integrating discrete and continuous PMLs from different stakehoilders' perspectives, which emphasizes the stakeholders' involvement during process modeling activities.

Hybrid process simulation has become an increasingly active research area in software process automation in the last decades. By combining continuous and discrete model simulation approaches, the hybrid process simulation model is able to break the limitations of applying any individual simulation method and sensitively capture the dynamics of software processes. In this paper, we summarize the current hybrid simulation approaches and extend the capability of hybrid simulation to address stakeholder's perspectives with inevitable changes of process evolution.

1.2 Major Contribution

Process modeling stakeholders' roles are seldom identified during process modeling and simulation activities, as most of them are not directly involved in building the process models. However, in practice, different modeling and simulation approaches are chosen based upon their capabilities to address stakeholders' concerns at different software development and maintenance phases, where stakeholder classes and phases are two dimensions of stakeholders' perspectives.

In this paper, we propose a stakeholder-based approach to model process by incrementally integrating discrete and continuous PMLs with hybrid simulations. Our incremental process modeling approach 1) addresses two dimensions of concerns,i.e., stakeholder classes and software development phases; 2) investigates the information flow of hybrid process simulation and tailors the hybrid modeling method to be adaptive to environment changes. Our case study shows that our approach can efficiently adjust the process model attributes in response to these changes.

The rest of this paper is organized as follows: Section 2 summarizes related works on software process modeling and simulation. Section 3 elaborates our incremental process modeling approach through stakeholder-based hybrid process simulation. Section 4 presents a case study to illustrate our method and then analyzes the results. Section 5 summarizes our research and envisions the future research directions.

2 Related Works

2.1 Software Process Modeling Languages

There have been a number of PMLs proposed during the last two decades. These PMLs can be classified into discrete PMLs and continuous PMLs based on their modeling perspectives [6].

Discrete PMLs. E^3 [7] provides an early object oriented approach to process modeling, which uses pre-defined classes and relations denoted by graphical symbols. DYNAMITE is formally defined in PROGRESS, which is an executable specification language that uses UML in early implementation[8].

Melmac [9], Slang [10] and Object Petri-Net (OPN) [4] extended the traditional Petri Nets to represent the software process. In particular, OPN is capable of supporting the separation of concerns among different process modeling perspectives using the object oriented approach.

Little-JIL [2,3] is a subset of JIL [11] with visualization support. Software process is modeled as a tree of steps, whose leaves represent the units of work: steps. The structure of step tree represents the way in which the work will be coordinated [12]. Little-JIL uses non-leaf steps to capture step ordering. Little-JIL also employs late-binding techniques in resource management.

Continuous PMLs. System dynamics (SD) is an approach to model the behaviors of complex systems over time, e.g. the defect generation flow in a software development process. It has been applied to business/software process modeling to deal with internal feedback loops and time delays that affect the behaviors of the entire process at the system level [5,13]. SD provides a systematic view of software process, which can capture the process dynamics and interactions among project attributes thus to probe process improvement solution by conducting sensitivity analysis on process attributes.

2.2 Software Process Simulation

Process models implemented in Little-JIL and OPN can be simulated based on state transition, and the results of modeled process can be predicted [14]. Discrete event simulation (DES) interprets process as a series of entities flowing through event sequences and captures the effects of variation in the entities on activities and the dependencies among activities [15]. Simulating the discrete process model can also help validate and verify constrains between discrete events, especially constrains in resource allocation. The limitation of discrete model simulation is that the changes inside each event cannot be captured since the activity is the minimum process advancing unit. The process attributes that keep changing during the entire life cycle of software development can only be approximated at each event advance, which is considered problematic [15].

SD was first applied in process simulation in late 80's [16] and then has been developed in [17] [18] [19]. In comparison to discrete process simulation, continuous process simulation is ideal to address the dynamic factors in process. However, it does not easily to represent the individual activities in process and the constraints among these activities. For example, the process work flow structure can not be modeled by continuous models. Moreover, the continuous model is frequently used in sensitivity analysis, as the quality of model outputs relies heavily on the quality of model inputs.

As an integration of discrete and continuous process simulations, hybrid process simulation can address both the micro-level and macro-level process dynamics. However, integrating these two simulation approaches faces the issues of compatibility of process attribute forms and synchronization of executing simulation. Most hybrid approaches falls in vertical integration [20], in which discrete model was first built at a lower level, and then continuously calculates the process factors and incorporates the feedback loop at the system level [21]. [20] proposed a horizontal approach to integrating discrete and continuous process simulations at different phases of software development. However, none of these approaches tried to combine vertical and horizontal integration in order to address the stakeholders' concerns in process modeling and simulation.

3 A Stakeholder-Based Approach in Incremental Process Modeling

3.1 Stakeholders' Dependencies on Process Modeling Languages

The stakeholder classes involved in process modeling and simulation include process performer (PP), process engineer (PE), process manager (PM), customer (CU), end user (EU), educator (ED), tool provider (TP), researcher (RS), union representative (UR), regulator (RG), standardizer (SD) and domain specific stakeholder (DS). Based on their responsibilities and activities in process modeling and simulation, stakeholders may have different levels of dependencies on discrete and continuous PMLs, as shown in Table 1 [6]. We use low, medium or high to indicate various levels of dependencies. For example, process performers are assigned tasks by the process manager, motivate to complete tasks at the deadline, provide feedback to the process manager and adjust tasks if necessary. They focus on tasks at the activity level and depend on discrete PMLs for modeling task related attributes. However, they are not much concerned about process attributes above the activity level. Process engineers play the most active roles in process modeling and simulation. Their responsibilities include designing the process, choosing the PML, building and verifying the process model, simulating process execution and optimizing the process. Thus they highly depend on both discrete and continuous PMLs to help in achieving their goals (e.g., optimizing resource allocation, minimize system defect density) at the activity level, sub-process level and system level.

3.2 Stakeholder's Perspectives in Continuous Process Modeling

Different process modeling stakeholders also have different perspectives in software process dynamics. These perspectives include workforce modeling, earned value evaluation, software evolution, software reuse, quality and defects, requirement volatility [22] and other process-specific perspectives. Based on the analysis of stakeholders' roles in process modeling activities [6], Table 2 summarizes their perspectives and their associated stakeholders classes. The acronyms for stakeholders are identical to those

Table 1. Levels of Dependencies on Discrete and Continuous PMLs

Stakeholder Classes	Discrete PMLs	Continuous PMLs
Process Performer (PP)	High	Low
Process Engineer (PE)	High	High
Process Manager (PM)	High	High
Customer (CU)	Medium	High
End User (EU)	High	High
Educator (ED)	High	High
Tool Provider (TP)	High	High
Researcher (RS)	High	High
Union Representative (UR)	Medium	Medium
Regulator (RG)	Low	Low
Standardizer (SD)	High	High
Domain Specific stakeholder (DS)	High	High

Table 2. Continuous Process Modeling Perspectives and Their Associated Stakeholder Classes

Continuous Process Modeling Perspectives		Associated Stakeholder Classes							
		PP	PE	PM	CU	EU	ED	TP	RS
People	Workforce Modeling		X	X	X	X	X	X	X
	Exhaustion and Burnout	X	X	X	X	X	X	X	X
	Learning		X	X	X	X	X	X	X
	Team Composition	X	X	X	X	X	X		X
Process/Product	Inspection		X	X	X	X	X		X
	Software Evolution		X		X		X		X
	Software Reuse		X	X	X	X	X		X
	COTS		X	X	X	X	X	X	X
	Software Architecting		X	X	X	X	X	X	X
	Quality and Defect		X	X	X	X	X	X	X
	Requirement Volatility		X	X		X	X	X	X
	Process Improvement		X	X	X	X	X	X	X
Project	Integrated Project Modeling		X	X		X	X	X	X
	Business Case Analysis	X	X	X		X	X	X	X
	Personnel Resource Allocation		X	X		X	X	X	X
	Staffing	X	X	X		X	X	X	X
	Earned Value		X	X		X	X	X	X

defined in section 3.1. The perspectives are classified into three categories: People, Process/Product and Project. The ”X” mark indicates the stakeholder class and its concerned perspectives in continuous process modeling.

3.3 Incremental Process Modeling Based on Stakeholders’ Perspectives

The primary objective of our incremental process modeling is to improve the quality of the process model and optimize the process via the hybrid simulation of both discrete and continuous PMLs based upon stakeholders’ perspectives. Our approach improves the process adaptability to internal (process dynamics) or external (environment) changes by incrementally integrating affected process attributes into the process model at different levels of abstraction determined by various stakeholders’ perspectives. Hybrid process simulation uses the continuous PML to model the changes in the affected process attributes and provide feedbacks to adjust the corresponding attributes in the discrete process model based on different stakeholders’ perspectives [20].

Our incremental process modeling method models the process at several levels of process abstraction. At the activity level, we model each process activity with both discrete and continuous PMLs, and then perform the discrete and continuous simulation synchronously. By comparing the process simulation results from both the discrete and continuous models on specific process attributes of each activity, we can adjust the corresponding process attributes in the discrete model to obtain desired optimal results. At the sub-process level, where stakeholder's perspectives may cover several phases, the hybrid process simulation results provide the feedback loop to the involved process activities in those phases, so that the related process attributes in the activity can be adjusted. Also, the hybrid process simulation results can be used as the inputs to model process activities in subsequent phases. At the system level, the overall process performance can also be measured and optimized using the similar approach. Our hybrid process simulation approach combines and improves these horizontal and vertical [20] hybrid simulation approaches by addressing different stakeholders' perspectives and improving the overall quality of process model by incrementally adjusting affected process attributes, integrating environment changes and optimizing process at three levels of abstraction as shown in Fig.1.

In Fig. 1, the horizontal axis represents the process time line and the vertical layers represent different level of abstraction based on different stakeholders' perspectives. Discrete process model is built for all activities in the process.

In the activity level, multiple continuous models can be constructed for different activities based on various stakeholders' perspectives. For example, stakeholder A in Fig. 1 concerns with activity a, while B concerns with activity b. We simulate the built discrete and continuous models synchronously in the activity level and evaluate the results. When the continuous process simulation results do not satisfy the expected results from the discrete process simulation, we identify and adjust the corresponding process attributes, and then evaluate the adjusted process attributes by simulating the hybrid model again. The adjusted attributes are accepted until the simulation results show the consistency between the discrete process simulation and continuous simulation. Such local optimization improves the process model by verifying and adjusting process attributes of each activity.

At the sub-process level, certain stakeholder class may concern with a set of activities across different phases based on its perspectives (see Table 2 and 3). For example, stakeholder C is interested in phase 2 and 3 in Fig. 1. A sub-process level hybrid simulation can be executed on the combined phase 2 and 3 using the continuous model based on C's perspective and discrete model covering involved activities in these phases. The dependencies among activities from the discrete process model are used in simulating the continuous process model. The simulation results are evaluated and the process attributes in the discrete model are adjusted to yield desired or optimized results. Finally, the adjusted attribute values are fed back into the continuous model for the next round of simulation.

A system-level hybrid simulation is similar to the sub-process level simulation except that it is based on the perspectives of stakeholders who concern about the entire process. This is illustrated as stakeholder D's perspective in Fig. 1.

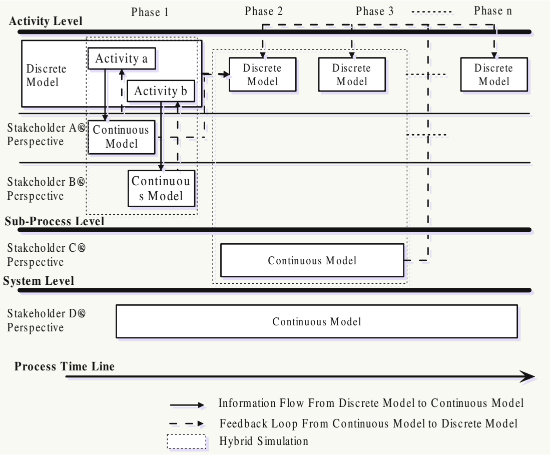


Fig. 1. Overview of Incremental Process Modeling with Hybrid Simulation from Different Stakeholders' Perspectives

3.4 Information Flow between the Discrete Model and Continuous Model

In order to integrate the discrete and continuous process models at the same perspective, we need to identify and model the information flow between them. We define $\mathit{Attribute}_{DM}$ as process attributes modeled by the discrete model and $\mathit{Attribute}_{CM}$ as process attributes modeled by the continuous model. $\mathit{Attribute}_{Interested}$ is defined as the process attributes that a specific stakeholder is interested in. Then we have

$$\mathit{Attribute}_{Interested} \subseteq (\mathit{Attribute}_{DM} \cap \mathit{Attribute}_{CM}),$$

where $\mathit{Attribute}_{Interested}$ includes both numeric and constraint attributes. these attributes are represented in different forms in discrete and continuous models. For instance, a numeric attribute can be a discrete value in discrete models while being a continuous distribution in continuous model. The constraint attributes define the modeling scope of a stakeholder's perspective, such as design review, defect detection etc.

In discrete models, the numeric attributes of a process activity usually have a fixed values during the execution of the activity. It is impossible for us to change these attributes, as discrete models treat each activity as the smallest advancing unit. For example, the manpower allocation and total workload are always defined as a fixed number for a process activity in discrete models during each simulation. In continuous models, the values of an attribute can be modeled as a statistical distribution. Process attributes such as the productivity are always modeled as a continuous distribution over time.

To implement hybrid process model simulation, first we need to first identify those attributes that are semantically same but in different forms in both discrete and continuous models. We define $\mathit{Attribute}_{IC}$ as attributes having the identical forms in both discrete and continuous model. Attributes having different forms in either models are respectively defined as $\mathit{Attribute}_{DMC}$ in the discrete model and $\mathit{Attribute}_{CMC}$ in the continuous model. Thus, we have

$$\mathit{Attribute}_{Interested} = \mathit{Attribute}_{IC} \cup \mathit{Attribute}_{DMC} \cup \mathit{Attribute}_{CMC}$$

Given the relationships (R) between these attributes, we can build simulation equations for the discrete model (DM) and continuous model (CM), where E_{DM} is the discrete simulation equation and E_{CM} is the continuous simulation equation. Thus,

$$E_{DM} = \mathrm{R}(t, \mathit{Attribute}_{IC}, \mathit{Attribute}_{DMC}) \text{ and } E_{CM} = \mathrm{R}(t, \mathit{Attribute}_{IC}, \mathit{Attribute}_{CMC})$$

E_{DM} and E_{CM} are simulated synchronously on time t. Usually one attribute is modeled in each hybrid simulation.

4 Case Study

4.1 Baseline Project and Process Models

The ISPW-6 software development process is a standard software process modeling example, where the core problem is a relatively confined portion of the software change process, focusing on the designing, coding, unit testing, and management of a localized change to a software system [23]. We will use this model to illustrate how to use our stakeholder-based approach to incrementally model the process by integrating the discrete and continuous PMLs to perform hybrid simulation. In this case study, we will focus on schedule and defect dynamics of the software process. We use the empirical data as shown in Table 3 from [19]. The baseline project core has 80,000 lines of code (LOC) in COBOL and 10,667 LOC as the increment overheads. The project was scheduled for one calendar year with 15 full-time engineers. A moderate change occurred late in the development phase due to the requirement volatility, an additional 5000 LOC workload need to be modeled to verify the schedule and quality constraints. Table 4 shows the planned schedule and 10 engineers are allocated to the rework tasks.

In this scenario, the process performer needs to follow the schedule assigned by the process manager. The process manager wants to verify and maintain the planned deadlines for each activity and the entire process. And the process engineer tries to estimate the residual defect density in the delivered product to ensure overall system quality is satisfied. The results from Table 1 indicate that the process performer is highly dependent on the discrete PMLs to obtain detailed task information but not much on the continuous PMLs, while the process manager and the process engineer require both discrete and continuous PMLs to satisfy their sub-process level goals.

Table 3. Baseline Project Attributes

Attributes	Value
Project Size	90,667 LOC
Increment 1	22,667 LOC
Increment 2	32,000 LOC
Increment 3	32,000 LOC
Schedule	250 Days
Team Size	15 Engineers
Estimated Budget	3750 Man-Days
Nominal Development Productivity	40 LOC/Man-Day
Nominal Defect Generation Rate	33 errors/KLOC
Nominal Defect Regeneration Rate	4:1
Nominal Defect Detection Rate	0.84
Nominal Review Productivity	220 - 1100 LOC/Man-Day
Nominal Test Productivity	40 LOC/Man-Day

Table 4. Planned schedule for Change

Activity	Man Power	Days
Schedule and Assign Tasks	1 Project Manager	1
Modify Design	6 Design Engineers	8
Review Design	4(1 DE, 1 QA, 2 Other)	1
Modify Code	6 Design Engineers	10
Modify Test Plans	4 QA Engineers	3
Modify Unit Test Package	4 QA Engineers	8
Test Unit	10 (6 DE, 4 QA)	10

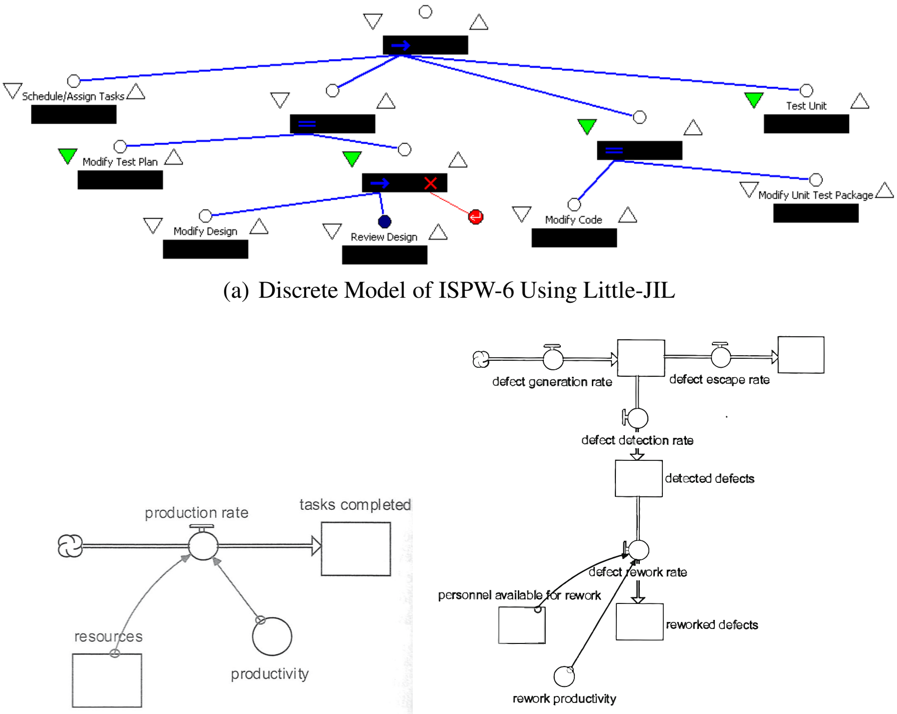
We used Little-JIL to build the discrete model of the ISPW-6 process [23] as shown in Fig. 2(a). Little-JIL enforces the critical path interdependencies among activities. And we applied SD for continuous modeling. Fig. 2(b) shows the SD model for each activity, to help the process manager to verify the planned deadlines from earned value perspective after the change occurred. Fig. 2(c) shows the process engineer's perspective on defect dynamics, which is at the sub-process level covering *Modify design*, *Review design*, *Modify code*, *Modify unit test plan* and *Test unit* activities of the process.

4.2 Deadline Verification and Adjustment

The process manager intends to verify if each activity and the entire process can meet the planned deadline in Table 4, and make adjustment to meet the deadline if necessary. He/She depends on both discrete and continuous PMLs to verify and maintain the deadline of this project from the earned value perspective in Table 2.

Activity Level Hybrid Simulation. At the activity level, the discrete PML Little-JIL is used to model dependencies among activities and the planned process attributes. The continuous PML SD is used to model the cumulative workload along the time line for each activity. We input the planned manpower and work duration from the discrete process model to the continuous process model as they are in the identical form. Fig. 3(a) shows the productivity distribution over time from the SD process model. We simulate both models for each activity to compare the cumulative workload at the deadline t. Fig. 3(b) and Fig. 3(c) show that the simulated *Modify design* and *Modify test plan* activities (the solid line) are behind planned schedule (the dash line).

Thus, the process manager decides to adjust planned process attributes in the discrete process model. In this case, as the *Modify design* activity is on the process critical path, its deadline has to be kept unchanged. Since its manpower allocation has been fixed, increasing the designers' productivity is the only option. For *Modify test plan* activity, as it is an non-critical path activity, we can either extend the deadline within its slack days or increase productivity by postponing the activity close to the deadline on the critical path. Assuming we choose the latter solution due to limit budget. The cumulative workload of *Modify test plan* is recalculated by using hybrid simulations, as if the *Modify test plan* were postponed up to 6 days while keeping its planned work duration. Fig. 3(d) shows the adjusted cumulative workload (the solid line) satisfy the planned cumulative workload (the dash line) when postponed 4 days from planned schedule.



(a) Discrete Model of ISPW-6 Using Little-JIL

(b) Earned Value

(c) Defect Dynamics

Fig. 2. (a) Discrete Model, and Continuous Models in (b) Earned Value, (c) in Defect Dynamics

In the above scenario, the process critical path is unchanged. However, when extending activity deadline is the only option to complete the planned workload, the process critical path will be changed due to the interdependencies among activities. For example, if the deadline of *Modify test plan* activity has to be extended, the *Modify unit test package* activity also has to be postponed. Thus, activity level adjustment only cannot guarantee this sub-process to meet its overall deadline. In this condition, we need to simulate in the sub-process level to satisfy the process manager's needs.

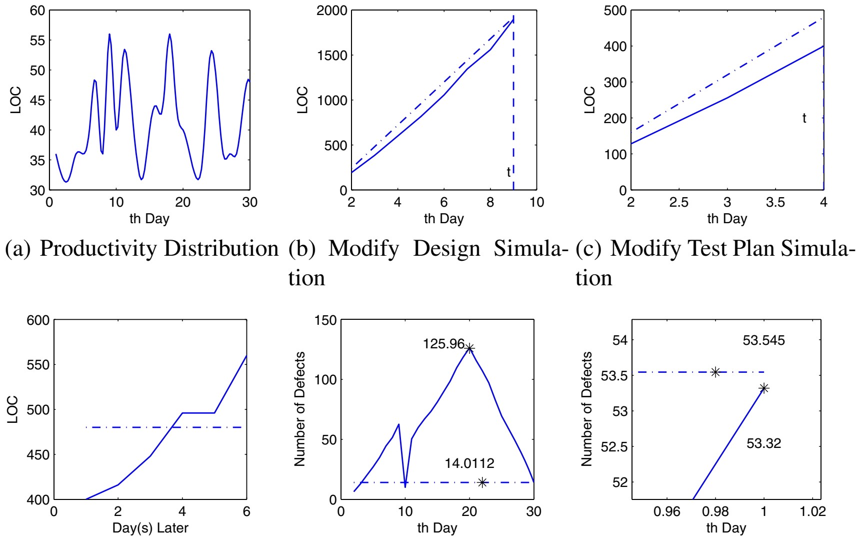
Sub-process Level Hybrid Simulation. To verify the overall deadline at sub-process level, a hybrid simulation can be performed, where the discrete process simulation provides the manpower allocation and activity dependencies to the continuous process simulation. The continuous process model is used to calculate the cumulative work along the time line for the entire sub-process. Based on our simulation result, the planned deadline of this process segment can still be met after the change is made.

However, in the above situation, the quality factors are not taken into account. The process manager needs help from the process engineer to estimate the actual defect number and to verify the deadline can be met while maintaining an acceptable defect density in the product delivery.

4.3 Defect Dynamics: Generation, Detection and Density Control

The process engineer investigates the quality factors of the ISPW-6 process to help process manager verify whether the residual defect density is within the acceptable range, while the original deadline is maintained. His/Her concern is from defect dynamics perspective as shown in Table 2. *Modify design* activity and *Modify code* activity generate defects, while *Design review* activity and *Test unit activity* detect defects.

Sub-process Level Hybrid Simulation. The discrete PML Little-JIL has been used in modeling manpower, workload and the dependencies among activities. The number of residual defects from *Design review* activity is amplified 4 times when they escape into *Modify code* activity due to defect regeneration effect. The review productivity (220-1100 LOC/Man-Day), test productivity (40 LOC/Man-Day), defect generation rate (33 defects/KLOC) and defect detection rate (0.84) are obtained from empirical data. Using these process attributes and time as input to sub-process level SD model as shown in Fig. 2(c), Fig. 3(e) shows the defect distribution (the solid line) along the time line, and there are 14 residual defects (the dash line) at the 30th day deadline. The calculated residual defect density is 2.8 defects/KLOC. Assuming the acceptable residual defect density is 3 defects/KLOC, this hybrid process simulation result verified that the quality of product is acceptable while the planned deadline is met. However, if the acceptable residual defect density is 2.5 defects/KLOC, the quality requirement is not satisfied based on the simulation result. In this case, adjustments to process attributes at the activity level in Little-JIL model is needed to maintain the planned deadline.



(a) Productivity Distribution (b) Modify Design Simulation (c) Modify Test Plan Simulation

(d) Modify Test Plan Reschedule Simulation (e) Residual Defect Number over Time (f) Detected Defects via Design Review

Fig. 3. Hybrid Simulation Results

Activity Level Hybrid Simulation. To decrease the number of residual defects and maintain the planned deadline, we can either reallocate manpower resource to increase defect detection rate in *Design review* activity and *Test unit* activity or decrease defect generation rate in *Modify design* activity and *Modify code* activity. As the manpower in this project is fully used in *Modify design*, *Modify code*, *Modify test plan* and *Test unit*, we could only adjust resource allocation in the *Design review* team to increase the defect detection rate. To satisfy the quality requirement of 2.5 defects/KLOC, additional 0.325 defects have to be detected in *Design review*, which account for additional 1.5 escaped defects at the deadline. By simulating the hybrid model of *Design review* activity with planned defect detection rate and review productivity, the number of detected defects is calculated as 53.22 defects shown in Fig. 3(f) by the solid line at the deadline. Thus, the adjusted *Design review* activity should detects 53.545 defects (the dash line) to meet the overall quality requirement. Feed this into the hybrid process model for the *Design review* activity at the activity level, we found the defect detection rate should be increased to 0.845. Using this as the criteria to reallocate the resources, the project manager can ensure that the overall deadline and quality requirement are both satisfied.

5 Conclusions and Future Work

In this paper, we present a stakeholder-based hybrid process modeling approach to incrementally model a software process using discrete and continuous models. In the case study, we applied this model for validating deadline, adjusting schedule and estimating defect successfully at activity and sub-process level. The hybrid simulation model provides powerful analysis capability by investigating both the static process attributes and process dynamics, especially addressing the different perspectives of stakeholders during the process model life cycle.

The future research can be carried out in two directions: 1) to improve this hybrid modeling scheme at other perspectives, such as software reuse and evolution; 2) to enhance the automation of the hybrid modeling method by automatically identifying the information flow.

References

1. Pall, G.A.: Quality Process Management. Prentice-Hall, Englewood Cliffs (1987)
2. Cass, A.G., Lerner, B.S., Sutton Jr., S.M., McCall, E.K., Wise, A., Osterweil, L.J.: Little-jil/juliette: a process definition language and interpreter. In: ICSE 2000: Proceedings of the 22nd international conference on Software Engineering, pp. 754–757. ACM Press, New York (2000)
3. Osterweil, L.J.: Jil and little-jil process programming languages. In: Gruhn, V. (ed.) EWSPT 1998. LNCS, vol. 1487, p. 152. Springer, Heidelberg (1998)
4. Huang, L., Boehm, B., Hu, H., Ge, J., Lü, J., Qian, C.: Applying the value/petri process to erp software development in china. In: ICSE 2006: Proceedings of the 28th international conference on Software engineering, pp. 502–511. ACM Press, New York (2006)
5. Madachy, R.J., Boehm, B.W.: Software Process Modeling With System Dynamics. John Wiley & Sons, Chichester (2004)

6. Bai, X., Huang, L.: A stakeholder perspective in evaluating process modeling languages and hybrid process simulation. Technical Report (2008)
7. Baldi, M., Gai, S., Jaccheri, M.L., Lago, P.: E3: object-oriented software process model design, pp. 279–292 (1994)
8. Jäger, D., Schleicher, A., Westfechtel, B.: Using uml for software process modeling. SIGSOFT Softw. Eng. Notes 24(6), 91–108 (1999)
9. Gruhn, V.: Managing software processes in the environment melmac. In: SDE 4: Proceedings of the fourth ACM SIGSOFT symposium on Software development environments, pp. 193–205. ACM Press, New York (1990)
10. Bandinelli, S., Fuggetta, A., Ghezzi, C., Lavazza, L.: Spade: an environment for software process analysis, design, and enactment, pp. 223–247 (1994)
11. Sutton Jr., S.M., Osterweil, L.J.: The design of a next-generation process language. SIGSOFT Softw. Eng. Notes 22(6), 142–158 (1997)
12. Zamli, K.Z.: Process modeling languages: A literature review. Malaysian Journal of Computer Science 14(2), 26–37 (2001)
13. An, L., Jeng, J.J.: On developing system dynamics model for business process simulation. In: WSC 2005: Proceedings of the 37th conference on Winter simulation, Winter Simulation Conference, pp. 2068–2077 (2005)
14. Raffo, D.M., Kellner, M.I.: Predicting the impact of potential process changes: A quantitative approach to process modeling. Elements of Software Process Assessment and Improvement (1999)
15. Wakeland, W.W., Martin, R.H., Raffo, D.: Using design of experiments, sensitivity analysis, and hybrid simulation to evaluate changes to a software development process: a case study. Software Process: Improvement and Practice 9(2) (2004)
16. Abdel-Hamid, T., Madnick, S.E.: Software project dynamics: an integrated approach. Prentice-Hall, Inc., Upper Saddle River (1991)
17. Lehman, M., Ramil, J.F.: The impact of feedback in the global software process. Journal of Systems and Software 46, 123–134 (1999)
18. Powell, A., Mander, K., Brown, D.: Strategies for lifecycle concurrency and iteration - a system dynamics approach. Journal of Systems and Software 46(2-3), 151–161 (1999)
19. Tvedt, J.D.: An extensible model for evaluating the impact of process improvements on software development cycle time. PhD thesis, Tempe, AZ, USA (1996)
20. Zhang, H., Jeffery, R., Zhu, L.: Investigating test-and-fix processes of incremental development using hybrid process simulation. In: WoSQ 2008: Proceedings of the 6th international workshop on Software quality, pp. 23–28. ACM Press, New York (2008)
21. Martin, R.H., Raffo, D.: A model of the software development process using both continuous and discrete models. In: Software Process: Improvement and Practice, vol. 5(2-3), pp. 147–157. John Wiley Sons, Chichester (2000)
22. Madachy, R.J.: Software Process Dynamics. Wiley, IEEE Press (2008)
23. Kellner, M.I., Feiler, P.H., Finkelstein, A., Katayama, T., Osterweil, L.J., Penedo, M.H., Rombach, H.D.: Ispw-6 software process example. In: Proceedings of the First International Conference on the Software Process, 1991, pp. 176–186 (1991)

A Process-Oriented Approach for the Optimal Satisficing of Non-Functional Requirements

Christopher Burgess and Aneesh Krishna

School of Computer Science and Software Engineering
University of Wollongong
Northfields Avenue
NSW, 2522, Australia
{cgb06,aneesh}@uow.edu.au

Abstract. In an ever more competitive world, the need for software systems to meet specific quality characteristics becomes increasingly apparent. These quality characteristics, or non-functional requirements, are often contradictory and ambiguous, making them difficult to manage during software development processes. This paper presents a modification of the NFR framework that facilitates the automated discovery of optimal system designs for the satisfaction of non-functional requirements. Just as with the NFR framework, this method can be used at any stage during the software development process in order to aid design decisions. The proposed method introduces the capacity to incorporate both qualitative and quantitative non-functional requirements, as well as the potential to include various cost factors into the optimisation process.

Keywords: Requirements engineering, non-functional requirements, NFR framework, softgoal interdependency ruleset graphs.

1 Introduction

Managing quality characteristics, or in other words non-functional requirements (NFR's), of a software system during the development process is often a difficult challenge. NFR's are often conflicting. In other words, functionalities that help satisfy one NFR may also prevent or damage the satisfying of another. For example, an information system that sends data update logs via email to a system manager may help to improve the accuracy of the system, but obviously quite significantly at the expense of security. NFR's are also often subjective in nature, resulting in different understandings of their meaning. System developers may consider a complex information management system to be efficient if updates are made between 1-2 seconds, however users of that system unaware of technical limitations may not consider such a delay to be very efficient at all.

For these reasons, the need for dedicated tools and methodologies for the management of NFR's becomes apparent. In spite of this, relatively little research has been conducted in the area, perhaps due to the inherent complexities involved. Glinz [7] provides an interesting discussion on the various understandings of NFR's in the RE community, and some of the difficulties involved in managing them. Most proposed

Q. Wang et al. (Eds.): ICSP 2009, LNCS 5543, pp. 293–304, 2009.

methods have taken a quantitative, product-oriented approach, where the satisfying of the NFR's is measured quantitatively after analysing a functional version of the system. In [8], NFR's are handled by the creation of independent teams who carry out separate inspection tasks, while in [9], independent teams carry out the same inspection task in order to attain a more thorough detection of faults. On the other hand, in [1], a management perspective is taken, whereby the software development process itself is evaluated through various models and metrics in order that quality software products may result.

In what is perhaps the most comprehensive method for the management of NFR's, Chung et al. [2], [5] have developed the NFR framework. The NFR framework builds on ideas from goal-based AI problem solving techniques and qualitative reasoning to deliver a qualitative, process-driven method for the management of NFR's, able to be used at any stage during the development process to aid design decisions. The NFR framework is qualitative in the sense that the satisfaction of NFR's is spoken of in natural language terms. As such, NFR's are represented as softgoals, rather than goals, in order to denote the notion that NFR's often do not have clearly defined satisfaction criteria. These softgoals are said to be satisficed [10] if they are considered to have been met to an acceptable level, rather than saying that they are satisfied with full certainty.

The basic idea behind the NFR framework is the *"incremental and interactive construction, elaboration, analysis, and revision of a softgoal interdependency graph (SIG)"* [5, p.17], which displays the NFR softgoals to be satisficed, as well as operationalizing softgoals, representing potential system functionalities. NFR softgoals and operationalizing softgoals can be refined and decomposed into more fine-grained softgoals in order to remove ambiguity. The nature and level to which operationalizing softgoals contribute to the satisficing of NFR softgoals are denoted via interdependency links. An example of a SIG can be seen in Figure 1.

The developer may then use the SIG to aid in system design by labelling bottom-level operationalizing softgoals as being either satisficed or denied, and then following a semi-automated label propagation procedure following the defined interdependencies and decomposition relationships, until all the top-level NFR softgoals have been assigned labels denoting the degree to which the selected operationalizing softgoals satisfice them. This label propagation procedure is guided by assigning labels to the interdependency links; which are one of EQUAL, MAKES, SOME+, HELPS, UNKNOWN, HURTS, SOME-, or BREAKS. MAKES is used whenever satisficing an operationalizing softgoal also results in the satisficing of an NFR softgoal. The HELPS label indicates that satisficing the operationalizing softgoal helps to satisfice an NFR softgoal, but is insufficient to fully satisfice the NFR softgoal alone. SOME+ is used whenever it is unclear if the interdependency ought to be MAKES or HELPS. The NFR softgoals themselves may be labelled as either Satisficed (S), Weakly Satisficed (W+), Unknown (U), Conflict (C), Weakly Denied (W-), or Denied (D). A full listing of the label to propagate to an NFR softgoal given a label for the operationalizing softgoal that contributes to it, as well as the connecting interdependency link label, is given in Table 1.

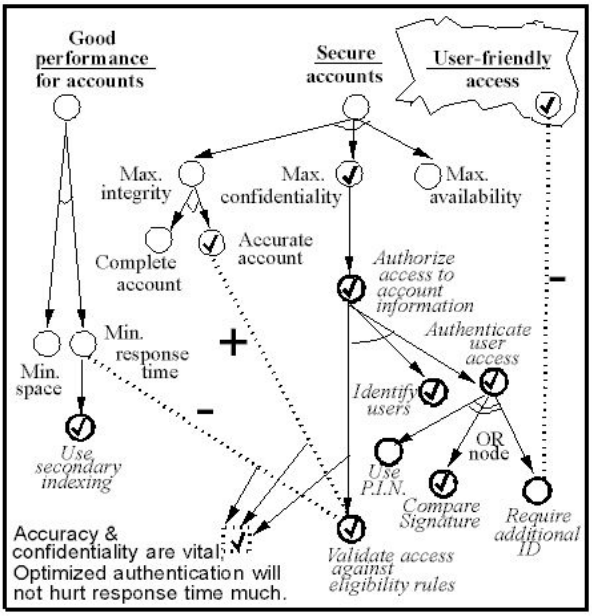


Fig. 1. An example SIG for a credit card system. The *Secure Accounts* NFR softgoal is decomposed into three softgoals for *Max. Integrity, Max. Confidentiality, and Max. Availability*. Interdependencies are shown between the *Validate access against elibility rules* operationalizing softgoal and the *Min. response time* and *Accurate account* NFR softgoals.

Table 1. Label to be propagated to a softgoal given a contributing softgoal label and a connecting interdependency link label

	BREAKS	**SOME+**	**HURTS**	**UNKNOWN**
D	W+	W+	W+	U
C	C	C	C	U
U	U	U	U	U
S	D	W-	W-	U
	HELPS	**SOME+**	**MAKES**	**EQUAL**
D	W-	W-	D	X
C	C	C	C	C
U	U	U	U	U
S	W+	W+	S	S

A single propagation of a label from one softgoal to the next via an interdependency can be done automatically according to the rules in Table 1. This, however, may result in conflicts whenever multiple interdependencies propagate opposing labels to the same NFR softgoal. These conflicts are resolved by the developer manually, by selecting the appropriate label after taking into consideration all the labels that have been propagated to it.

The NFR framework also includes a catalogue of *methods*, which defines decomposition templates to aid developers in the process of refining softgoals into more refined softgoals. Known interdependencies are also stored in a catalogue of correlation rules, which can aid the developer in constructing the SIG, whereby the addition of an NFR or operationalizing softgoal triggers the detection of interdependencies between the newly added softgoal and existing softgoals in the SIG. Both of these catalogues help to reduce SIG development time, making the NFR framework a convenient tool to use for the management of NFR's.

The proposed method in this paper defines a new structure for SIG's, called Softgoal Interdependency Ruleset Graphs (SIRG's), which no longer requires developer intervention for the propagation of labels. This introduces the capacity for determining optimal sets of operationalizing softgoals automatically, rather than by trial-and-error at the hand of the developer. NFR goals have also been introduced, which allow for NFR's to be labelled quantitatively, rather than qualitatively as in the case of NFR softgoals. Both NFR softgoals and goals can be combined within the same SIRG. There is also the potential for various cost factors to be included in the optimisation procedure, such as development time and cost, maintenance cost, and development difficulty or risk. A tool, the *NFR Optimiser*, has also been created which implements the proposed method, of which a small illustration has been outlined.

In Section 2 of the paper, SIRG's and their features are explained, outlining the methods by which labels are propagated in order to allow for automated analysis. In Section 3, optimisation of SIRG's is discussed, principally in relation to the evaluation of a SIRG given particular labels for the top-level NFR softgoals. The *NFR Optimiser* is introduced in Section 4, including a brief example of its use. Section 5 gives a brief comparison of SIG's and SIRG's and their relative advantages and disadvantages, while Section 6 concludes the paper, and outlines some areas for further work.

2 Softgoal Interdependency Ruleset Graphs

Softgoal Interdependency Ruleset Graphs (SIRG's) are a new graph structure that has been developed which may lead to automated propagation of labels, thereby creating the possibility for their optimisation without developer input. SIRG's keep many of the same concepts as those that are found in SIG's and the NFR framework. The main difference is the introduction of a new node type, called Interdependency Rulesets (IR's), which contain all the information necessary to propagate labels throughout the graph, removing the need for developer input during label propagation. The core features of SIRG's are as follows.

Softgoals. Just as with SIG's, NFR's, as well as potential system functionalities, are represented as softgoals to be achieved. In SIRG's however, NFR's may also represented as goals in situations where the satisfaction of the NFR may be denoted in quantitative terms (i.e. percentages).

Labels. All softgoals can be assigned labels indicating the degree to which they are satisficed. NFR softgoals can be given one of the labels Strongly Satisficed, Moderately Satisficed, Weakly Satisficed, Indeterminate, Weakly Denied, Moderately Denied, or Strongly Denied, while NFR goals are given percentages rather than labels, ranging from 100% to -100%. Operationalizing softgoals, on the other hand, can be labelled as either Satisficed or Denied, where satisficed operationalizing softgoals correspond to accepted functionalities for the software system in question.

Relationships. Softgoals are connected by different types of relationships, which describe the effect that softgoals have on the satisficing of other softgoals in the graph. These relationships fall into two main types: decompositions and interdependencies. Decompositions occur between softgoals of the same type, ie. between NFR softgoals or goals and other NFR softgoals and goals, and between operationalizing softgoals and other operationalizing softgoals. These are useful for refining softgoals into more fine-grained softgoals, which is particularly helpful for reducing the ambiguity that often surrounds NFR's. Examples of this can be seen in Figure 1, such as the refinement of 'Good performance' into 'Min. space' and 'Min. response time'. Interdependencies occur between operationalizing softgoals and NFR softgoals and goals, and are used to denote the impact that an operationalizing softgoal has on the satisficing of the NFR softgoal or goal.

Decompositions themselves come in two forms; AND and OR decompositions. AND decompositions are used whenever the satisficing of the parent softgoal generally requires the satisficing of all the child softgoals. Whereas OR decompositions are used when the satisficing of the parent softgoal only requires that one of the child softgoals is satisficed.

Interdependency Rulesets (IR's). The main feature of SIRG's that allows for automated propagation of labels (to some extent) is the introduction of a new node type: Interdependency Rulesets (IR's). IR's contain all the information required to propagate labels throughout the SIRG, removing the need for developer intervention.

IR's may be used in two separate scenarios. They are primarily used to determine the label to propagate amongst interdependencies between a set of operationalizing softgoals and a single NFR softgoal or goal, although they may also be used to create custom decompositions, when the standard AND or OR decompositions do not suffice.

Each NFR softgoal or goal that receives contributions from operationalizing softgoals via interdependency links has a single IR. The IR contains simple, if-then rules, where each rule defines the label to be propagated to the NFR softgoal, given a particular combination of operationalizing softgoal labels. To illustrate, Figure 2 shows an NFR softgoal for the accuracy of high-spending Gold Accounts in a credit card system. Beneath this softgoal is its associated IR, displayed with the double square icon, which stores all the rules for propagating labels to the NFR softgoal. Two operationalizing softgoals are also shown, with interdependency links drawn to the IR.

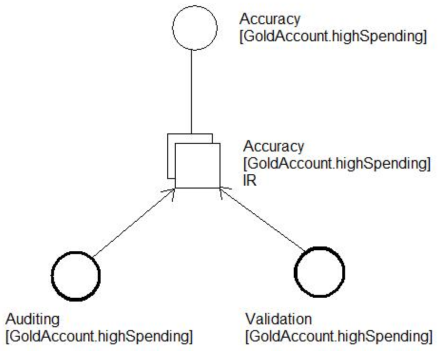

Accuracy
[GoldAccount.highSpending]
Accuracy
[GoldAccount.highSpending]
IR
Auditing
[GoldAccount.highSpending]
Validation
[GoldAccount.highSpending]

Fig. 2. Example of an Interdependency Ruleset (IR)

The if-then rules in the IR may be represented in table form, an example of which is shown in Table 2. Given these rules, the required label for the accuracy NFR softgoal can be determined regardless of the labelling of the operationalizing softgoals.

Table 2. Example rules for IR in Figure 2

Auditing Label	Validation Label	Accuracy Label
Satisficed	Satisficed	Strongly Satisficed
Satisficed	Denied	Weakly Denied
Denied	Satisficed	Weakly Satisficed
Denied	Denied	Strongly Denied

IR's may also be used to define custom decompositions where the standard AND or OR decomposition relationship does not suffice. An example of a custom decomposition is given in Figure 3.

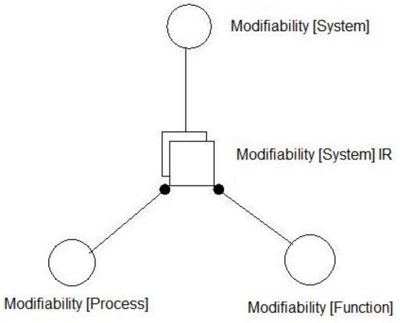


Fig. 3. Example custom decomposition for a system modifiability softgoal

Note that unlike with SIG's, interdependency links are not themselves assigned any label, as all the information required for propagating labels is contained within the IR itself. It is permissible, however, to display a small positive or negative sign alongside links to indicate the general nature of the interdependency to aid visualisation.

Label Propagation. Given the structure for SIRG's outlined thus far, there are several possible scenarios for label propagation. Firstly, the standard AND and OR decomposition relationships are governed by simple rules. For AND decompositions, the minimum (least satisficed) label amongst the child softgoals is assigned to the parent, while for the OR decomposition, the maximum (most satisficed) label is assigned to the parent. Note that it is possible to have NFR goals as children to NFR softgoals. In this case, NFR goal percentages are first mapped to an NFR softgoal label according to the thresholds outlined in Table 3, before the minimum or maximum rule is applied. At this stage, these thresholds have been chosen purely on an intuitive basis. Further experimentation and case study analysis may result in future adjustment of these thresholds.

Table 3. Mappings of NFR goal percentages to NFR softgoal labels

NFR goal percentage	NFR softgoal label
>= 80%	Strongly Satisficed
50 - 79%	Moderately Satisficed
20 - 49%	Weakly Satisficed
-19 - 19%	Indeterminate
-49 - -20%	Weakly Denied
-79 - -50%	Moderately Denied
<= -80%	Strongly Denied

In all other situations, softgoal relationships, whether they are interdependencies or custom decompositions, are governed by an IR node which specifies the label to propagate given any possible combination of child softgoal labels.

3 SIRG Optimisation

Given that the required information and rules for label propagation are contained completely within the SIRG itself, the possibility arises for finding the optimal set of bottom-level operationalizing softgoals automatically. Various combinations of satisficed and denied labels for the bottom-level operationalizing softgoals may be tried, and an overall evaluation score of the satisficing of the top-level NFR softgoals determined for each. Thus, the set of satisficed operationalizing softgoals which produces the best evaluation score corresponds to an optimal system design that most satisfices the system's NFR's.

The actual optimisation method used is open to question. An exhaustive search, which tests all possible combinations, may certainly be plausible if the number of softgoals remains sufficiently small, however further experimentation is required in order to determine at what point the size of the SIRG becomes too large for an exhaustive search to be feasible. Obviously, if an exhaustive search is feasible, then the optimal solution is guaranteed to be found. Early tests have shown that genetic algorithms may also be an effective and efficient method for optimising SIRG's.

SIRG Evaluation. In order to determine an evaluation score given a particular set of top-level NFR softgoal labels and NFR goal percentages, combining both the qualitative and quantitative information contained therein, the following method is proposed.

The evaluation method involves the construction of a histogram, where each top-level NFR softgoal and goal contributes an equal volume, *v*. The histogram ranges from -100 to 100 to match the percentage range of NFR goals, and for each top-level NFR goal, a bar *v* high is added to the histogram bin at its percentage value. NFR softgoals also contribute the volume *v* to the histogram, but over a range of histogram bins that correspond to the mappings in Table 3 for the softgoals label *n*. Thus, a bar v/r_n high is added to each bin within the range corresponding to the label *n*, where r_n is the number of bins within that range.

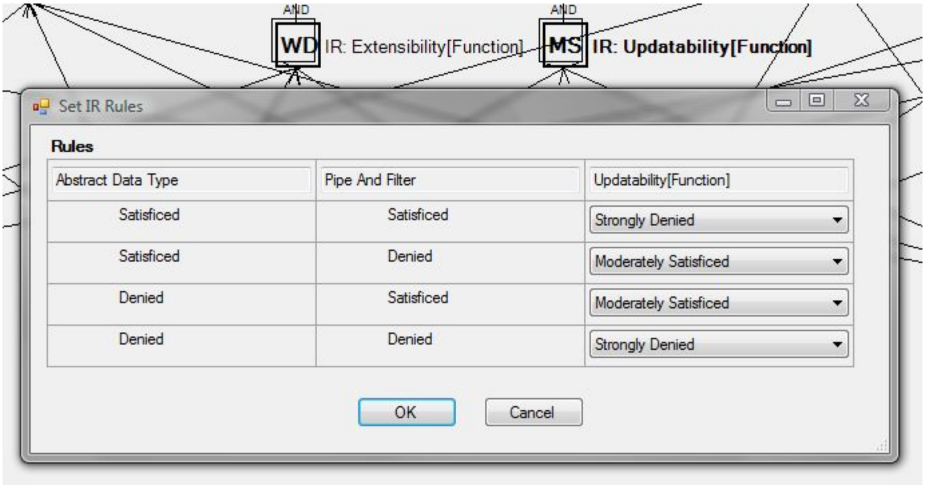
Once the histogram has been constructed for all the top-level NFR softgoals and goals, an overall evaluation score, *s*, for the histogram can be calculated using (1), where h_i is the histogram value at bin *i*, and *k* is a proportionality factor that controls the weight of the histogram value as a function of *i*. If $k = 1$, the weights are linearly proportional to *i*, while higher values result in greater weight being given to values at either end of the histogram.

$$s = \sum_{i=1}^{100} h_i (\frac{i}{100})^k - \sum_{i=-100}^{-1} h_i (\frac{-i}{100})^k . \tag{1}$$

4 The NFR Optimiser

To test the use of SIRG's in analysing NFR's, a tool has built, simply called *NFR Optimiser*. This tool provides the ability to create SIRG's, define IR's, and determine optimal sets of operationalizing softgoals accordingly. The *NFR Optimiser* uses an exhaustive search procedure in order to optimise the SIRG. To help illustrate, a simple example of a KWIC (Keyword in Context) system, analysed in [5, ch.13], was optimised using the *NFR Optimiser*. This example considers four main NFR's: comprehensibility, modifiability, performance, and reusability. Potential system designs fall into four main categories: Shared Data, Abstract Data Type, Implicit Invocation, and Pipes and Filters. The SIRG representation of this problem domain, taken from the *NFR Optimiser*, including abbreviations for all softgoal labels, is given in Appendix A. Note that the *NFR Optimiser* also allows for priority levels to be set for NFR softgoals, and these are denoted by the single and double exclamation marks.

Given the IR rules defined for this example, the Abstract Data Type system was selected as the best solution for the KWIC problem. In this case, given that the set of operationalizing softgoals are mutually exclusive, all IR rules where more than one of the operationalizing softgoals are satisficed were given the Strongly Denied label in order to prevent multiple operationalizing softgoals from being selected. In future, it is intended that an option to specifically create sets of mutually exclusive operationalizing softgoals will be included.



Abstract Data Type	Pipe And Filter	Updatability[Function]
Satisficed	Satisficed	Strongly Denied
Satisficed	Denied	Moderately Satisficed
Denied	Satisficed	Moderately Satisficed
Denied	Denied	Strongly Denied

Fig. 4. Screenshot of the rules defined for the Updatability[Function] IR

An example set of rules for an IR is given in Figure 4, where the NFR softgoal Updatability[Function] receives interdependency links from both Abstract Data Type and Pipes and Filter.

5 SIG's and SIRG's: A Comparison

Procedurally, the use of SIRG's in the management of NFR's differs little to that of SIG's in the NFR framework. It remains process-driven, and the capacity for the creation of catalogues of decomposition templates and correlation rules for automatically detecting interdependencies remains valid. Just as with SIG's, SIRG's may be developed and edited throughout the development process to model NFR's and aid in making design decisions to help ensure that NFR's are being met.

The use of SIRG's provides a means to automatically determine the optimal system design for meeting NFR's, while on the other hand, determining optimal system designs using SIG's is largely a manual process. Performing this analysis manually may be sufficient for small systems, where the number of NFR's and the range of system functionalities considered are relatively small. As such, using SIRG's for small systems may provide little benefit. However, as the size and complexity of the system increases, and consequently the number of softgoals and interdependencies, the automated optimisation of SIRG's may prove advantageous. We need further experimentation to test effectiveness of SIRG's on larger case studies though.

One advantage that SIRG's do have is the ability to combine both qualitative and quantitative information. As the software system evolves during the development process, it may be possible to convert NFR softgoals to NFR goals as more quantifiable information is gained as a result of system testing, further improving the accuracy of the analysis.

6 Conclusions and Further Work

This paper has presented an adaptation of the NFR framework, including an implementation of the method within the tool *NFR Optimiser*. Softgoal Interdependency Ruleset Graphs (SIRG's) have been introduced, which is a new method for graphically representing NFR's, system functionalities, and the relationships between them, designed to facilitate automated optimisation. Use of the *NFR Optimiser* during the software development process has the potential to aid developers in managing NFR's, intended as a process-driven method for automatically determining the optimal set of system functionalities that best meet a given set of NFR's. More case study analysis is required, however, in order to more adequately determine the tools usefulness.

The proposed method, and the tool *NFR Optimiser*, are in its early stages, and there are still significant areas of improvement possible. While briefly introduced in the *NFR Optimiser* illustration, applying priority levels to NFR softgoals is an area requiring further attention. In terms of SIRG evaluation, in essence priority levels are handled by weighting the total volume v applied to the histogram, however the most appropriate number of priority levels to be used, and their relative weights remains an open question. Another significant area for future work is the introduction of various cost factors, such as development time and cost, maintenance cost, and development difficulty or risk, for each operationalizing softgoal. This would then turn SIRG optimisation into a multi-objective optimisation problem, and would allow for the consideration of both the satisficing of NFR's and the minimisation of system costs to be considered simultaneously.

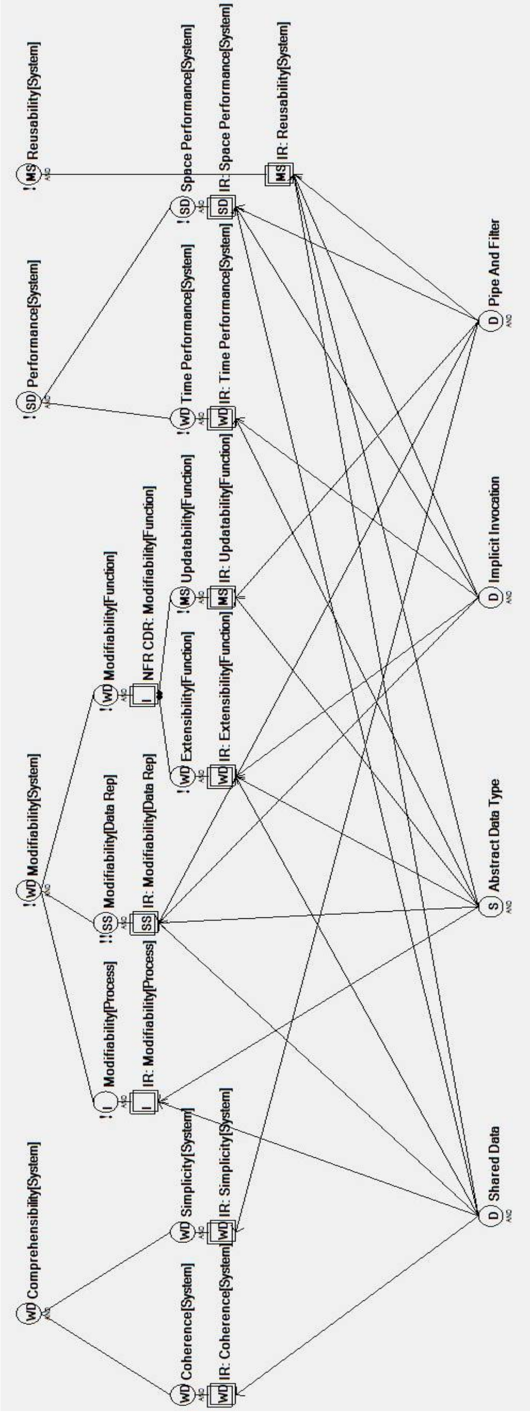
One major benefit of the NFR framework is the inclusion of catalogues of known interdependencies, and decomposition templates, which aid the developer in creating SIG's efficiently and accurately. There is also the potential for SIRG's to be augmented by similar catalogues. In fact, it may very well be possible for SIRG's to utilise exactly the same catalogues as those included with the NFR framework, though that remains to be seen. Further to this, common IR's, where the required rules for a given set of operationalizing softgoals are already specified, potentially may also be added to the interdependency catalogue to aid the developer further.

References

1. Basili, V.R., Musa, J.D.: The Future Engineering of Software: A Management Perspective. IEEE Computer 24(9), 90–96 (1991)
2. Chung, L.: Representing and Using Non-Functional Requirements: A Process Oriented Approach. Ph.D. Thesis, Dept. of Comp. Sci., Univ. of Toronto (1993)
3. Chung, L., Nixon, B.A.: Dealing with Non-Functional Requirements: Three Experimental Studies of a Process-Oriented Approach. In: 17th IEEE International Conference on Software Engineering, Seattle, pp. 25–37 (1995)

4. Chung, L., Nixon, B.A., Yu, E.: Using Quality Requirements to Systematically Develop Quality Software. In: Proc. 4th International Conference on Software Quality, McLean, VA, USA (1994)
5. Chung, L., Nixon, B.A., Yu, E., Mylopoulos, J.: Non-Functional Requirements in Software Engineering. Kluwer Academic Publishing, Dordrecht (2000)
6. Glinz, M.: On Non-Functional Requirements. In: 15th IEEE International Requirements Engineering Conference (2007)
7. Linger, R.C.: Cleanroom Software Engineering for Zero-Defect Software. In: Proc. 15th International Conference on Software Engineering, Baltimore, MD, USA, pp. 2–13 (1993)
8. Schneider, G.M., Martin, J., Tsai, W.T.: An Experimental Study of Fault Detection in User Requirements Documents. ACM Transactions on Software Engineering and Methodology 1(2), 188–204 (1992)
9. Simon, H.A.: The Sciences of the Artificial, 2nd edn. MIT Press, Cambridge (1981)

Appendix A: *NFR Optimiser's* SIRG for KWIC Example


! WD Comprehensibility[System]
AND
WD Coherence[System]
WD IR: Coherence[System]
WD Simplicity[System]
WD IR: Simplicity[System]
! I Modifiability[Process]
I IR: Modifiability[Process]
! WD Modifiability[System]
!! SS Modifiability[Data Rep]
SS IR: Modifiability[Data Rep]
! WD Modifiability[Function]
I NFR CDR: Modifiability[Function]
! WD Extensibility[Function]
WD IR: Extensibility[Function]
! MS Updatability[Function]
MS IR: Updatability[Function]
! SD Performance[System]
! WD Time Performance[System]
WD IR: Time Performance[System]
! SD Space Performance[System]
SD IR: Space Performance[System]
! MS Reusability[System]
MS IR: Reusability[System]
D Shared Data
S Abstract Data Type
D Implicit Invocation
D Pipe And Filter

A Pattern for Modeling Rework in Software Development Processes

Aaron G. Cass[1], Leon J. Osterweil[2], and Alexander Wise[2]

[1] Department of Computer Science
Union College
Schenectady, NY 12308
cassa@union.edu

[2] Department of Computer Science
University of Massachusetts Amherst
Amherst, MA 01003
{ljo,wise}@cs.umass.edu

Abstract. It is usual for work completed at one point in a software development process to be revisited, or reworked, at a later point. Such rework is informally understood, but if we hope to support reasoning about, and partial automation of, software development processes, rework be more formally understood. In our experience in designing formalized processes in software development and other domains, we have noticed a recurring process pattern that can be used to model rework quite successfully. This paper presents that pattern, which models rework as procedure invocation in a context that is carefully constructed and managed. We present some scenarios drawn from software engineering in which rework occurs. The paper presents rigorously defined models of these scenarios, and demonstrates the applicability of the pattern in constructing these models.

1 Introduction

Rework, the activity of reconsidering and modifying an earlier decision, is a feature common in software engineering and other creative processes. While reconsideration may be relatively straightforward, modifying earlier choices can be far more complicated, as it typically entails reconsidering and modifying other choices that had been made subsequent to the choice being modified. These subsequent modifications can then lead to still further reconsiderations and modifications, potentially creating a daunting collection of reconsiderations and modifications that can leave the participants confused and increasingly incapable of keeping the rework activity under control.

In software development, for example, it is not uncommon to find that a design under consideration is becoming unduly complicated because of requirements or architectural decisions that had been made previously. Often this causes the designer to revisit the requirements, for example, to see if they are needlessly restrictive, and to revisit either the requirements or architecture to see if they are, for example, inadventently inconsistent either internally or with each other. These revisitations often lead to changes, which in turn lead to others. Requirements changes then cause reconsideration of architecture decisions, which can cause the architecture to be inconsistent with the changed

Q. Wang et al. (Eds.): ICSP 2009, LNCS 5543, pp. 305–316, 2009.

requirements. Architecture changes can then require revisiting design and requirements decisions. The full effect of a single change is often referred to as the "ripple effect", and it can be highly disruptive to development processes that might otherwise seem to be orderly and systematic. Thus, the simplicity of a pure "waterfall" process is typically augmented by back-edges that denote iteration. But these iterations are typically driven by changes and cause other changes. Thus what may appear to be simple iterations are actually iterations needed to effect rework changes. Indeed it has been estimated that a very large percentage of the total effort in a complex development project is devoted to rework.

Because of the prevalence of rework, it seems important to better understand it. A clear understanding of the nature of rework could lead to stronger support for the activity, through carefully tailored tools and automated aids. Such an understanding might also help control the propagation of the ripple effect of modifications resulting from rework. It might also lead to better understanding of the relationship between rework and traditional development, perhaps helping to identify common subactivities and a smoother integration of these two dominant components of creative work.

In earlier work [1] we suggested that rework can be modeled as re-invocation of one or more development activities that had been carried out previously, but now must be carried out in a *context* different from the original execution, where a context is any aspect of the process step's dynamic, run-time environment (e.g. input parameters, resources, or personnel employed) that can cause the step to be carried out differently or produce a different result. In that approach, we used procedure invocation semantics to view rework as the invocation of a procedure in a context. So, instead of simply saying that rework is "going back" to a previously-executed development stage, we say that rework is a re-invocation of some activity, activities, or subphase(s) of that previously-executed stage. Invocation semantics make it clear where the rework activity gets its data from and where any output data is passed – down and up (respectively) the procedure invocation trace. This earlier work left largely unexplored, however, the specific details of how this rework context information was gathered, passed to, and used by the rework activity itself. In this paper we present examples of rework, define a rework process pattern based on re-invocation in a managed context, and use this pattern to specify rework in some software development activities.

2 Motivating Examples of Rework

As suggested above, rework happens when it becomes clear that a previous decision has become problematic, often because it has come into conflict with subsequent decisions. In such cases, rework is undertaken to resolve the conflict. In this section we present several concrete examples of rework that we have encountered, and which seem amenable to solution using the pattern presented later in this paper. Though we have found rework in dispute resolution [2], scientific data processing [3,4], health care delivery [5], and other domains, the examples here are drawn from software engineering.

Although there is a lack of agreement about the exact way to develop software, there is little disagreement that a finished product consists of a set of different types of artifacts, usually including a specification of requirements, a design meeting those requirements, executable code, and evidence that the code satisfies the requirements. Ultimately, these artifacts must be acceptably consistent, both internally and with each other. For example, the design should be shown to specify an approach that enables the requirements to be met, and the executable code should be shown to be a correct implementation of the design.

While all of these dimensions of consistency should be achieved by the end of a development project, there is no expectation that all of these consistencies will be achieved easily or straightforwardly. For example, developers expect that initial design decisions will be inconsistent with requirements, initial code might not correctly implement designs, and that executing code may make clear the unreasonableness of requirements specifications. Indeed, the process of developing a finished product inherently entails the more or less continuous reevaluation and reconsideration of all prior decisions.

For example, a requirements specification may be reconsidered because it is apparently inconsistent with other requirements, because subsequent design efforts are complicated by the requirement, or because code seems unable to satisfy the requirement. Identifying the inconsistency is what we will call a **triggering event**. In such cases, the nature of the conflict is a key part of the **context** under which developers re-evaluate and possibly re-produce the problematic requirement. We view this re-evaluation and re-production as re-instantiation of the activity that produced the faulty requirement in a new context, with different input parameters. Re-instantiation in the new context leads to what we will call **re-invocation** of the decision process, resulting in rebinding of the results of the execution. The re-invocation may or may not result in a modification of the requirement. In either case, execution resumes where it left off, at the site of the triggering event. This may entail re-evaluating the triggering condition, and that might trigger further rework. In software development, a modification of an artifact can wind up triggering a long sequence of rework activities, which can entail multiple reconsiderations of a decision about a single artifact. The fact that there are additions and changes to the context for subsequent reconsiderations of the same artifact improves the chance that previous experiences will inhibit making the same decision multiple times, reducing the chances of unproductive loops in the process.

3 A Pattern of Managed Rework

The previous examples suggest a pattern of rework, which we will now define using the vocabulary of Gamma et al. [6]. Fig. 1 shows the structure of the pattern using a UML activity diagram [7].

3.1 Applicability

- Use the pattern if internal consistency of work products must be maintained and one wants to handle the inconsistencies that arise before continuing.
- Use the pattern if rework will trigger a long, complex sequence of consequences, whose management will be facilitated by an at least semi-automated process.

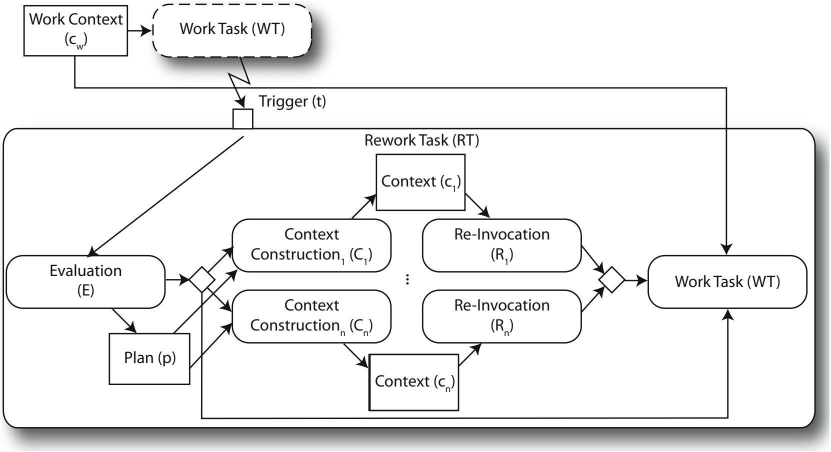


Fig. 1. Structure of the pattern

3.2 Participants

Work task (WT)**.** The process activity in progress when the need for rework is noticed.

Trigger $(t = \{e, i\})$**.** The trigger is a message that is sent in response to the identification of an inconsistency that seems to require rework. The trigger should identify the entity to be reworked e, and the inconsistency detected i.

Rework task $(RT = \{E, \{(C_1, R_1), \dots (C_n, R_n)\}\})$**.** The process model must specify how to respond to each trigger by specifying the following parts:

Evaluation $(E(t) \rightarrow p)$**.** An activity to be carried out to evaluate the trigger t in order to create a plan p that specifies what, if any, response is required as a response to the trigger. Process models can omit an explicit evaluation activity if there is only a single possible plan that could result from a trigger, in which case $p = t$.

Context Construction $(C_j(p) \rightarrow c_j)$**.** An activity to create an appropriate calling context c_j for the re-invocation.

Re-invocation $(R_j(c_j))$**.** The re-invocation of a previously executed activity.

3.3 Collaborations

- A process model using to the pattern must define a triggering mechanism. Typically, the trigger is the result of a checking activity and the trigger is represented as an exception object – e.g., a design review might check the internal consistency of a design and throw an exception if the review fails.
- When trigger t is fired, work task WT is suspended and the rework task RT begins.
- Rework begins with the evaluation of trigger t by E to determine whether re-invocation of a previously-executed activity is needed. Based on the trigger, Evaluation will create plan p, which may consist of a choice among several different re-invocations to address the trigger. Evaluation may involve human effort, or the

activity may be (at least partially) automated. If re-invocation is needed, an appropriate context c_j for re-invocation R_j is constructed by context construction C_j. Re-invocation R_j can then be undertaken, using parameters provided by context c_j. Once re-invocation is complete, rework task RT is complete, and the process proceeds where it left off by re-invoking work task WT in its original context c_w (i.e. with its original parameters).

3.4 Consequences

- A precise model describes both the context in which rework occurs and how to proceed after the rework has been completed.
- Processes using rework implemented with this pattern may be executable. Because rework is explicit in the process model, such executable processes could be used to monitor how well an activity is progressing.
- Activities that may be carried out in rework contexts must be designed to be carried out in all possible re-invocation contexts. For example, a requirements specification activity within a development process must be designed to allow for modification of the requirements during phases other than the initial requirement specification.

3.5 Related Patterns

- When the immediate reworking of an entity is not desired, the exception handling pattern Deferred Compensation may be used [8]. This pattern breaks rework tasks into two disjoint activities – one contains the evaluation activity, while the other, the deferred activity, contains context construction and re-invocation activities.
- Object Derivation [9] offers an approach in which requests for inconsistent objects serves as a trigger for *backward chaining*.
- Observer [6] may be used as a vehicle for creating triggers when reworking an item can result in the need for *forward chaining*.
- Task Deferral [10] is another mechanism for triggering rework. In this pattern, the availability of data is itself a trigger for the forward chain.

4 Managing the Context

It is the context in which the re-invocation occurs that differentiates rework from simple procedure invocation. The context defines the entities to be modified, the information available to support this modification, and the constraints placed on the re-invoked activity. Therefore, the re-invocation context must define the binding of objects to both the in- and out-parameters of the re-invocation, ensure that any appropriate constraints are enforced during the re-invocation, and specify the response to any events that arise while the re-invocation is in progress. Formally, we define the context as:

- a clear designation of the entity that is to be the subject of the rework activity.
- a set of bindings between the formal parameters to R_j and the actual arguments in the plan $p : \{(f_1, a_1), \dots (f_n, a_n)\}$.
- a set of constraints B that are in force during the re-invocation.
- a set of handlers H for any signals s that may occur during the re-invocation.

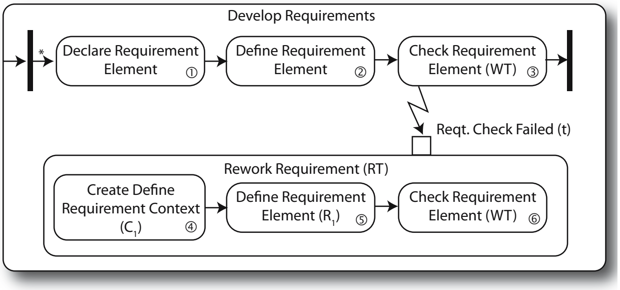

Develop Requirements
*
Declare Requirement Element ①
Define Requirement Element ②
Check Requirement Element (WT) ③
Reqt. Check Failed (t)
Rework Requirement (RT)
Create Define Requirement Context (C_1) ④
Define Requirement Element (R_1) ⑤
Check Requirement Element (WT) ⑥

Fig. 2. Control flow in the requirements development process

The set of bindings in the context specifies the information available, and the destination of entities that are created during the re-invocation. When creating this binding, the difference between pass-by-reference and pass-by-value is particularly important. Since rework often involves the modification of existing entities, is is usual to pass these entities by reference. We note, however, that the derivation history [9,3,4] of these entities is often particularly important in the creation of a rework plan as that history is often necessary in order to guide the re-invocation away from repetition of choices that have previously been shown to have led to later problems.

```
try
    element ← Declare Requirement Element
        (informal requirements)                       ①
    Define Requirement Element (element)              ②
    Check Requirement Element (work context)          ③
catch failure : Reqt. Check Failed
    context ← Create Define Requirement
        Context (failure)                             ④
    Define Requirement Element (context)              ⑤
    Check Requirement Element (work context)          ⑥
end try
```

Fig. 3. Pseudo-code for the requirements development process. Step numbers correspond to those in Fig. 2.

In our experience, exceptions are often used as triggers of rework, and as seen in the examples provided in Section 5, changes made in later phases of a larger activity often create inconsistencies or problems that are detected as the violations of constraints that then generate exceptions that in turn initiate rework sequences that "ripple" through the larger activity. The pattern we define here allows the set of handlers H to control this rippling by treating re-invocations as "work tasks" from which triggers may be emitted, each of which associates a set of parameter bindings, specifications of which entities are to be reworked, and other components of the context for the re-invocation.

5 Examples Using the Pattern

We begin with rework in requirements specification, shown in Figures 2 and 3. This activity consists of the parallel creation of a set of Requirement Elements, each of which consists of a Requirement Specification Declaration (created by Declare Requirement Element) and a Requirement Specification Definition (created by Define Requirement Element). Each Requirement Element is reviewed in Check Requirement Element, and reworked if the review indicates any deficiencies.

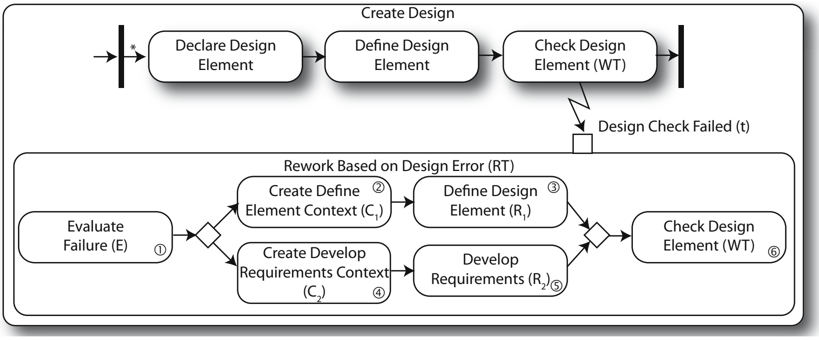

Create Design
*
Declare Design Element
Define Design Element
Check Design Element (WT)
Design Check Failed (t)
Rework Based on Design Error (RT)
Evaluate Failure (E) ①
Create Define Element Context (C₁) ②
Define Design Element (R₁) ③
Create Develop Requirements Context (C₂) ④
Develop Requirements (R₂) ⑤
Check Design Element (WT) ⑥

Fig. 4. Control flow in the design process

The failure of Check Requirement Element serves as the trigger. As the Requirement Element must always be reworked if its review fails, this use of the rework pattern contains no evaluation activity – responding to the trigger immediately results in the creation of an appropriate context. In this example, we assume that the failure indicates the need to rework a specific Requirement Specification Definition whose identity is passed as part of the trigger. The rework then begins with creation of the appropriate context by Create Define Requirement Context, then proceeds with the re-invocation of Define Requirements Element, and then resumption of the execution of Check Requirement Element in the context from which the initiating trigger occurred.

```
try
    element ← Declare Design Element
        (requirement)
    Define Design Element (element)
    Check Design Element (work context)
catch failure : Design Check Failed
    plan ← EvaluateFailure(failure)                    ①
    if plan is rework the design then
        $context_1$ ← Create Define Element
            Context(plan)                              ②
        Define Design Element ($context_1$)              ③
    else rework the requirements
        $context_2$ ← Create Develop Requirements
            Context(plan)                              ④
        Develop Requirements($context_2$)                ⑤
    end if
    Check Design Element(work context)                 ⑥
end try
```

Fig. 5. Pseudo-code for the design process. Step numbers correspond to those in Fig. 4.

In reinvoking Define Requirements Element as the response to the detection of a difficulty with the result of a prior invocation, this process definition demonstrates our view of how rework can be defined accurately. In order to explain this approach to defining rework adequately it is necessary to elaborate upon the way in which this process definition manages its artifacts and their flow. Develop Requirements takes as input an informal set of requirements and produces a set of Requirement Elements as output. When, as described above, Check Requirement Element fails it fires a trigger t this results in an instance of the rework pattern. As there is only a single response to the failure, the evaluation activity $E(t)$ has been left out, and no plan is created, and the re-invocation context is created using information from Reqt. Check Failed directly. While the context should include a range of information to support the modification of the element such as the reason the check failed, of particular interest to us is the requirement element to be defined. In the original work task Check Requirement Element (WT), the element is defined by the normal flow of Develop Requirements but here is defined to be the element e that is part of the trigger Reqt. Check Failed. Because the context binds the formal parameters in an activity to the actuals in the calling context, the changes to the element are reflected in the set of requirement elements.

As previously noted, Develop Requirements takes an informal requirements description and produces requirement elements by, in parallel, defining individual elements by creating a requirement element for part of the informal requirements ①, defining the requirement ②, and finally reviewing the defined requirement ③.

In the event that the requirement is inadequate, Check Requirement Element signals this by throwing *failure*, an instance of the trigger Reqt. Check Failed that includes the failed element. The trigger is the input to Create Define Requirement Context which creates a context including the element to be passed ④ to the re-instantiation of Define Requirement Element ⑤ and finally, control is returned to Check Requirement Element ⑥.

We now use the pattern to specify a more complicated form of rework. Figures 4 and 5 present a design activity, the overall structure of which is similar to that of the Develop Requirements activity presented above. Specifically, after the design elements are declared and defined, there is a review activity, Check Design Element that may trigger rework. In Create Design however this review activity incorporates a determination of whether or not the design element should be revised (by the re-invocation of Define Design Element) or if the requirements must be reworked by Develop Requirements to accommodate discoveries about Requirement Elements that have been made during design. This is an example of the not-uncommon situation where detecting the existence of a problem is only the first step in a process that leads to a re-invocation.

Before re-invocation can take place, participants must determine an appropriate solution to the problem. The pattern represents this as evaluation activity Evaluate Failure ① that takes the failure and produces a *plan* to assess the problem $(E(t) \rightarrow p)$. In this example, in response to the Design Check Failed trigger, Evaluate Failure creates one of two possible plans: one that requires reworking the design element in much the same way as the requirement element was reworked above ② ③, and a second in which new requirement elements are created in response to new informal requirements created as part of the plan by re-invocation of Develop Requirements ⑤ in a context created by Create Develop Requirements Context ④ that includes the new informal requirements, and then finally returning to the work context Check Design Element ⑥.

6 Related Work

Others have studied process patterns, notably Russell et al., who have specifically studied workflow structures that can be used for exception handling [11]. They derive several patterns that can be used to handle exceptions caused by a single work item. While our pattern is rightly seen as one way of handling a kind of exception – namely those exceptions that cause a previous decision to come under suspision – our work differs from that of Russell et al. in that we observe a pattern in existing process models instead of deriving possible patterns from low-level considerations.

The pattern presented in this paper helps to describe rework in formal process models. As such, it aims to help solve a long-standing problem in process improvement.

It is generally accepted that rework is a feature (or a bug) in real-world software development, and that modelling processes therefore requires modeling rework carefully [12,13]. And yet, most life-cycle models do not formally model rework and rework is not formally treated in popular software engineering texts (for example, [14,15,16]). Many life-cycle models (e.g. the Spiral Model [17]) assume that steps are repeated many times with different contexts, but do not formally model how context is managed.

In order to implement the pattern described here in a process or workflow model, the modelling language must support a trigger mechanism and a mechanism for managing the parameter binding needed for a re-invocation context. Some modelling languages support this approach more directly than others. For example HFSP's [18] `redo` clause allows reinstantiation of a step with different parameters. With other languages, especially those with semantics similar to general-purpose programming languages, this pattern can be implemented using exception handling and scopes.

6.1 Implementing Triggers as Exceptions

Because triggers are seen as devices for initiating activities that are considered to be outside the normal flow of control, it seems natural to implement a trigger with an exception handling mechanism. Several languages provide such a mechanism, borrowing from general purpose programming-language semantics [19,20,21]. Other languages allow the specification of consistency conditions that produce exceptions when violated (for example, AP5 [22], Marvel [23], Merlin [24], EPOS [25], and ALF [26]).

Wang and Kumar [10] propose a different approach to exception handling that could also be used to support rework. Their approach assumes a data-flow based workflow system, in which control flow is inferred from data dependencies between activities – if an activity B must occur after activity A, even with no data flow between A and B, a *soft* data dependency is added. Then, if A should fail, the soft dependency can be relaxed and activity A can be *deferred* to such a time as it can be safely executed (reworked). However, their approach does not seem to allow specifying a change in the invoking context of A – the source of A's data is fixed. Therefore, it seems that some kinds of rework could be modelled in this approach but it is not flexible in managing the context for rework.

6.2 Implementing Context with Scope

An invocation context must bind parameters and exceptions and provide an environment in which the re-invocation is carried out. It seems natural to use a scope to define such an environment. Any inputs an activity needs or outputs it produces can be found by searching within a scope. Therefore, by providing a different scope through re-invocation, we provide a different environment in which the activity can be performed.

Little-JIL [19] supports hierarchical scoping for parameter binding and exception handling. Also, several languages based on flow graphs and Petri-nets allow nesting of activities, where each nesting provides a scope [27,28,29].

7 Conclusions and Future Work

The examples provided in this paper fit nicely into the pattern that we have presented. Moreover the examples seem to us to provide elegant representations of the actual nature of rework. The pattern makes it clear that rework does indeed entail repeating activities and steps that had been executed previously, but it also shows that the re-execution is not exactly a repetition, but is a revisitation of previous work now with new knowledge, as contained in context information such as calling arguments. The pattern makes clear how the new knowledge is created and brought to bear.

The pattern makes it clear that rework does not entail "returning to a previous phase". In an important sense it has always been obvious that reworking a requirement because of a problem found in design did not cause a "return to the requirements phase", but rather a pause in the activities involved in design while activities involved in requirements were revisited. This intuition now seems to be very well represented in the pattern of "re-execution in a managed context", expressed precisely and elegantly through the semantics of procedure invocation. In short, the rework examples we have shown are some form of carefully managed, potentially recursive, procedure invocation.

The pattern presented here suggests opportunities to improve development environments. In recognizing that rework often entails creating new contexts for previously executed process steps, this work seems to highlight the importance of maintaining the information basis for constructing such contexts. This information basis may consist of specific instances of types of software development artifacts such as design components and design decisions, or of large and elaborate structures of such instances that have arisen during extensive development and rework activities. This suggests to us that future work aimed at creating powerful development support systems might do well to focus on how to maintain precise and articulate information about these artifact instances, and the histories of their development. As most of our previous work has focused on defining process steps, the work indicated here suggests a complementary focus on the artifact instances that they require and generate. Such complementary work might then focus on how to store, structure, and present artifact structures in ways that enable tools to better support development, which inevitably includes rework. Such tools would not simply present a developer with the need to revisit a previously executed step, but would supplement that with an articulate description of the circumstances under which the step had previously been carried out. This would enable the developer to make better informed decisions about how to address the needed rework.

The modest number of examples provided in this paper are only a representative sample of a larger number of examples that seem to fit into the pattern that has been presented here. These examples all seem to be cleanly and clearly represented as instances of the pattern that we have presented. It is our conjecture that this pattern will suffice to describe many other instances of rework, found both in software engineering and in other disciplines. We remain interested in examining other instances in order to explore our hypothesis that this pattern might well serve as a definition of the term "rework". Should our conjecture prove to be correct, then we expect that this work could lead to more effective support for rework as is needed in many of the varied domains in which it is a central feature of how work is carried out.

Acknowledgements

The authors wish to express gratitude to Stanley M. Sutton, Jr, Reda Bendraou, Barbara Staudt Lerner, Stefan Christov, Lori A. Clarke, and members of the Laboratory for Advanced Software Engineering Research at the University of Massachusetts Amherst who have participated in this research, and clarified the points made in this paper.

This material is based upon work supported by the US National Science Foundation under Award Nos. CCR-0427071, CCR-0204321 and CCR-0205575. The views and conclusions contained herein are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of The National Science Foundation, or the U.S. Government.

References

1. Cass, A.G., Sutton Jr., S.M., Osterweil, L.J.: Formalizing rework in software processes. In: Proc. of the 9th European Workshop on Soft. Proc. Technology, Helsinki, Finland, September 1–2 (2003)
2. Clarke, L.A., Gaitenby, A., Gyllstom, D., Katsh, E., Marzilli, M., Osterweil, L.J., Sondeimer, N.K., Wing, L., Wise, A., Rainey, D.: A process-driven tool to support online dispute resolution. In: Intl. Conf. on Digital Government Research, San Diego, CA. ACM Press, New York (2006)
3. Osterweil, L.J., Wise, A., Clarke, L.A., Ellison, A.M., Hadley, J.L., Boose, E., Foster, D.R.: Process technology to facilitate the conduct of science. In: Li, M., Boehm, B., Osterweil, L.J. (eds.) SPW 2005. LNCS, vol. 3840, pp. 403–415. Springer, Heidelberg (2006)
4. Osterweil, L.J., Clarke, L.A., Podorozhny, R., Wise, A., Boose, E., Ellison, A.M., Hadley, J.: Experience in using a process language to define scientific workflow and generate dataset provenance. In: Proc. of the 16th ACM SIGSOFT Intl. Symp. on Foundations of Soft. Engineering (FSE16), Atlanta, GA. ACM Press, New York (2008)
5. Christov, S., Chen, B., Avrunin, G.S., Clarke, L.A., Osterweil, L.J., Brown, D., Cassells, L., Metens, W.: Rigorously defining and analyzing medical processes: An experience report. In: Giese, H. (ed.) MoDELS Workshops 2007. LNCS, vol. 5002. Springer, Heidelberg (2008)
6. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading (1995)
7. Object Management Group: OMG Unified Modeling Language (OMG UML) Superstructure. Technical Report formal/2007-11-02, Object Management Group, Version 2.1.2 (November 2007)
8. Lerner, B.S., Christov, S., Wise, A., Osterweil, L.J.: Exception handling patterns for processes. Technical Report 08-06, UMass Dept. of Comp. Sci. (March 2008)
9. Clemm, G., Osterweil, L.: A mechanism for environment integration. ACM Trans. on Prog. Lang. and Systems (TOPLAS) 12(1) (1990)
10. Wang, J., Kumar, A.: Exception handling using task deferral in document-driven workflow systems. In: Proc. of the Annual Workshop on Information Technology and Systems (WITS) (2005)
11. Russell, N., van der Aalst, W.M.P., ter Hofstede, A.H.M.: Exception handling patterns in process-aware information systems. Technical report, BPM Center (2006)
12. Haley, T., Ireland, B., Wojtaszek, E., Nash, D., Dion, R.: Raytheon Electronic Systems experience in software process improvement. Technical Report CMU/SEI-95-TR-017, Carnegie-Mellon Software Engineering Institute (November 1995)

13. Butler, K., Lipke, W.: Software process achievement at Tinker Air Force Base. Technical Report CMU/SEI-2000-TR-014, Carnegie-Mellon Software Engineering Institute (September 2000)
14. Ghezzi, C., Jazayeri, M., Mandrioli, D.: Fundamentals of Software Engineering. Prentice Hall, Englewood Cliffs (1991)
15. Pressman, R.S.: Software Engineering: A Practitioner's Approach, 4th edn. McGraw-Hill, New York (1997)
16. Sommerville, I.: Software Engineering, 5th edn. Addison-Wesley, Reading (1996)
17. Boehm, B.W.: A spiral model of software development and enhancement. IEEE Computer 21(5), 61–72 (1988)
18. Suzuki, M., Iwai, A., Katayama, T.: A formal model of re-execution in software process. In: Proc. of the 2nd Intl. Conf. on the Soft. Process, Berlin, Germany, pp. 84–99. IEEE Press, Los Alamitos (1993)
19. Wise, A.: Little-JIL 1.0 Language Report. Technical Report 98-24, UMass Dept. of Comp. Sci. (April 1998)
20. Sutton Jr., S.M., Osterweil, L.J.: The design of a next-generation process language. In: Jazayeri, M. (ed.) ESEC 1997 and ESEC-FSE 1997. LNCS, vol. 1301, pp. 142–158. Springer, Heidelberg (1997)
21. Sutton Jr., S.M., Heimbigner, D., Osterweil, L.J.: APPL/A: A language for software-process programming. ACM Trans. on Soft. Engineering and Methodology (TOSEM) 4(3), 221–286 (1995)
22. Cohen, D.: AP5 Manual. USC, Info. Sci. Institute (March 1988)
23. Kaiser, G.E., Barghouti, N.S., Sokolsky, M.H.: Experience with process modeling in the MARVEL software development environment kernel. In: Shriver, B. (ed.) 23rd Annual Hawaii Intl. Conf. on System Sci., Kona HI, vol. II, pp. 131–140 (January 1990)
24. Junkermann, G., Peuschel, B., Schäfer, W., Wolf, S.: MERLIN: Supporting cooperation in software development through a knowledge-based environment. In: Finkelstein, A., Kramer, J., Nuseibeh, B. (eds.) Soft. Process Modelling and Technology, pp. 103–129. Wiley, Chichester (1994)
25. Conradi, R., Hagaseth, M., Larsen, J.O., Nguyên, M.N., Munch, B.P., Westby, P.H., Zhu, W., Jaccheri, M.L., Liu, C.: EPOS: Object-oriented cooperative process modelling. In: Finkelstein, A., Kramer, J., Nuseibeh, B. (eds.) Soft. Process Modelling and Technology, pp. 33–70. Wiley, Chichester (1994)
26. Canals, G., Boudjlida, N., Derniame, J.C., Godart, C., Lonchamp, J.: ALF: A framework for building process-centred software engineering environments. In: Finkelstein, A., Kramer, J., Nuseibeh, B. (eds.) Soft. Process Modelling and Technology, pp. 153–185. Wiley, Chichester (1994)
27. Bandinelli, S., Fuggetta, A., Grigolli, S.: Process modeling in-the-large with SLANG. In: Proc. of the 2nd Intl. Conf. on the Soft. Process, pp. 75–83. IEEE Computer Society Press, Los Alamitos (1993)
28. Deiters, W., Gruhn, V.: Managing software processes in the environment melmac. In: Proc. of the 4th ACM SIGSSOFT/SIGPLAN Symp. on Practical Soft. Dev. Environments, Irvine, CA, pp. 193–205. ACM Press, New York (1990)
29. Casati, F., Ceri, S., Paraboschi, S., Pozzi, G.: Specification and implementation of exceptions in workflow management systems. ACM Trans. on Database Systems (TADS) 24(3), 405–451 (1999)

Achieving On-Time Delivery: A Two-Stage Probabilistic Scheduling Strategy for Software Projects

Xiao Liu[1], Yun Yang[1], Jinjun Chen[1], Qing Wang[2], and Mingshu Li[2]

[1] Faculty of Information and Communication Technologies
Swinburne University of Technology
Hawthorn, Melbourne, Australia 3122
{xliu,yyang,jchen}@swin.edu.au
[2] Laboratory for Internet Software Technologies, Institute of Software
Chinese Academy of Sciences
Beijing, 100080 P.R. China
{wq,mingshu}@itechs.iscas.ac.cn

Abstract. Due to the uncertainty of software processes, statistic based schedule estimation and stochastic project scheduling both play significant roles in software project management. However, most current work investigates them independently without an integrated process to achieve on-time delivery for software development organisations. For such an issue, this paper proposes a two-stage probabilistic scheduling strategy which aims to decrease schedule overruns. Specifically, a probability based temporal consistency model is employed at the first pre-scheduling stage to support a negotiation between customers and project managers for setting balanced deadlines of individual software processes. At the second scheduling stage, an innovative genetic algorithm based scheduling strategy is proposed to minimise the overall completion time of multiple software processes with individual deadlines. The effectiveness of our strategy in achieving on-time delivery is verified with large scale simulation experiments.

Keywords: Software Process, Schedule Estimation, Project Scheduling, Probabilistic Strategy, Genetic Algorithm.

1 Introduction

A software project is typically characterised by dynamic changes of the development environment and variant decisions of human stakeholders [13, 15]. Therefore, the estimation of software development schedule (and cost) with uncertainty as well as project scheduling under uncertainty are widely investigated and applied in software projects [6, 8]. But still, they are considered to be challenging issues for software development organisations of all sizes.

It has been witnessed that for a majority of software projects, on-time delivery of core capabilities is increasingly become the main focus of software processes in the dynamic business world nowadays [13]. Meanwhile, for many software development organisations, especially of small and medium sizes, their main business targets are short-term contracts from a relatively fixed group of customers in the market. These

Q. Wang et al. (Eds.): ICSP 2009, LNCS 5543, pp. 317–329, 2009.

customers usually require the development of small scale software components within hard deadlines so as to meet their dynamic and urgent business needs. The competitive strength of these software development organisations critically relies on the ability of on-time delivery. Therefore, instead of pursuing multiple objectives such as reducing project schedule, budget and staff overload at the same time, this paper focuses on achieving on-time delivery. For software development organisations, an estimated software schedule serves as the guideline for project bidding and planning given specific customer needs and software process performance [14]. Furthermore, project scheduling is to decide "who does what" and optimise specific objectives, e.g. minimum schedule and budget. In recent years, great efforts have been dedicated to statistic based schedule estimation and stochastic project scheduling in software project management [6, 9]. However, most current work investigates schedule estimation and project scheduling independently without an integrated process. On one hand, estimated schedules for specific software processes cannot be realised without project scheduling to assign proper employees with proper tasks. On the other hand, without schedule estimation, project scheduling will not be effectively guided and probably result in frequent schedule overruns.

To achieve on-time delivery, this paper proposes a two-stage probabilistic project scheduling strategy to address the above issues. Specifically, at the first pre-scheduling stage for individual software processes, a probability based temporal consistency model is presented to facilitate a win-win negotiation between customers and project managers. This negotiation, namely pre-scheduling, supports the setting of balanced deadlines based on the process performance baseline. At the second scheduling stage for multiple software processes, an innovative genetic algorithm (GA) [1] based scheduling strategy which utilises a two-phase searching algorithm and a package based initialisation approach is proposed. The objective for the scheduling stage is to minimise the overall completion time of multiple software processes given that all individual software processes can be completed ahead of the deadlines as set at the pre-scheduling stage.

The remainder of the paper is organised as follows. Section 2 presents the related work and problem analysis. Section 3 gives the overview of our two-stage probabilistic project scheduling strategy. Section 4 presents the pre-scheduling stage and Section 5 proposes the scheduling stage. Section 6 describes large scale simulation experiments to verify the effectiveness of our strategy in achieving on-time delivery. Finally, Section 7 addresses our conclusions and future work.

2 Related Work and Problem Analysis

2.1 Related Work

With the dynamic nature of software development environments, various uncertainty and inconsistencies arise and thus bring challenges for schedule (and cost) estimation in software processes [15]. Jørgensen and Shepperd present a systematic review of software development schedule/cost estimation studies in [9]. For the 9 categories of research topics, estimation methods account for 61% of the samples and rank the first place. In recent years, various strategies such as feature prioritisation, core capability determination, risk driven strategies, earned value management, statistical process control and so forth, are presented to deal with uncertainty assessment and uncertainty control for software schedule (and cost) [13]. For schedule estimation of a specific software process, one of the most important concepts is Process Performance Baseline (PPB) [6]. PPB utilises two indicators, i.e. process performance and process capability. Here, process performance is "a measure of actual results achieved by following a process" and process capability is "the range of expected results that can be achieved by following a process". In [14], a statistic based approach is proposed to establish and refine software process performance baseline where the average value μ and the standard deviation σ of data samples are defined as process performance and capability respectively. Specifically, process performance baseline is normally controlled under the limits of $\mu \pm 3\sigma$.

Project scheduling is to allocate proper tasks to proper employees or subcontractors in order to result in successful projects. Due to the uncertainty of software projects, most project scheduling strategies aim to generate a feasible schedule within given constraints, such as schedule and budget, to serve as a baseline schedule for real project execution. Five fundamental approaches for scheduling under uncertainty are identified in [6], i.e. reactive scheduling, stochastic project scheduling, fuzzy project scheduling, proactive scheduling and sensitivity analysis. Specifically, stochastic project scheduling strategies mainly concern about scheduling project tasks with uncertain durations in order to minimise the expected project schedule [3, 16]. Among them, genetic algorithm, as a commonly applied heuristic method, is often employed to solve complicated optimisation problems in resource constrained scheduling problems [12]. An empirical study in [7] demonstrates that the evolution algorithm is capable of finding the best-known solutions in 68% of the 2370 instances with an average overall error rate of 0.95%. [3] proposes a time-line based model as well as 8 heuristic rules to simulate real-world situations to enhance the ability of GA. In [1], structured studies have shown that GA is flexible and accurate for many different software project scenarios.

2.2 Problem Analysis

For a specific software project, schedule estimation and project scheduling are two fundamental issues for achieving on-time delivery. In practice, for project bidding and negotiation, project managers usually need to estimate project schedules based the statistic performance of software development organisations. During this period of time, project deadlines are set based on estimated schedules. However, practical data show that about one-third of the projects exceed their estimated schedules by 25% [13]. Two of the critical problems cause schedule overruns are as follows.

1) A project schedule is not well balanced between the software process performance baseline and customer needs. If project deadlines are far beyond the software process performance baseline, schedule overruns are highly expected. Therefore, in practice, some robust project scheduling strategies such as temporal protection intentionally extends the statistic task durations to protect the baseline schedule from resource breakdowns [6]. Meanwhile, most work only emphasises the role of project managers to estimate schedules and set project deadlines for individual software processes. However, the deadlines may often be set unrealistically tight due to customers'

pressure and thus results in frequent schedule overruns. Therefore, compromised project schedules which achieve proper balance between the software process performance baseline and customer needs are certainly more desirable.

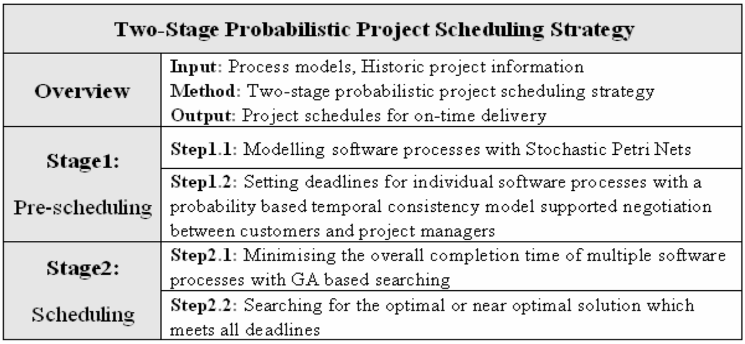
2) An individual baseline schedule does not consider the situation of multiple software processes [3]. One of its priorities for project scheduling is to ensure that software processes can be completed within specific deadlines. However, since software process performance is heavily dependent on the people who execute the process rather than the device of the product line, one of the major reasons deteriorates the performance and causes significant delays is the competition of employees among multiple software processes. In fact, this problem frequently occurs in a software development organisation and causes serious overruns even though project schedules are estimated based on the performance baseline. In another word, without considering the situation of multiple software processes, baseline schedules for individual software processes cannot guarantee the success of on-time delivery.

To tackle the above two problems, joint efforts need to be dedicated by schedule estimation and project scheduling as an integrated project scheduling strategy.

3 A Two-Stage Probabilistic Project Scheduling Strategy

In this section, we present the overview of our two-stage probabilistic project scheduling strategy. As depicted in Table 1, our project scheduling strategy consists of two stages, i.e. the pre-scheduling stage and the scheduling stage.

Table 1. Project Scheduling Strategy



Two-Stage Probabilistic Project Scheduling Strategy	
Overview	**Input**: Process models, Historic project information **Method**: Two-stage probabilistic project scheduling strategy **Output**: Project schedules for on-time delivery
Stage1: Pre-scheduling	**Step1.1**: Modelling software processes with Stochastic Petri Nets
	Step1.2: Setting deadlines for individual software processes with a probability based temporal consistency model supported negotiation between customers and project managers
Stage2: Scheduling	**Step2.1**: Minimising the overall completion time of multiple software processes with GA based searching
	Step2.2: Searching for the optimal or near optimal solution which meets all deadlines

At the pre-scheduling stage, the main objective is to set balanced deadlines for individual software processes. At this stage, Step 1.1 is to model software processes with Stochastic Petri Nets [2] which will be introduced later in this section as the specification tool. Step 1.2 is to set balanced deadlines for individual software processes with a probability based temporal consistency model which is to support a win-win negotiation between customers and project managers. At the scheduling stage, the main objective is to generate a scheduling plan which assigns proper tasks to proper employees in order to achieve on-time delivery. At this stage, an innovative GA based scheduling strategy is conducted as Step 2.1 to minimise the overall completion time of multiple software processes. Step 2.2 is to search for the optimal or near optimal solution (i.e. the best or near best project scheduling plan) which meets the deadlines for individual software processes set at the pre-scheduling stage.

Technical details of pre-scheduling and scheduling will be presented in Section 4 and Section 5 respectively. Here, we introduce Stochastic Petri Nets (SPN) to specify software processes. SPN can provide powerful abstractions of such as control flows, underlying resources, stochastic temporal information. Besides conventional Petri Nets notations of place, transition and arc, an additional new notation of task weight is employed for specifying statistic task durations. As proposed in [11], task weight is defined with the probability and statistic iteration times of each task based on four fundamental control flow patterns, i.e. sequence, choice, parallelism and iteration. Specifically, for a sequence process, the task weight is specified as 1. For a choice process, the task weight is equal to the probability of the path to be chosen. For a parallelism process, the task weight of the path with largest expected duration is defined as 1 while others are defined as 0 since they do not dominate the overall execution time. For an iteration process, the task weight is defined as the statistic iteration times. The duration distribution is associated with each task based on the statistic performance. The purpose of modelling software processes with SPN is to reflect the stochastic temporal information to support the schedule estimation and project scheduling.

4 Pre-scheduling for Individual Software Processes

The temporal consistency model is defined to quantitatively measure the state of specific processes given specified temporal constraints [4, 5]. In this paper, we employ a probability based temporal consistency model to support deadline setting. The probability based temporal consistency model is defined based on the concept of weighted joint normal distribution [11]. Weighted joint normal distribution can be used to analyse the distribution of the overall completion time, namely the performance baseline for a specific process based on the distribution of individual task durations. Here, we first define some notations. For a specific task t_i, its maximum duration and minimum duration are defined as $D(t_i)$ and $d(t_i)$ respectively. In addition, we employ the "3σ" rule which has been widely used in statistical analysis [10]. The "3σ" rule depicts that for any sample coming from normal distribution model, it has a probability of 99.73% to fall into the range of $[\mu-3\sigma,\mu+3\sigma]$ where μ is the average value and σ is the standard deviation. In this paper, we define the maximum duration as $D(t_i)=\mu_i+3\sigma_i$ and the minimum duration as $d(t_i)=\mu_i-3\sigma_i$. Meanwhile, for a software process SP which consists of n tasks, its deadline is denoted as $F(SP)$ and its start time is denoted as $S(SP)$. A deadline of $F(SP)$ is a fixed time by which the process SP must be completed [5]. Now, based on [11], we present the definition of probability based temporal consistency model on fixed time temporal constraints (deadlines) as follows.

Definition: (Probability based temporal consistency model)

For a software process SP with a deadline of $F(SP)$ that starts at $S(SP)$, $F(SP)$ is said to be of:

1) Absolute Consistency (AC), if $\sum_{i=1}^{n} w_i(\mu_i + 3\sigma_i) < F(SP) - S(SP)$ *;*

2) Absolute Inconsistency (AI), if $\sum_{i=1}^{n} w_i(\mu_i - 3\sigma_i) > F(SP) - S(SP)$ *;*

3) $\alpha\%$ *Consistency (* $\alpha\%$ *C), if* $\sum_{i=1}^{n} w_i(\mu_i + \lambda\sigma_i) = F(SP) - S(SP)$ *.*

Here, w_i *stands for the weight of task* t_i*,* λ *(* $-3 \le \lambda \le 3$ *) is defined as the* $\alpha\%$ *confidence percentile with the cumulative normal distribution function of*

$$f(\mu_i + \lambda\sigma_i) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{\mu_i+\lambda\sigma_i} e^{{-(x-\mu_i)^2}\Big/{2\sigma_i^2}} \bullet dx = \alpha\% \ (0 < \alpha < 100).$$

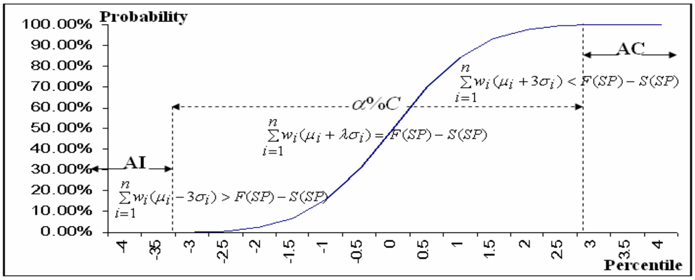


Fig. 1. Probability Based Temporal Consistency

With the above model, different states of the software process towards a specific deadline are described with continuous probability values. As shown in Figure 1, these values form a continuous Gaussian curve. According to the probability based temporal consistency model, the basic performance baseline (average activity duration) can only guarantee a probability consistency state of 50% which is normally much lower than the customer's minimal confidence. Therefore, a more realistic deadline acceptable to all stakeholders needs to be negotiated [11].

Now we demonstrate the win-win negotiation for setting balanced deadlines. Here, we denote the obtained weighted joint distribution of the target software process SP as $N(\mu_{sp}, \sigma_{sp}^2)$ and assume the minimum threshold to be $\beta\%$ for the probability consistency which implies the customer's acceptable bottom-line probability for on-time delivery. The actual negotiation starts with the customer's initial suggestion of a deadline of $F(SP)$ and the evaluation of the corresponding temporal consistency state by the project manager. If $F(SP) - S(SP) = \mu_{sp} + \lambda\sigma_{sp}$ with λ as the $\alpha\%$ percentile, and $\alpha\%$ is below the threshold of $\beta\%$, then the deadline needs to be adjusted, otherwise the negotiation terminates. For the negotiation, the subsequent process is the iteration that the customer proposes a new deadline which is less restricted as the previous one and the project manager re-evaluates the consistency states, until it reaches or is above the minimum probability threshold [11]. This win-win negotiation, i.e. pre-scheduling process, facilitates the setting of balanced project deadlines.

5 Scheduling for Multiple Software Processes

To achieve on-time delivery, our scheduling objective is to minimise the overall completion time of multiple software processes given their individual deadlines set at the pre-scheduling stage can be met. For such an objective, we propose an innovative GA based project scheduling strategy. GA is a class of evolutionary algorithms inspired by evolutionary biology [1, 16]. In GA, three basic GA operations, i.e. selection, crossover and mutation, are conducted to imitate the evolution process in nature. After the stopping condition is met, the solution with the best fitness value which represents the optimal or near-optimal solution is returned [3].

Here, in order to enhance the performance and satisfy real-world situations, we first identify two critical aspects which should be tackled in GA based project scheduling.

(1) How to achieve on-time delivery for individual software processes while minimising the overall completion time for multiple software processes.

As discussed in Sections 1 and 2, it is a priority to ensure on-time delivery. However, for software development organisations, it is also important to minimise the overall completion time for multiple software processes in order to reduce the project cost. Therefore, how to effectively balance these two objectives is a critical aspect.

(2) How to implement heuristic rules in GA for practical restrictions.

In real-world software projects, there are many restrictions which affect the tasks-to-employees assignment. Therefore, in order to support decision making under more realistic conditions, many heuristic rules such as resource continuity (e.g. assigning a group of highly related tasks to a fixed employee so as to reduce the overhead of task transfer), adjustment of workload and overstaffed projects, are supplemented [3]. However, how to implement these heuristic rules in GA is a challenging issue.

```
Strategy:GA based project scheduling
Input: Software process models
       Employee models
Output: Generated Scheduling Plans
//two dimensional encoding
1) ENCODING(L);
// package based generation of initial populaton
2) INITIALISATION(Population)
// Optimising overall completion time
3) While (maximum generation is not reached)
   {
    //selecting Parents from Population
4)    SELECTION(Population) ;
5)    CRSSSOVER(Parents)⟶Children;
    //change resource for a random task
6)    MUTATION(Children);
    //check with software process models and heuristic rules
7)    VALIDATION(Children);
    //store the best child into a solution set; replace worst child
    //with the best child
8)    STORE(BestChild, SolutionSet);
9)    REPLACE(WorstChild, BestChild);
   }
10) Return SolutionSet and BestChild in Population.
// Searching for the BestSolution from solution set
11) BestSolution =BestChild;
12) While (not end of the SolutionSet)
   {
    //Compare each solution and set BestSolution as the one meets all
    //individual deadlines and has smaller overall completion time
13)     BestSolution=COMPARE(BestSolution, Child);
   }
14) Return the BestSolution;
// decoding and update
15) DECODING(BestSolution);
16) UPDATE(L).
```

(a) GA Based Project Scheduling Strategy

```
Function: INITIALISATION
Input: Software process models $SP\{SP_i(1 \le i \le k)\}$
       Employee models $EP\{EP_i(1 \le i \le p)\}$
Output: $sched_i, alloc_i$ : chromosome
1) for i= 1 to size    // generate a fixed size of solutions
// generate a single solution
2)     for each Software Process SP
// arrange tasks to form packages
// function size() returns the number of tasks in each SP
3)         $sched_i\left(\sum_{m=1}^{j-1} size(SP_m)+1 : \sum_{m=1}^{j} size(SP_m)\right)$
               $= task\left(\sum_{m=1}^{j-1} size(SP_m)+1\right) : task\left(\sum_{m=1}^{j} size(SP_m)\right);$
// allocate employees to each package
4)         for each Package P in each Software Process SP
// check package requirements with employee models and
// return the matched employee set $EP'\{\}$
5)             $EP'\{\} = Check(Package, EP\{\})$
// apply heuristic rules $R\{\}$ and return the valid employee set $EP''\{\}$
6)             $EP''\{\} = Check(R\{\}, EP'\{\})$
// random allocate valid employees to the package
7)             $alloc_i\left(\sum_{m=1}^{j-1} size(SP_m)+1 : \sum_{m=1}^{j} size(SP_m)\right)$
               $= random(EP''\{\});$
8)         end
9)     end
10) end
```

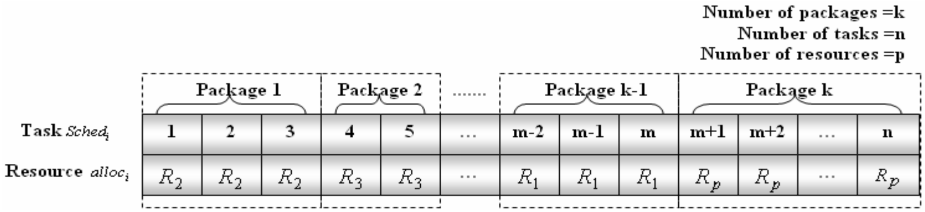
(b) Package Based Initialisation

Fig. 2. Pseudo-code for GA based Project Scheduling Strategy

In our GA based scheduling strategy, we propose a two-phase searching algorithm to address the first aspect and a package based initialisation approach to address the second aspect. The pseudo-code for our GA based project scheduling strategy is presented in Figure 2 where Figure 2(a) presents the scheduling strategy and Figure 2(b) presents the package based initialisation approach. As shown in Figure 2(a), our scheduling strategy has two main input parameters, i.e. software process models and the employee models. Here, software processes models are specified with SPN as introduced in Section 3. Each software process model describes the control flows, namely the precedence relationships between tasks. Meanwhile, based on the results of the pre-scheduling stage, stochastic temporal information such as process deadlines and the statistics of task durations are also provided. The employee models mainly define the specific skills possessed by individual employees and some other information related to the decisions on task assignment such as his/her proficiency level (measured in execution speed), payment rate, workload, project experience and so on. Our strategy starts with the encoding of an empty task-employee list and generation of initial population with package based initialisation (line 1 and line 2). Line 3 to line 14 is the GA based two-phase searching algorithm to find the best solution. Finally, the best solution is decoded (line 15) and the task-employee list is updated (line 16). After this scheduling process, the task-employee list which represents the optimal or near-optimal scheduling plan is generated.

Two-phase searching algorithm. To address the first aspect identified, our GA based scheduling strategy adopts a two-phase searching algorithm as depicted in Figure 2(a). The first phase (line 3 to line 10) is to optimise the overall completion time of multiple software processes. Based on genetic operations, e.g. selection, crossover and mutation (line 4 to line 6), the searching space is expanded and solutions with higher fitness values are found. Here, the fitness value is defined according to the overall completion time of all the software processes. The smaller the overall completion time, the higher the fitness value is. The function of validation (line 7) is to check the generated solutions if they are correct with restrictions, e.g. the precedence relationships and other heuristic rules. During each generation, the best child is stored in a solution set (line 8) and the worst child is replaced by the best one (line 9). The second phase (line 11 to line 14) is to search for the best solution from the whole solution set composed of the best child in each generation produced in the first searching phase. The best solution found should meet the deadlines of individual software processes while having the minimum overall completion time. Our two-phase searching algorithm guarantees the return of balanced solutions. The reason is that on one hand, a vast number of possible solutions are generated and evaluated in the first searching phase. On the other hand, the best child with the minimum overall completion time of each generation is recorded in the solution set. Therefore, if a balanced solution cannot be found, especially after huge numbers of generations, we are able to claim that it is not possible to find a valid solution which can achieve on-time delivery for all software processes. Otherwise, a balanced scheduling plan which meets the deadlines of individual software processes should be found in the solution set. However, to support decision making, highly ranked solutions in the solution set will be returned instead in this case where no best solution exists. The project managers can make further decisions, e.g. recruitment of more employees or outsource to subcontractors, to ensure the success of on-time delivery.

Package based initialisation approach. As the second aspect identified, practical restrictions in the real-world software projects should be considered in the scheduling strategy so as to support and satisfy realistic conditions. For such a purpose, we propose an innovative package based initialisation approach based on two dimensional encoding. As depicted in Figure 3, the first dimension $Sched_i$ denotes scheduled tasks and the second dimension $alloc_i$ denotes the allocated resources (employees or subcontractors). As discussed in many literatures, GA initialisation for the initial population is critical towards the outcomes of GA [16]. Basically, the initial population should be valid and effective. In our scenario, to be valid, the initial population should assign specific tasks to valid employees who possess the ability to fulfil these tasks. Meanwhile, for a specific employee, the tasks assigned to him/her should be conformed to their precedence relationships defined in software process models. To be effective, more restrictions, e.g. resource continuity and the 8 heuristic rules proposed in [3], should be applied for practical project management. For such an issue, a package based random initialisation approach is to support the generation of population which is both valid and effective. 'Package' here denotes a group of highly related tasks in the same software process, which can be defined by experienced project managers, with correct precedence relationships. Meanwhile, these tasks share the same employees allocated to each package with the enforcement of restrictions. The design of a package here not only ensures the correct task flows for a software process since tasks with correct precedence relationships are assigned to the same employees, but also is capable of facilitating resource continuity and other heuristics. For a specific package, a set of employees are formed first by checking required abilities (line 5 of Figure 2(b)). Afterwards, a further checking process is applied to select valid employees based on enforced heuristics (line 6 of Figure 2(b)). Finally, one of the valid employees is randomly assigned to this package (line 7 of Figure 2(b)). As shown in Figure 3, for the employee assignment of n tasks, an employee set, say $\{R_1, R_2, R_3, \ldots R_p\}$ is first formed. After that, specific heuristic rules are applied. For example, one of the employees, say R_2, which currently has the lowest workload, is assigned to Package 1. Following a similar way, a valid and effective solution comprised of Packages 1, 2, ..., k (Figure 3) is automatically generated (loop of line 2 of Figure 2(b)). The initial population is formed by a fixed size of automatically generated solutions (loop of line 1 of Figure 2(b)).



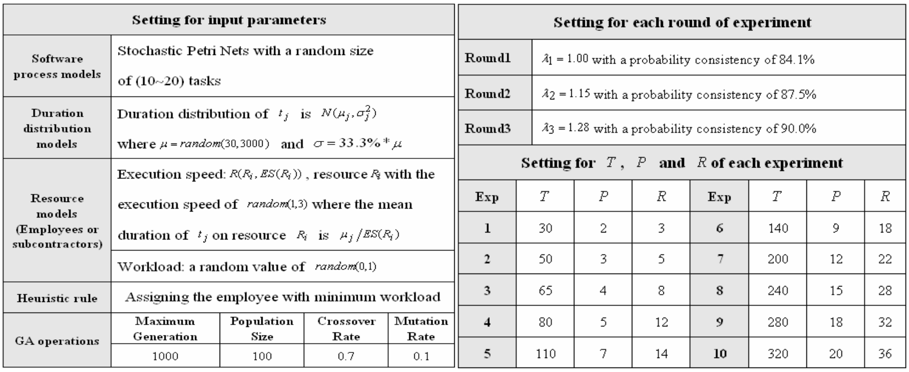
Task $Sched_i$	1	2	3	4	5	...	m-2	m-1	m	m+1	m+2	...	n
Resource $alloc_i$	R_2	R_2	R_2	R_3	R_3	...	R_1	R_1	R_1	R_p	R_p	...	R_p

Fig. 3. Package Based Initialisation Approach

6 Evaluation

In this section, we evaluate the effectiveness of our two-stage probabilistic project scheduling strategy through large scale simulation experiments where the simulation results are independent of specific platforms. The settings for simulation experiments are presented in Table 2. We manually generate stochastic Petri Nets with a random size of 10 to 20 tasks. To simplify our simulation scenario so as to focus on the objective of on-time delivery, we only set two attributes for employee models, i.e. the execution speed (measured for proficiency level) and the workload. The execution speed of each employee is randomly specified from a range of 1 to 3 where 1 denotes that the mean execution time equals to the expected duration while 3 denotes that the mean execution time equals to the expected duration divided by 3. The workload of each employee is a random value from 0 to 1 where 0 means the employee has not been assigned with any tasks and 1 means the employee is fully occupied and cannot be assigned with more tasks. For each task assignment to a specific employee, his/her workload increases by 0.1 so an employee can take up to 10 tasks. Accordingly, we adopt one simple heuristic rule that is assigning the current task to the employee with minimum workload. As for the settings of GA operations, maximum generation is used as the stopping condition and its value is 1000. The initial population size is set as 100. The crossover rate and mutation rate are set as 0.7 and 0.1 respectively as common practice. To evaluate our strategy with large scale simulation experiments, we conducted 3 rounds, i.e. COM(1.00), COM(1.15) and COM(1.28) with different λ as 1.00, 1.15 and 1.28 to reflect different pre-scheduling results with 84.1%, 87.5% and 90.0% consistency respectively. As depicted in Table 3, within each round, we conducted 10 experiments with different settings of number of tasks (T) with a range of 30-320 in total whilst 10-20 for each process, the number of processes (P) with a range of 2-20, and the number of resources (R) with a range of 3-36.

Table 2. Settings for Simulation Experiments



Setting for input parameters				
Software process models	Stochastic Petri Nets with a random size of (10~20) tasks			
Duration distribution models	Duration distribution of t_j is $N(\mu_j,\sigma_j^2)$ where $\mu = random(30,3000)$ and $\sigma = 33.3\% * \mu$			
Resource models (Employees or subcontractors)	Execution speed: $R(R_i, ES(R_i))$, resource R_i with the execution speed of $random(1,3)$ where the mean duration of t_j on resource R_i is $\mu_j / ES(R_i)$			
	Workload: a random value of $random(0,1)$			
Heuristic rule	Assigning the employee with minimum workload			
GA operations	Maximum Generation	Population Size	Crossover Rate	Mutation Rate
	1000	100	0.7	0.1

Setting for each round of experiment							
Round1	$\lambda_1 = 1.00$ with a probability consistency of 84.1%						
Round2	$\lambda_2 = 1.15$ with a probability consistency of 87.5%						
Round3	$\lambda_3 = 1.28$ with a probability consistency of 90.0%						
Setting for T , P and R of each experiment							
Exp	T	P	R	Exp	T	P	R
1	30	2	3	6	140	9	18
2	50	3	5	7	200	12	22
3	65	4	8	8	240	15	28
4	80	5	12	9	280	18	32
5	110	7	14	10	320	20	36

Two attributes, i.e. the *improvedPercent* (the difference of the overall completion time before and after our GA based scheduling strategy divided by the one before) and the *overrunRate* (the number of processes which fails to be completed within deadlines divided by the number of total processes), are investigated. Here, for fairness, the overall completion time before our scheduling strategy is specified as the average completion time of all valid solutions generated in the GA based searching phase. Meanwhile, we also investigate the *overrunRate* of the initial project scheduling plans generated by package based initialisation but without the GA based searching phase, namely a NIL strategy, for the purpose of comparison. To investigate the statistic performance, each experiment is executed for 100 times. Therefore, in our simulation, 3 rounds, 10 experiments and 100 execution times, namely a large scale of 3000 independent experiments, have been executed.

The simulation results are presented in Figure 6. As can be seen from the left subplot in Figure 6, the mean *improvedPercent* and the number of tasks are roughly on the same trend. This verifies the effectiveness of our strategy in scheduling multiple software processes and optimising their overall completion time. Furthermore, our strategy performs even better when the scheduling scenario becomes more complicated as shown by increasing *improvedPercent*. We also note that despite the increase of probability consistency, i.e. from 84.1%, 87.5% to 90.0%, which means less restricted deadlines, the mean *improvedPercent* does not vary much. Hence, a larger probability consistency does not guarantee a better *improvedPercent.* As for the *overrunRate* depicted in the right subplot of Figure 6, the simulation results show that if without GA based searching phase, i.e. the NIL strategy, the average *overrunRate* is high with a growing trend based on the increase of the number of software processes. However, with our scheduling strategy, the *overrunRate* is much lower as depicted. In this case, the increase of probability consistency results in the lower mean *overrunRate.* To conclude, we can claim that our two-stage probabilistic scheduling strategy is effective for achieving on-time delivery.

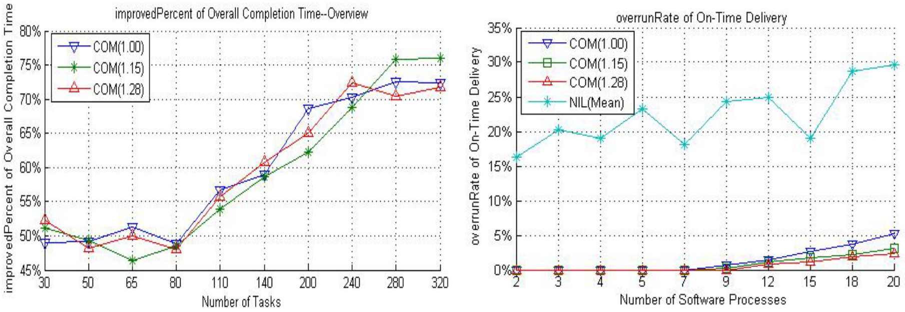


Fig. 6. Simulation Results

7 Conclusions and Future Work

In this paper, we have proposed a two-stage probabilistic scheduling strategy which integrates statistic based schedule estimation and stochastic project scheduling in order to achieve on-time delivery of software projects. Our strategy aims to generate the best scheduling plan which can meet the deadlines of individual software processes while having the minimum overall completion time of multiple software processes.

Hence, at the first pre-scheduling stage, a probability based temporal consistency model has been presented to facilitate a win-win negotiation between customers and project managers for setting balanced project deadlines based on individual software process performance baselines. Given these deadlines, at the second scheduling stage, an innovative GA based project scheduling strategy which utilises a two-phase searching algorithm and a package based initialisation approach has been proposed to search for the best scheduling plan under the resource constraint of the current software development organisations. Based on the results of large scale simulation experiments, it has been verified that the best scheduling plan can be found in most cases with our project scheduling strategy. However, even in the case where such solution does not exist due to the resource constraint, the generated solutions can still support the project manager to make further decisions such as recruitment of employees or outsourcing to ensure the success of on-time delivery.

In the future, to tackle the case where the best scheduling plan can not be found, we will try to identify the key software processes where on-time delivery can be achieved with minimum increase of extra cost.

References

1. Alba, E., Chicano, J.F.: Software Project Management with GAs. Information Sciences 177, 2380–2401 (2007)
2. Bucci, G., Sassoli, L., Vicario, E.: Correctness Verification and Performance Analysis of Real-Time Systems Using Stochastic Preemptive Time Petri Nets. IEEE Transaction on Software Engineering 31(11), 913–927 (2005)
3. Chang, C.K., Jiang, H., Di, Y.: Time-line Based Model for Software Project Scheduling with Genetic Algorithms. Information and Software Technology 50, 1142–1154 (2008)
4. Chen, J., Yang, Y.: Adaptive Selection of Necessary and Sufficient Checkpoints for Dynamic Verification of Temporal Constraints in Grid Workflow Systems. ACM Transaction on Autonomous and Adaptive Systems 2(2) Article 6 (2007)
5. Chen, J., Yang, Y.: Temporal Dependency based Checkpoint Selection for Dynamic Verification of Fixed-time Constraints in Grid Workflow Systems. In: Proceedings of 30th International Conference on Software Engineering, Leipzig, Germany, pp. 141–150 (2008)
6. Herroelen, W., Leus, R.: Project Scheduling Under Uncertainty: Survey and Research Potentials. European Journal of Operational Research 165, 289–306 (2005)
7. Hindi, K.S., Yang, H., Fleszar, K.: An Evolutionary Algorithm for Resource-Constrained Project Scheduling. IEEE Transaction on Evolutionary Computation 6(5), 512–518 (2002)
8. Jørgensen, M.: Evidence-Based Guidelines for Assessment of Software Development Cost Uncertainty. IEEE Transaction on Software Engineering 31(11), 942–954 (2005)
9. Jørgensen, M., Shepperd, M.: Systematic Review of Software Development Cost Estimation Studies. IEEE Transaction on Software Engineering 33(1), 33–53 (2007)
10. Law, A.M., Kelton, W.D.: Simulation Modelling and Analysis, 4th edn. McGraw-Hill, New York (2007)
11. Liu, X., Chen, J., Yang, Y.: A Probabilistic Strategy for Setting Temporal Constraints in Scientific Workflows. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 180–195. Springer, Heidelberg (2008)

12. Tracy, D.B., Howard, J.S., Noah, B.: Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems. Journal of Parallel and Distributed Computing 61(6), 810–837 (2001)
13. Wang, Q., Jiang, N.: Practical Experience of Cost/Schedule Measure through Earner Value Management and Statistical Process Control. In: Proceedings of 2006 International Software Process Workshop, Shanghai, China (2006)
14. Wang, Q., Jiang, N., Gou, L., Liu, X., Li, M., Wang, Y.: BSR: A Statistic-based Approach for Establishing and Refining Software Process Performance Baseline. In: Proceedings of 28th International Conference on Software Engineering, Shanghai, China, pp. 584–594 (2006)
15. Yang, Q., Li, M., Wang, Q.: An Algebraic Approach for Managing Inconsistencies in Software Processes. In: Wang, Q., Pfahl, D., Raffo, D.M. (eds.) ICSP 2007. LNCS, vol. 4470, pp. 121–133. Springer, Heidelberg (2007)
16. Zhang, D., Tsai, J.P.: Machine Learning Applications in Software Engineering. World Scientific, Singapore (2005)

Incrementally Introducing Process Model Rationale Support in an Organization

Alexis Ocampo[1], Jürgen Münch[1], and William E. Riddle[2]

[1] Fraunhofer Institute for Experimental Software Engineering,
Fraunhofer Platz 1, 67663 Kaiserslautern, Germany
{ocampo,muench}@iese.fraunhofer.de
[2] Solution Deployment Affiliates
223 N Guadalupe #313, Santa Fe, New Mexico, USA 87501
riddle@WmERiddle.com

Abstract. Popular process models such as the Rational Unified Process or the V-Modell XT are by nature large and complex. Each time that a new release is published software development organizations are confronted with the big challenge of understanding the rationale behind the new release and the extent to which it affects them. Usually, there is no information about what has changed or most importantly why. This is because of the lack of a flexible approach that supports organizations responsible for evolving such large process models in documenting their decisions and that reflects the extent of the capabilities to which they can provide this information. This paper describes an approach to incrementally deploying rationale support as needed to match an organization's needs, the capabilities and interests of the organization's process engineering teams, and the organization's willingness to support the effort required for the collection and application of the rationale information.

Keywords: rationale conceptual models, rationale capture and application methods, incremental method deployment, REMIS.

1 Introduction

Software process models support software engineers in systematically performing the engineering processes needed to develop and maintain software products. As these processes are enacted, suggestions and needs for adjustment or refinement arise, which, in turn, demands an evolution of the models. Changing these models in an organization is typically a complex and expensive task. In many cases, due to budget and time constraints, arbitrary decisions are made, and process models are evolved without storing or keeping track of the justification behind such changes. This frequently results in inconsistencies or ambiguity being introduced into the process models.

The work presented in this paper responds to the need for systematically performing changes to a process model by contributing an approach for rationale support of process model evolution called "REMIS". REMIS guides process engineers in the tasks of capturing the reasoning (i.e., rationale) behind such changes and analyzing the evolution. REMIS has been developed in a bottom-up fashion based on observations and experience from different case studies (industrial projects). The main

Q. Wang et al. (Eds.): ICSP 2009, LNCS 5543, pp. 330–341, 2009.

contribution of REMIS to current research in the field of process model evolution consists of transferring and adapting design rationale concepts in order to support systematic process model evolution. Fig. 1 shows the specific contributions which are: a) a conceptual model for describing the rationale for software process model changes, b) a method for capturing and analyzing the rationale for software process model changes, c) a classification of common situations for process model change, d) a tool that supports the method and e) an incremental deployment strategy. Previous publications describe in detail the contents of a), b), c) and d). This paper describes e), the method-deployment strategy.

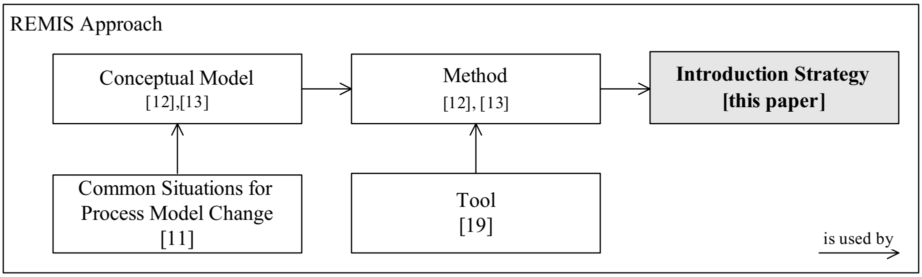

REMIS Approach
Conceptual Model [12],[13]
Method [12], [13]
Introduction Strategy [this paper]
Common Situations for Process Model Change [11]
Tool [19]
is used by

Fig. 1. The REMIS Approach

Why is a strategy so important and necessary for introducing the REMIS approach into an organization? One special characteristic of rationale approaches is their degree of intrusiveness in the modeling process. That is, the extent to which the approach interferes with the modeling process. Such interference can happen not only during the capture of rationale but also during the retrieval of this rationale. The reason to highlight this characteristic is based on a long-term discussion about the costs and cultural implications of capturing rationale information in the rationale research and practitioner communities [5]. Although the approaches are in constant maturation, the resistance of practitioners to capture rationale information has been associated with intrusiveness. More intrusive approaches need a stronger accompanying process in order to be successful than less intrusive ones [17]. Therefore the deployment strategy presented in this paper is oriented toward mitigating this intrusiveness risk based on the assumption that a process engineering team must capture and apply information about process modeling decisions and their rationale according to the team's needs, capabilities and allocated-effort.

Two case studies used as input for the definition of the deployment strategy – the central part of this paper – are presented in the Section 2. This is followed, in Section 3, with a brief accounting of the requirements for a conceptual model, an associated rationale documentation method, and a deployment strategy addressing the problems revealed by the case study. Section 4 then discusses in brief the conceptual model and the method. Section 5 describes in detail the strategy for incrementally deploying the REMIS approach as needed by a process engineering team and as allowed by their capabilities. The paper ends first – in Section 6 – with a discussion of how the conceptual model, the method and deployment strategy satisfies the requirements given in Section 3, and then – in Section 7 – a summary of the work presented in this paper and a discussion of how the REMIS approach, its underlying conceptual model, and the deployment strategy might be improved through further research and development.

2 Case Studies for Eliciting Requirements on Rationale for Process Model Evolution

This section presents in brief, the experience captured in two different case studies in which rationale information was collected while evolving large and complex process models. A more detailed description of each case study, i.e., the study's definition, design and results can be found at [20]. The observations of the first case study served as inputs for its application in the second case study. The conceptual models used in both iterations and observations on the feasibility of their use, constituted the basis for the requirements of the REMIS approach and the deployment strategy.

2.1 ESA Case Study

For the European Space Agency (ESA), the relevant standards applicable for developing software are: ECSS-E-40B Space Engineering - Software [6] (mostly based on the ISO 12207 standard [8]), and ECSS-Q80-B: Space Product Assurance - Software [7]. Organizations or projects that are part of ESA are required to develop and use specific tailorings of the ECSS standards suited to their work. This is a particularly complex task because it requires detailed understanding of the whole standard, something that an average software developer or project manager usually does not have [16]. At the ESA Space Operations Center ESOC (the ESA organization where this project took place), this tailoring was called the Software Engineering and Management Guide (SEMG) [9] and was used for all their major projects.

After several years of experience with the ECSS standards, these were revised by ESA, and a new version was published. This also meant that the SEMG had to be revised, in order to be compliant with the revised ECSS standard. This compliance had to be proven by means of traceability of every ECSS requirement to its implementation and by providing a tailoring rationale for every tailored requirement.

The goal of this study was to analyze the feasibility of the conceptual model and the approach for documenting and analyzing the rationale for process model change.

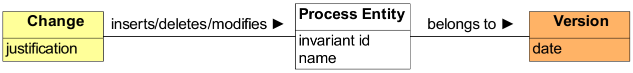


Fig. 2. Conceptual Model - ESA Case Study

Fig. 2 shows the conceptual model. *Changes* result from decisions captured in the justification and are performed on Process Entities. Some examples of Changes performed to Process Entities are: *Entity x has been inserted*; *Entity y has been deleted*; and *Entity x has been modified*. A *Process Entity* reflects a concept defined by a vocabulary/notation for modeling/describing process models, e.g., SPEM [14], V-Modell XT [10], SPEARMINT [4] and BPML [15].

The data about the changes to the SEMG were collected in meta-information tables attached to each section. Process engineers provided information about the rationale for a change each time a change was performed to the SEMG standard. Then they used an automated mechanism for storing this information in a database [12].

Observations: The tables that were used by process engineers for describing what changed and why were very useful for systematic reviews. However, sometimes the provided information about what changed was too detailed, sometimes too abstract. This might be due to the fact that the conceptual model did not anticipate a difference between finely granular changes (e.g., grammar errors or misspellings) and larger ones (e.g., wrong control flow). The lack of structure of a justification influenced the understandability of the collected information. The ESA reviewers commented on confusing justifications that identified what was performed instead of information on why. ESA reviewers also missed information concerning the alternatives taken into account by process engineers before performing the change. That information could have help reviewers understand faster the rationale and avoid unnecessary discussions. These findings motivated the need to change the conceptual model and the instrumentation and to use it in a second case study.

2.2 ASG Case Study

In this case study, process engineers were in charge of defining, establishing, evaluating, and systematically evolving the development process model applied in the project to develop a platform for Adaptive Services Grid (ASG) [1].

In general, the main idea behind the systematic evolution approach followed was to start with commonly accepted process knowledge, refine it with information gathered from the practitioners, and therewith improve the textual descriptions and diagrams of the process according to the real project needs. The conceptual model and the method developed in the ESA case study [2] were extended as well as the tool-assisted way of editing and storing the process model and its rationale information.

The goal of this study was to analyze the feasibility of the refined conceptual model and the refined approach for documenting and analyzing the rationale for process model changes.

As in the previous case study, meta-information tables were used as a means for realizing the conceptual model or the rationale for process model changes [12]. In practice, the process engineer discussed and resolved the issue while introducing the corresponding rationale information, then performed the changes to the respective process entities, and finally established a reference to the corresponding rationale concept.

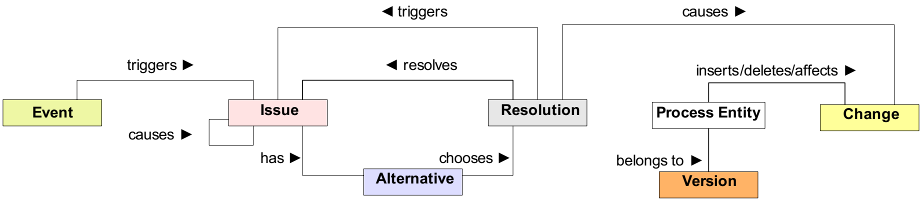


Fig. 3. Conceptual Model - ASG Case Study

In the extended conceptual model (shown in Fig. 3), an *Event* is a trigger of issues. Events may be characterized by a name and a short description (i.e., two of its attributes may be *name* and *short_description*). Events may also be characterized by their *type*. At least two types of events are possible: internal (e.g., new/updated corporate policies, e.g., policy changes stemming from changes to an organization's business goals) and external (e.g., new/updated software engineering technology, e.g., new testing support tools and techniques). *Issues* are situations that arise as a consequence of an Event, that need to be addressed, and that are related to a part of the described system. Additionally, an Issue may be categorized by its type. This type may be selected from a classification of issues pertinent to an organization. At this point, REMIS reflects a general, organization-independent classification of issues resulting from the ESA case study (i.e., imprecision; verbosity; inaccuracies; non-compliance; inconsistency).

Issues are often stated as *questions*. The question has the purpose of forcing software engineers to reason about the situation (the problem) they are facing before starting to think about or providing resolutions (the potential solutions). Some of the characteristics of an issue are a *synoptic_description*, a *status* (e.g., open, closed), and a *detailed_discussion*. The *detailed_discussion* may be used to capture the minutes of a meeting, E-mails, memos, letters, etc. in which the issue was discussed by software engineers or stakeholders.

Alternatives are assigned to an Issue; at least one Alternative might be proposed to resolve an Issue. Alternatives are described at least by a *subject*, and more comprehensively in a *description*. The assessment describes the acceptance of the alternative with respect to the characteristics pertinent to evaluating its achievement, e.g., its feasibility, cost and required-effort. Usual values are positive or negative. A *Resolution* changes the process model. A Resolution might lead to opening more Issues. Every Resolution is characterized by a *short_description*, a *long_description*, a *justification*, and a *status* (for example, open or closed). The justification is included to be able to capture a summary of the analysis of the different alternatives as well as a short note.

Observations: The extended conceptual model played an important role, because it allowed structuring better the reasoning behind a decision compared to the previous study. Especially concerning the alternatives taken into account. Having this information motivated self-reflection about the real need for changing the process model. Equally, the structure of the conceptual model allowed reusing this information in a straightforward way, before performing future changes. The types of events and issues provided a means for starting up a classification of common situations for process model change.

3 Requirements for a Rationale for Process Model Changes

The case studies revealed that the fundamental need is to collect rationale information which facilitates making and justifying design decisions underlying the process model's evolution in response to changes to its requirements or its operational context. This includes information about the alternatives which were considered and the rationale underlying the adoption or rejection of the various alternatives. Further, the case studies indicate that it is important that this information not merely captures low-level, "micro" details but that it be possible to integrate over the detailed information to provide information at the higher, "macro" pertinent to process model issues. Therefore, the basic requirement is:

- R1: Support the collection of information which may be directly used, or may be interpreted and analyzed, to understand alternatives, choose among them and justify their choice or rejection as necessary.

The case studies also indicate that the effort required to collect the information should be acceptable, as "minimal" as possible. Partially, this means that the planning of process model evolution activities account for the fact that some effort will be needed; an evolution plan must include an allocation of effort for collecting and applying rationale information. The information collection and application effort must be an acceptable increase over the effort needed for the evolution activities. Doubling the effort would obviously be unacceptable. Based on the authors' experiences, a 33% increase is probably an upper-bound, with the increase normally being in the range 15-20% with larger increases only when justifiable, for example when the system will undergo extensive independent review.

Accommodating a restriction such as this upon the information collection and application effort requires supporting the effort with at least guidelines – and, even better, guideline-implementing tools and techniques – that enhance the software engineers' abilities. It also requires the ability to customize the guidelines, tools and techniques to both enhance the pertinence, and therefore value, of the information and eliminate effort 'wasted on' the collection of unnecessary information.

This leads to two additional requirements:

- R2: Provide guidelines helping software engineers efficiently as well as effectively carry out rationale information collection and application tasks. When possible, provide tools (e.g., information templates) and techniques (e.g., decision making-support approaches) which implement the guidelines and reduce the effort needed to follow the guidelines.
- R3: Allow customization of a 'default' set of guidelines, tools and techniques serving to match the needs for specific process model evolution activities.

The final requirement is not directly revealed by the case studies. Rather, it comes from further considering the issue of customization. Requirement R3 reflects the need to customize with respect to the nature of the process model being evolved. The final requirement reflects a need to customize the guidelines, tools and techniques to match the abilities of an organization's software engineering, their tolerance for carrying out 'overhead' tasks, and the organization's willingness to support the extra effort needed to collect and apply rationale information (i.e., the organization's tolerance for effort increases needed to collect and apply the information). This requirement is:

- R4: Allow the incremental adoption of the guidelines, tools and techniques in steps of increasing scope, depth, difficulty, effort and value.

4 The REMIS Approach

The final conceptual model underlying REMIS, shown in Fig. 4, results from the incremental, iteration-driven research strategy (based on the previously described case studies) aimed at understanding the information needs for capturing the rationale underlying change to a process model.

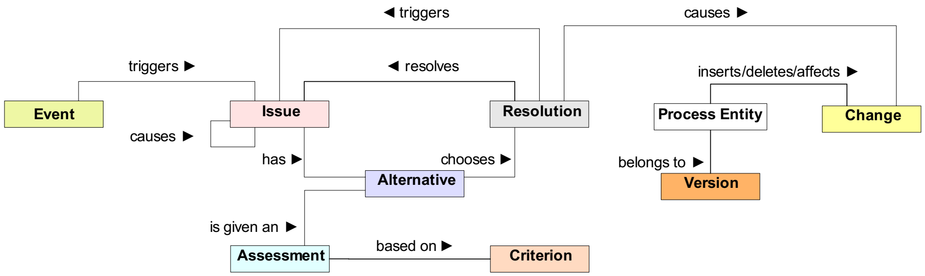


Fig. 4. REMIS Conceptual Model

It can be seen in Fig. 4 that compared to the conceptual model described in the ASG case study, that the following additional concepts have been introduced. An Alternative's *Assessment* is based on criteria. A *Criterion* is an influencing factor pertaining to a given organization in a certain context. A set of Criteria characterize the context in which changes are made. Criteria are important not only for comparatively assessing Alternatives, but also for recording evidence of the most influential factors that affect a decision. In the software design domain there is a noticeable lack of research regarding the Criteria affecting the assessment of design alternatives. For lack of a better approach, at this point in time the REMIS approach relies up the GQM paradigm [3] to dynamically, case-by-case, define the Criteria that affect an organization's software process modeling evolution efforts. This paradigm explicitly includes weights reflecting the importance of Criteria for an organization in a given evolution context with respect to other Criteria. Finally, every Resolution identifies changes which satisfy the various Criteria.

The method provided by REMIS (see Fig. 5) is also based on experiences from the case studies and well supported by the conceptual model.

The following paragraphs discuss briefly the purpose and description of the method’s product flow (for a more detailed discussion, please refer to [12]).

The purpose of the activity *Analyze change request* is to understand, assess, and prioritize the feedback provided by engineers or practitioners concerning the process model. Rationale visualization can be useful at this point for answering different types of questions relevant to process engineers. Examples of such questions are:

- Which still-open issues may conflict with the change/improvement proposal being analyzed?
- Which process model entities are affected by a previous resolution that conflicts with the new change/improvement proposal being analyzed?

The proposals are then prioritized. The process engineer selects those improvement proposals that should be considered according to the prioritization. Additionally, the process engineer decides whether the rationale should be elicited *synchronously (i.e., while performing the changes)* or rather *asynchronously (i.e., after performing the changes)*. The decision should normally be based on factors such as (a) relevance of the change/improvement proposal; (b) available resources; (c) available infrastructure; and (d) degree of maturity in eliciting the rationale.

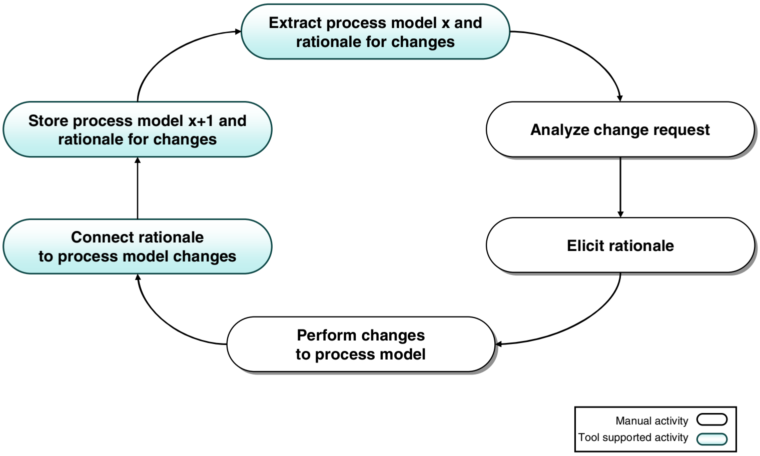

Extract process model x and rationale for changes
Analyze change request
Elicit rationale
Perform changes to process model
Connect rationale to process model changes
Store process model x+1 and rationale for changes
Manual activity
Tool supported activity

Fig. 5. REMIS Method

The activity *Elicit rationale* consists of the process engineer analyzing and discussing with other stakeholders (e.g., project manager, quality manager) the change improvement proposals and deciding on a strategy for implementing the resulting changes. The reasoning behind the decision is captured during the analysis and/or discussions. Existing rationale information (that explains the evolution of the process model up to that moment) can support the process engineer in this activity. Rationale visualization can be useful again at this point for answering different types of questions.

Once the process engineer is sure about what changes to perform, she/he proceeds to the activities *Perform changes to model entities* and *Connect rationale to process model changes*. The process engineer then will implement the agreed-upon changes to the set of process model entities by using the specific process modeling tool used in his/her organization. In order to connect the rationale to the just performed process model changes, the process engineer can use two different techniques: one that mimics the technique used in the case studies and proposes inserting references to the rationale information directly into the process model entity being altered [12]. A second one that consists of after performing the changes (i.e., *asynchronously*) identifying the set of changed process model entities (by means of an special technique for comparing models called Delta-P [18]) and inserting a reference to the respective rationale for each one of those changes [13].

The purpose of the activity *Store new process model version x+1 and rationale for changes* is to make persistent the changes performed to a model and to annotate the model with a new version identifier. The process model evolution repository consists of a body of content formed by process model entity instances of a well-defined metamodel and the rationale information. The activities *Connect rationale to process model changes* and *Store new process model version x+1 and rationale for changes* are supported by the REMIS tool [19] in order to systematically keep the consistency between the different versions of the process models and its rationale.

5 The Incremental Introduction Strategy

The capture and visualization of the rationale for process model evolution must be accomplished in a systematic manner. Convincing an organization to change the way it works or to adapt to a new mechanism is a complicated task. Therefore – and based on the experiences of the case studies reported in [2], [12] and [13] – a staged incremental method, which facilitates the institutionalization of rationale and visualization into a software development organization, is proposed. This means that organizations have to incrementally learn how to collect rationale, what to collect, how to use it, and, especially, they have to identify which level of "maturity" in rationale-driven evolution they want to achieve. Fig. 6 presents the different steps defined for incrementally introducing and institutionalizing rationale support for process model evolution. One advantage of using the RDF notation [21] as a basis for the specification of the conceptual model is the possibility of incrementally adding concepts to the rationale vocabulary. This facilitates gradual introduction as well as the design and implementation of tool support. The following paragraphs provide a more detailed description of the activity for eliciting process model rationale, highlighting the differences for different levels of deployment - i.e., REMIS 0, 1, 2 and 3 - identified in Fig. 6.

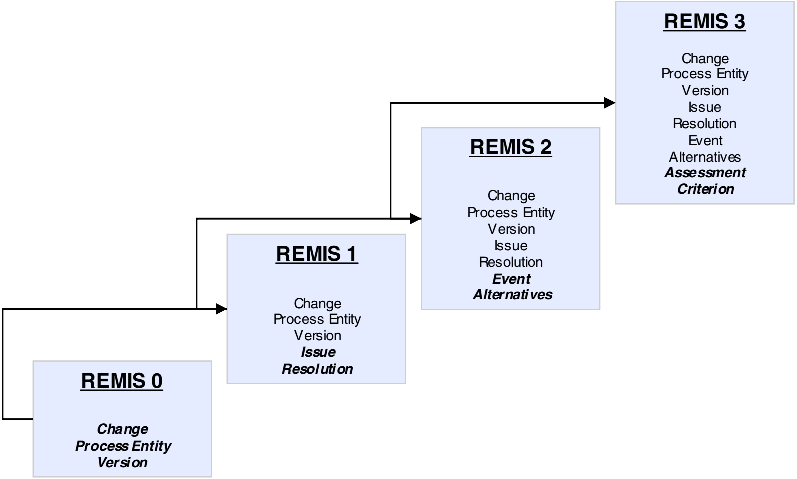


Fig. 6. Incremental Strategy

The purpose of the REMIS 0 level is to capture the basic justification for changes to process model entities that belong to a given model version. At this level, rationale information only consists of the justification for a change. This can be a short description of the reason for performing a change.

The purpose of the REMIS 1 level is to capture the basic structuring of the reasoning behind a decision. At this level, rationale information consists of the issues and the respective resolutions that generate changes. This information can be found in organizations that use any sort of problem/resolution management process. Usually such processes are supported by a bug tracking systems where this information is captured [13].

The purpose of the REMIS 2 level is to capture the elaborated reasoning of a decision. At this level, rationale information consists of the events, the issues and their respective alternatives, and the resolutions that generate changes. Information about the alternatives cannot be found in organizations that use typical problem/resolution processes/tools, because they are not equipped to collect this kind of information. This is the reason why collection of this information is optional. However, the collection of alternatives is important for organizations because they reveal the style or preferences of the teams in charge of evolving the model. Alternatives that were not taken into account are especially important in those cases where knowledge about the application domain is minimal because the description of these alternatives offers the opportunity to retrospectively consider what should be or what should not be done in the future.

The purpose of the REMIS 3 level consists of understanding the influence of criteria on a decision. This is the highest level. In it, the most comprehensive rationale information is collected. Rationale information consists of the events, the issues, the respective alternatives, the criteria taken into account for assessing alternatives, and, finally, the resolutions that generate changes. Eliciting criteria and assessing them are optional activities.

Definition of the criteria varies from project to project. External definitions of criteria can also be incorporated into the project definition.

6 Fulfillment of the Requirements

The focus – its underlying rationale – for the REMIS approach is upon satisfying requirement R4 (Allow incremental adoption of the guidelines, tools and techniques). Four levels of change information capture and application are described in Section 5. These allow organizations to initially make a minimal investment in, and incur a minimal impact for, tracking changes to a system so that the purpose of individual changes may be explained and argued, and previously considered, but rejected, changes may be effectively and efficiently re-considered. As an organization's needs and capabilities to track changes increase, and its willingness to incur the impacts increases, the organization may move to more expansive 'levels' of the REMIS approach. The levels are defined to support the gradual and smooth introduction of capability as it has been observed in practice.

Unlike previous rationale conceptual models [5], the REMIS conceptual model is defined to allow incremental expansion of attention to information from, first, basic information regarding the changes made at level REMIS 0 to, ultimately at level REMIS 3, information regarding not only the changes but also the events precipitating them, the alternative changes that were considered, and the rationale underlying the choice of the change that was made. This depiction emphasizes the fact that the underlying conceptual model allows 'expansion upon demand', in other words: expansion of the conceptual model as needed to meet an organization's needs for the capture and application of change rationale information and its tolerance for the impact upon its system development efforts. The underlying conceptual model therefore not only satisfies requirement R1 but also satisfies this requirement with a conceptual model which is considerably better – more flexible and incremental – than previously-developed models.

The REMIS approach also satisfies requirements R2 and R3. It provides techniques and supporting tools that support an organization's capture and application of process model rationale information. These techniques and tools have been defined as a result of several exercises in a variety of industrial projects. They are available to organizations which have an interest in applying the REMIS approach to rationale capture and application. And they will evolve through their future application to various situations.

7 Summary and Outlook

This article presents an approach - based on requirements that were derived from observing development and maintenance practices in industry - to incrementally deploying process model rationale support. In addition, the underlying REMIS approach is described that consists of a flexible conceptual model and an associated method, both supporting the effective and efficient collection and application of information about a process model's design alternatives and their selection rationale. REMIS is based on several extensive software process change exercises in industry.

Summarizing our experience with deploying rationale support we have observed that organizations should deploy rationale concepts incrementally and that this deployment process might take quite long (up to several years). The approach described in this article can be seen as a good basis and applications of the approach indicate that rationale support provides significant contributions to the expected higher-level benefits (such as reduction of evolution cost).

Based on experience with developing the method and introducing it in industry. several open questions and research directions have been identified. A selection of these topics that might be subject to future work is the following:

- What are suitable techniques for integrating and aggregating rationales to provide support for higher-level understanding and decision making?
- How to visualize the history of process models in a way that the history can be easily explained with the help of the rationale?
- How to demonstrate the value of rationale support to the higher-level goals of an organization?

Acknowledgments. We would to thank Sonnhild Namingha from Fraunhofer IESE for proofreading this paper.

References

[1] Adaptive Services Grid Project, "ASG", http://asg-platform.org/cgi-bin/twiki/view/Public

[2] Armbrust, O., Ocampo, A., Soto, M.: Tracing Process Model Evolution: A Semi-Formal Process Modeling Approach. In: Oldevik, J., Aagedal, J. (eds.) ECMDA - TW 2005, pp. 57–66 (2005)

[3] Basili, V., Weiss, D.: A Methodology for Collecting Valid Software Engineering Data. IEEE Transactions on Software Engineering 1984, 728–738 (1984)

[4] Haman, D., Kempkens, R., Rösch, P., Verlage, M., Webby, R., Zettel, J., Becker-Kornstaedt, U.: Support for the Process Engineer: The Spearmint Approach to Software Process Definition and Process Guidance. In: Jarke, M., Oberweis, A. (eds.) CAiSE 1999. LNCS, vol. 1626, p. 119. Springer, Heidelberg (1999)

[5] Dutoit, A.H., McCall, R., Mistrík, I., Paech, B.: Rationale Management in Software Engineering, http://dx.doi.org/10.1007/978-3-540-30998-7

[6] Space engineering Software - Part 1: Principles and requirements. ESTEC, P.O. Box 299, 2200 AG Noordwijk, The Netherlands: ESA Publications Division, November 28 (2003)

[7] Space product assurance - Software product assurance. ESTEC, P.O. Box 299, 2200 AG Noordwijk, The Netherlands: ESA Publications Division, October 10 (2003)

[8] Systems and software engineering – Software life cycle processes: ISO, March 18 (2008)

[9] Jones, M., Gomez, E., Mantineo, A., Mortensen U.K.: Introducing ECSS Software-Engineering Standards within ESA. Practical approaches for space- and ground-segment software, ESA bulletin 111 (August 2002), http://www.esa.int/esapub/bulletin/bullet111/chapter21_bul111.pdf

[10] V-Modell XT, http://www.kbst.bund.de/cln_012/nn_999710/Content/Standards/V__Modell__xt/v__modell__xt__node.html

[11] Ocampo, A., Münch, J.: Process evolution supported by rationale: An empirical investigation of process changes. In: Wang, Q., Pfahl, D., Raffo, D.M., Wernick, P. (eds.) SPW/ProSim 2006. LNCS, vol. 3966. Springer, Heidelberg (2006)

[12] Ocampo, A., Münch, J.: Rationale modeling for software process evolution. In: Software Process: Improvement and Practice, June 12, pp. 1077–4866 (2008), http://dx.doi.org/10.1002/spip.387

[13] Ocampo, A., Soto, M.: Connecting the Rationale for Changes to the Evolution of a Process. In: Münch, J., Vierimaa, M. (eds.) PROFES 2006. LNCS, vol. 4034. Springer, Heidelberg (2006)

[14] Software Process Engineering Metamodel Specification, 2nd edn. (January 2005), http://www.omg.org/technology/documents/formal/spem.htm

[15] Business Process Modeling Notation Specification (February 2006), http://www.omg.org/technology/documents/br_pm_spec_catalog.htm#BPMN

[16] Ponz, D., Spada, M.: Three Years of ECSS Software Standards: An Appraisal and Outlook: OPS-G Forum, January 20 (2006)

[17] Shum, S., Selvin, A., Sierhuis, M., Conklin, J., Haley, C., Nuseibeh, B.: Hypermedia Support for Argumentation-Based Rationale. Rationale Management in Software Engineering, pp. 111–132

[18] Soto, M., Münch, J.: Focused identification of process model changes. In: Wang, Q., Pfahl, D., Raffo, D.M. (eds.) ICSP 2007. LNCS, vol. 4470, pp. 182–194. Springer, Heidelberg (2007)

[19] Ocampo, A., Münch, J.: The REMIS approach for rationale-driven process model evolution. In: Wang, Q., Pfahl, D., Raffo, D.M. (eds.) ICSP 2007. LNCS, vol. 4470, pp. 12–24. Springer, Heidelberg (2007)

[20] Ocampo, A.: The REMIS Approach to Rationale-based Support for Process Model Evolution. PhD thesis, University of Kaiserslautern, PhD Thesis in Experimental Software Engineering, vol. 25 (2009)

[21] Manola, F., Miller, E. (eds.): RDF Primer. W3C Recommendation (2004), http://www.w3.org/TR/rdf-primer/ (last checked 2009-02-20)

A Process for Driving Process Improvement in VSEs

Francisco J. Pino[1,3], Julio Ariel Hurtado Alegría[1], Juan Carlos Vidal[2], Félix García[3], and Mario Piattini[3]

[1] IDIS Research Group – Electronic and Telecommunications Engineering Faculty
University of Cauca, Street 5 # 4 – 70 Popayán, Colombia
{fjpino,ahurtado}@unicauca.edu.co
[2] Faculty of Engineering and Business
University of Ciencias de la Informática, Pedro de Valdivia 450, Santiago, Chile
jcvidal@ucinf.cl
[3] Alarcos Research Group – Institute of Information Technologies & Systems
University of Castilla-La Mancha, Paseo de la Universidad, 4, 13071, Ciudad Real, Spain
{Felix.Garcia,Mario.Piattini}@uclm.es

Abstract. A success factor in Software Process Improvement –SPI- in very small enterprises –VSEs- is that improvement effort must be guided and managed by means of specific process. Nonetheless, many proposals related to this issue have not considered that type of process explicitly. So, aiming to establish SPI in VSEs systematically and coherently, we have defined a light process for managing and leading the improvement process step-by-step, called PmCOMPETISOFT. This paper introduces that process, which guides the implantation of an improvement cycle in an iterative and incremental manner. It also describes our experience of the application of the proposed process in four VSEs, through case studies. The results of the case studies show that the companies increased the capability of their processes, and that it is feasible to implement this process in this type of organizations, by investing an effort which corresponds to the particular characteristics of each.

Keywords: SPI, VSEs, improvement process, case studies.

1 Introduction

If we are to carry out a software process improvement –SPI- initiative in an organization, it is necessary to take several proposals into account (models, methods or standards) so that we may: (i) have good and available practices for software development (processes reference model), (ii) determine the state of the processes and discover opportunities for improvement (process assessment method) and (iii) direct the process improvement activities towards the innermost part of the organization (model to guide SPI). Although SPI proposals are available to all enterprises, many very small software enterprises –VSEs- (firms with fewer than 25 employees, according to [1]) do not use these proposals. Some of the reasons for this phenomenon include the following: the organizations remain unaware of these methodologies [2] and the proposals are difficult to apply to these organizations due to the large investment of time, money, and resources involved in an improvement project [3].

Q. Wang et al. (Eds.): ICSP 2009, LNCS 5543, pp. 342–353, 2009.

According to [4], in those small companies which use process improvement models, the one which *model to guide SPI* is that which is used least. It is also the case that many national and international proposals related to SPI in small software organizations have not given explicit consideration to *model to guide SPI*. This is a great drawback, since this type of model for guiding SPI provides the guidelines that are needed to organize all the activities related to process improvement (including the process reference model and the process assessment method). We should also point out that one success factor for SPI in VSEs is that the improvement effort be guided by means of specific procedures and the combination of different approaches, following a systematic and coherent initiative [4].

In response to the situation outlined above, in the methodological framework for SPI in VSEs created in the COMPETISOFT Ibero-American project [5], great importance is given to the model for guiding SPI activities. This is due to the fact that this project maintains that if we are to help small companies set up and pursue the path towards process improvement, then a guideline of this type is needed (including greater depth of detail, the way in which the process becomes integrated into other components of the methodological framework, and its suitability with regard to the company's particular characteristics and needs). Given all this, one of the components of the methodological framework is a specific *framework for guiding activities of SPI* (*improvement framework*), see Fig. 1. This *improvement framework* defines four components, but this article focuses on a presentation of PmCOMPETISOFT, which is a process for the establishment of process improvement in small software organizations. It aims to improve the processes in the organization in a systematic and coherent way, in line with the company's own specific business goals. A description of the application of this process in four case studies is also given in this paper.

The paper is structured as follows. The next section presents related works. The COMPETISOFT improvement framework and its PmCOMPETISOFT process are then described. Section 4 describes the application of this process in four case studies. Finally, conclusions and future work are set out.

2 Related Work

Several proposals exist which present a set of processes that small companies could use to attain significant benefit from process improvement. These include MoProSoft [6] (which proposes 6 processes based on ISO 12207, CMM, ISO 9001), MPS.BR [7] (which proposes 23 processes based on ISO 12207 and CMMI), Adept [8] (which proposes 12 processes based on CMMI), Rapid [9] (which proposes 8 processes based on ISO 15504:1998), among others. All of these proposals are related to assessment methods or process reference models and all of them define a group of processes that should be taken into account by small companies in their improvement efforts.

With regard to research on models that direct improvement implementation for small companies, several proposals have emerged in recent years. These include, amongst others: (i) IMPACT [10], which is based on the idea that the process is an abstraction of the practices carried out in many different projects by many different people; (ii) MESOPyME [11], which has as its focal point the reduction of time and effort in the implementation of SPI by using the concept of action packages as a base; (iii) The application of the IDEAL model to small and medium enterprises such as

[12] and [13]; and (iv) PROCESSUS [14], based on the process modeling paradigm, in which each procedure is dealt with as a process, which is defined, established, implemented and maintained.

The contribution of the proposal described in this paper is to present an explicit process which will be a step-by-step guide to the implementation of process improvement, and which small software organizations will be capable of taking on. This process constitutes the backbone of the improvement framework. The improvement framework describes four components which have been defined by taking into account: (i) widely recognized frameworks, such as ISO/IEC 15504-4 [15], IDEAL and SCRUM; and (ii) special characteristics of the VSEs, as presented in [2] and [4]. These components describe tailored and integrated improvement practices, aiming to offer to the VSEs a framework which is useful and practical for addressing SPI. Furthermore, according to [4] the strategies that have been used to SPI on VSEs are diverse and include: adaptation and use of SPI models, establishment of software processes to guide the SPI efforts, prioritization of the SPI efforts and evaluation of a SPI programme. Only the improvement framework addresses these improvement strategies in an integrated manner, and the component integrator is PmCOMPETISOFT.

3 Improvement Framework

Fig. 1 shows the three elements of the COMPETISOFT methodological framework and displays the four components of its *improvement framework.*

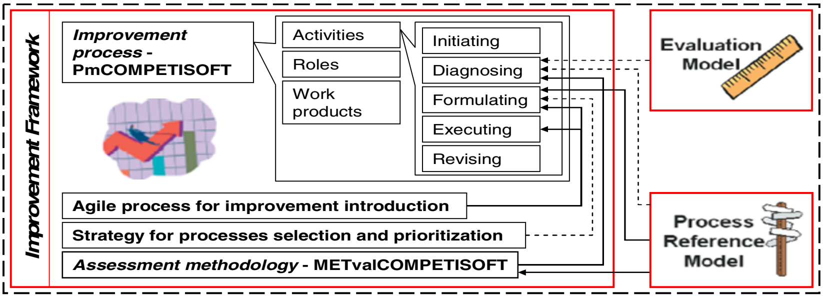

Improvement Framework
Improvement process - PmCOMPETISOFT
Activities
Roles
Work products
Initiating
Diagnosing
Formulating
Executing
Revising
Evaluation Model
Process Reference Model
Agile process for improvement introduction
Strategy for processes selection and prioritization
Assessment methodology - METvalCOMPETISOFT

Fig. 1. Methodological framework of COMPETISOFT

The *COMPETISOFT improvement framework* defines: (i) A process, called PmCOMPETISOFT, with which to manage and lead the software improvement process step-by-step; (ii) An agile process for improvement introduction, which uses the SCRUM agile method to support the managing and carrying out of the activities of the formulation and execution of improvement; (iii) A strategy for process selection and prioritization, which presents the selection of a set of processes that are considered critical to the implementation of a process improvement project in small companies [16]; and (iv) A methodology for software process assessment, called METvalCOMPETISOFT, which supports the activity of diagnosing the software processes in small organizations.

The agile process, the strategy for process selection and prioritization, and methodology for software process assessment are outside the scope of this article, which is focused on the description of the PmCOMPETISOFT process.

3.1 The PmCOMPETISOFT Process

This section provides a detailed description of PmCOMPETISOFT, which plans to satisfy the following principles: (i) Early and continuous achievement of improvements, (ii) Continuous and rapid process diagnosis, (iii) Elemental process measurement, (iv) Effective group collaboration and communication, and (v) Continuous learning. This process is influenced by the ISO/IEC 15504-4, IDEAL and SCRUM models. From these, we have analyzed, integrated and tailored several improvement practices, in order to offer a specialized and suitable guide which will meet the needs of the VSEs when leading SPI. In this sense, PmCOMPETISOFT is described in terms of purpose, objectives, roles, activity diagram, activities, work products, and tools support, according to the process pattern established by COMPETISOFT. Due to space restrictions, we have described only some of these elements, but in [17] a complete description of PmCOMPETISOFT is presented.

Activity diagram. Fig. 2 shows the PmCOMPETISOFT activity diagram, which uses SPEM 2.0 notation and includes roles, activities and work products.

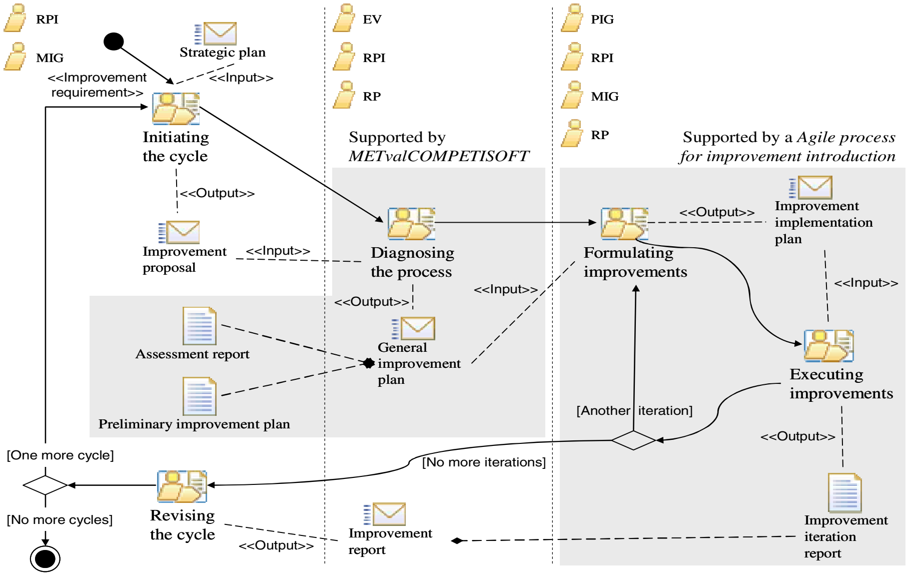


Fig. 2. PmCOMPETISOFT Activity Diagram

Roles. The roles involved in PmCOMPETISOFT are: Management Improvement Group (MIG), Responsible for process improvement (RPI), Process Improvement Group (PIG), Responsible for process or Participant (RP), Evaluator (EV). It is important to consider both that one employee may play various roles and that a single role can be played by several employees.

Activities. The continuous improvement process is made up of one or more improvement cycles. Each improvement cycle consists of 5 activities: Initiating the cycle, Diagnosing the process, Formulating improvements, Executing improvements and Revising the cycle. These activities are presented below:

- *Initiating the cycle:* the person *Responsible for the process improvement* and the *Management Improvement Group* create an *Improvement Proposal* which is aligned to the organization's strategic planning as laid out in the *Strategic Plan.* This proposal guides the organization through each of the following activities of the cycle. The proposal must be approved by the *Improvement Management Group* (MIG) if the assignation of the necessary resources is to be guaranteed.
- *Diagnosing the process:* the *Evaluator* and the person *Responsible for process improvement* carry out the process assessment activity (internal evaluation) in order to discover the general state of the organization's processes and to analyze the results. The objective is both to establish opportunities to improve a process (improvement cases) and to define their improvement priority. The improvement priority permits them to define the order in which the iterations will take place. Preliminary and general planning for the improvement cycle is carried out. The information related to this activity is registered in the *General Improvement Plan.*
- *Formulating improvements:* the *Process Improvement Group* validates the *General improvement plan.* This group plans and designs the improvement cycle's current iterations (based on the process improvement cases) and defines the strategy to be followed to improve the process which has been selected. The effort required in the first iteration is used as a basis for, amongst other things, the estimation of effort, cost, time and resources in the other iterations of the improvement cycle. The information related to this activity is registered in the *Improvement Implementation Plan.* This activity can be executed in the cycle once or various times.
- *Executing improvements:* the *Process Improvement Group* manages and executes the improvement cases which correspond with the current iteration, in accordance with the established plans. If the plan of the iteration has been satisfactorily developed, it is accepted and the new processes or changes are established within the organization. The information related to this activity is registered in the *Improvement iteration report,* which is part of the Improvement Report. This report describes the performance and evaluation of the current iteration and also analyzes the improvements introduced into the organization's processes. This activity can be executed once or several times in the improvement cycle.
- *Revising the cycle:* all the elements related to the execution of each of the improvement cycles are corrected or adjusted. Finally, a post-mortem analysis of the work carried out in the entire improvement cycle takes place. The person *Responsible for Process Improvement* (RPI) reinforces the improvement cycle which has been carried out before reinitiating the installation phase of a new cycle. The lessons learnt, measurements developed to measure the fulfillment of the objectives, the processes improved, etc. are registered in the *Improvement Report.*

When more explicit guidelines are required to support the *Diagnosing the process* activity, VSEs can use the *Assessment methodology – METvalCOMPETISOFT.* The *Agile process for improvement introduction* can similarly be used to guide the *Formulating and Executing improvements* activities (see Fig. 1 and Fig. 2).

Work Products. A concrete self-contained template has been developed for each of PmCOMPETISOFT's work products, to make its construction easier. These work products are: Improvement proposal, General improvement plan (make up of the Assessment report and Preliminary improvement plan), Improvement implementation plan, and Improvement report. The effort of carrying out the tasks associated with each activity and related to said products is registered in each of the work products.

PmCOMPETISOFT has been described with the standard SPEM 2.0 and edited with the EPF Composer, in order to generate documentation in a standard format which is updated and is available to organizations through the Web. We also have developed a tool called GENESIS [18], which is used to support the person *Responsible for process improvement* in the management and implementation of an SPI project and in the administration of generated knowledge.

4 Case Studies

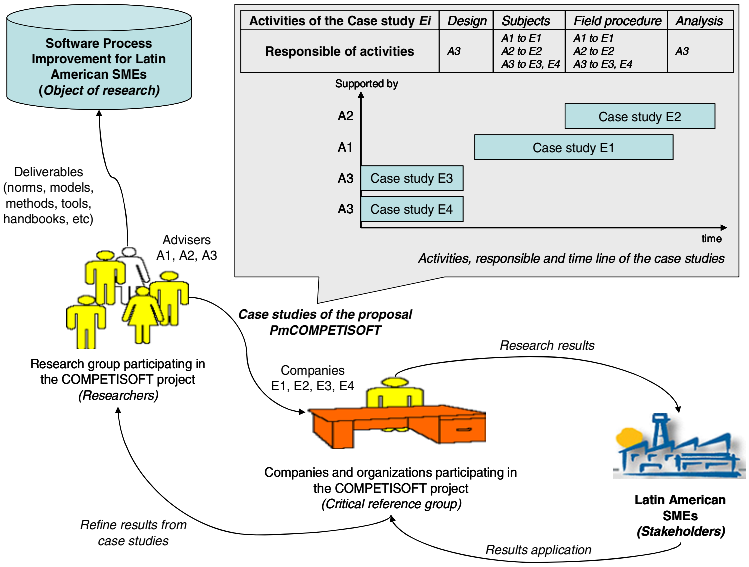
The definition, refinement and application of COMPETISOFT's components have been carried out through the use of the Action-Research investigation method (A-R), which divides the project participants into two groups: the first is made up of researchers from different universities and the second, called the critical reference group, includes computer professionals from small software organizations. In order to validate the proposed process we have conducted four case studies by following the protocol template for case studies presented in [19] (Fig. 3). The following subsection describes the case studies in terms of design, subjects, field procedures and analysis.

4.1 Design

The *main research question* addressed by this study is: Is the PmCOMPETISOFT process suitable for carrying out Software Process Improvement efforts in small software enterprises? *Additional research questions* addressed by these case studies are: (i) Is the effort of applying the proposed process suitable for the small companies? and (ii) Does the PmCOMPETISOFT process enable small companies to increase their process capabilities? Taking into account the focus presented by [20], the *design type* of the case study in this work is multiple cases – holistic, since the strategy has been applied in the context of four small companies. The *objet of study* is a new process through which to establish SPI in VSEs (PmCOMPETISOFT). The *measures* used to investigate the research questions are: (i) the effort of carrying out the tasks associated with each PmCOMPETISOFT activity, and (ii) the capability level of the processes under analysis (which need to be improved) of each company.

4.2 Subjects and Analysis Unit

The *Participating companies* in the case studies are from Argentina, Chile y Spain (called in this work E1, E2, E3 and E4), and they are part of the COMPETISOFT project critical reference group. The *analysis units* are the PmCOMPETISOFT activities and the processes to be improved within each company.



Activities of the Case study *Ei*	*Design*	*Subjects*	*Field procedure*	*Analysis*
Responsible of activities	*A3*	*A1 to E1* *A2 to E2* *A3 to E3, E4*	*A1 to E1* *A2 to E2* *A3 to E3, E4*	*A3*

Software Process Improvement for Latin American SMEs (*Object of research)*
Deliverables (norms, models, methods, tools, handbooks, etc)
Advisers A1, A2, A3
Research group participating in the COMPETISOFT project *(Researchers)*
Supported by
A2
A1
A3
A3
Case study E2
Case study E1
Case study E3
Case study E4
time
Activities, responsible and time line of the case studies
Case studies of the proposal PmCOMPETISOFT
Companies E1, E2, E3, E4
Research results
Companies and organizations participating in the COMPETISOFT project *(Critical reference group)*
Latin American SMEs *(Stakeholders)*
Refine results from case studies
Results application

Fig. 3. Application of A-R and Case studies to the COMPETISOFT project

All of these organizations started the first process improvement cycle with the support of an adviser in improvement processes who is part of the COMPETISOFT project researchers group. Table 1 describes the properties of the participant enterprises in the case studies carried out to observe and analyze the application of PmCOMPETISOFT in a real managerial context. It was suggested to the companies taking part in the case study that in the first improvement cycle they should incorporate the processes related to Profile 1 of the Process Reference Model of COMPETISOFT, which includes the processes of: *Software development* - SD, *Software maintenance* – MS, and *Specific project administration* - SPA. A further recommendation was that the *PmCOMPETISOFT process* should be used to perform the improvement activities in each organization.

Table 1. Characteristics of organizations involved in the case studies

Company	Country	Employees	Path	Main areas of professional activity
E1	Argentina	8 (7)	15 years / N&I	Development of new tailored information systems with ongoing integration of new technology
E2	Chile	18 (12)	9 years / N&I	Computer Engineering projects for the agricultural (wine and food) industry.
E3	Spain	7 (6)	4 years / N	Software development on WEB.
E4	Spain	21 (15)	12 years / N	Software development through contracts and agreements with public organizations.

Employees: Number of employees in the enterprise (People in software development and maintenance).
Path: Number of years of existence of the company / scope of the market for its products (National–N / International–I).

4.3 Field Procedure and Data Collection

The procedure governing field procedure and the data collection of the case studies is closely related to the PmCOMPETISOFT process activities, roles and work products. A description of this procedure is presented in the following subsection.

Initiating. A formal agreement in working towards process improvement was signed between each of the companies and the advisor. For the improvement cycle, E1, E3 and E4 assigned a person to the role *RPI,* and permitted them 4 hours/week. E2 also assigned a person to this role with 16 hours/week. A weekly meeting with the advisor and the *MIG* to monitor the progress of the project was also agreed on.

Each company took the processes that it was particularly interested in improving from Profile 1 of COMPETISOFT, based on its own needs and business objectives. In addition, a development project of the company was chosen, into which improvements were introduced (pilot project). The objective set out in the *Improvement proposal* of the first improvement cycle for the E1 enterprise was to improve the SPA process. For the companies E2, E3 and E4 it was to improve the SD and SPA processes. As well as these goals, the different companies too set as objective for the first cycle to increase by one level the capability of the processes chosen for improvement, taking as their starting point the value of the capability of the processes, which was established by means of an initial assessment.

Diagnosing. The process attributes of level 2 of the assessment method were used to carry out the initial assessment of the companies' processes as well as to determine the capability of the chosen processes. These attributes are PA1.1 Process performance, PA2.1 Performance management, PA2.2 Work product management and the process capability level ratings defined by assessment methodology – METvalCOMPETISOFT (which conforms to ISO/IEC 15504-2). The COMPETISOFT advisor played the role *EV*. The advisor evaluated the processes by applying the technique of evidence gathering: interviews and surveys, using the information-gathering tools developed for this purpose. The initial assessment was reported and published in each one of the firms by means of its respective *Assessment report*. Table 2 shows the initial capability of the processes in each of the enterprises.

The information concerning these processes, which was registered in the *Assessment report*, was analyzed by the *RPI* and the advisor, in order to determinate specific opportunities for improvement in each organization. For instance, apart from the improvement opportunities for the chosen processes, E1 took the decision to customize the software tool which is used to support management (Visual Studio Team System-VSTM) with regard to the COMPETISOFT reference model, in order to support its process improvement efforts. It was established that this tool would give support to the activities, documents and roles of the 9 processes of the reference model (apart from those of Profile 1, the Business Management -BM, Process Management- PM, Project Management-PjM, Human Resources Management- HRM, Infrastructure, Goods and Service Management- IM and Knowledge Management-KM). E1 put one person to work on this for 20 hours/week throughout a 2 month period. Likewise, E2 also decided to improve the formulation of proposals along with establishing the scope of software projects, which are activities that are specific to the BM process. A *Preliminary improvement plan* was generated for each VSE.

Formulating. In order to set out a general plan (establishing the improvement iterations) for carrying on with the tasks of formulation and execution of improvements, the *MIG* analyzed the *Improvement proposal* and the *General Improvement plan*. The aim of this was to refine and validate the scope of the improvement cycle, by considering the state of the processes, the company's requirements and the resources available, amongst other things. E3 and E4 therefore refined their improvement objective for the SPA process in terms of implanting some base practices only. That was due to the fact that the initial evaluation reported that no practice related to this process was being carried out. Enterprise E1 planned two iterations, while E2, E3 and E4 planned three. For each iteration the PIG (made up of advisor and RPI) used the improvement opportunities found to plan and design the corresponding improvements which were registered in the *Improvements implementation plan*. The definition of processes was based on the activities and work products of level 1 established by the COMPETISOFT reference model.

Execution. The proposed improvement activities were given to the *RPI* of each organization who, along with the person *RP*, was in charge of introducing the activities into the organization. In all the companies the employees related to the processes to be improved were given an active part to play. This was done to involve them in carrying out the improvement, thus optimizing this success factor. For instance the employees were involved in defining techniques, specific activities and templates of processes, the object being to promote the bottom-up improvement strategy. A meeting that took place at least once a week was programmed between the advisor, the RPI and the RP, in an effort to work on how to carry out the improvement activities that had been designed. The information relating to executing the improvement was registered in the *Improvement iteration report*.

Revising. We performed a post-mortem analysis of the work which took place throughout the improvement cycle, the object being to obtain a knowledge base for future improvement cycles. At the end of the improvement cycle a final assessment was carried out, and we also established how much effort was used to carry out this cycle (see Table 2). An *Improvement Report* was generated for each VSE.

Table 2. Initial and final capability of the organization´s process and cycle effort

		Capability of Processes										**Effort (hours)**		
Com.	**Assess-ment**	**SD**	**SPA**	**SM**	**BM**	**PM**	**PjM**	**HRM**	**KM**	**IM**	**Cycle length (months)**	Adviser (A)	Comp. (C)	Total
E1	*Initial*	-	2	-	-	-	-	-	-	-	6	40	264	304
	Final	1	2	*	1	1	1	1	1	1				
E2	*Initial*	0	1	0	-	-	-	-	-	-	5	89	255	344
	Final	1	2	*	*	-	-	-	-	-				
E3	*Initial*	0	0	-	-	-	-	-	-	-	3	15	39	54
	Final	1	*	-	-	-	-	-	-	-				
E4	*Initial*	0	0	-	-	-	-	-	-	-	3	41	47	88
	Final	1	*	-	-	-	-	-	-	-				

* Base practices of this process have been implanted.

4.4 Analysis and Discussion

Table 2 shows that the four VSEs have increased the capability level of their SD and SPA processes, among others. It is important to highlight that other processes such as SM and BM have also increased the capabilities of enterprises E1 and E2. This increase can be observed in the established base practices, which have been reported in the Improvement Reports. The table shows that E1 was the company which increased its level of capability in the greatest number of processes. We consider that personalization and the use of the software management tool were decisive factors in fostering and accelerating the development of the improvement cycle. Through the application of the PmCOMPETISOFT the small companies have introduced new base practices to their processes, thus allowing them to increase their capability. Based on the collected data, there is evidence that the PmCOMPETISOFT process has enabled these small companies to increase the capability of their processes.

Table 2 also shows that E2 was the firm which invested the largest amount of effort in the improvement cycle. At the same time, the mean effort per improved process is the highest of the four companies (172 hours per process). This is due to the fact that the improvement was, to a great extent, held up by the high turn-over of staff in the organization. This has made the company give priority to the management of Human Resources and to Knowledge management with regard to their next improvement cycle. Note that the effort involved on the part of companies E3 and E4 is similar (39h and 47h). However, the advisor's effort (each of the companies had the same advisor) is greater (almost triple) in the case of E4. This is because the gathering and analysis of information relating to the activities of initiating, diagnosing and formulating were performed first for E4, and then for E3. That is, this effort is related to the learning and experience acquired by the advisor in the tasks and products that had to be carried out in order to perform the process improvement activities.

From Table 2 we may can also draw the conclusion that the effort spent on improving processes per week for each organization is E1 12.7 h, E2 17.2 h, E3 4.5 h and E4 7.3 h (including the advisor's time). So the average effort spent on improvement initiatives is approximately one person taking ten hours per week. We observed that employees of each organization involved in the improvement cycle were able to take on this effort in improvement activities with no detriment to their daily activities. The percentage of effort taken with regard to each of the PmCOMPETISOFT activities for the four firms were: Initiating 4.2% (2.2% Adviser -A- and 2.0% Company -C-), Diagnosing 10.5% (6.9% A and 3.6% C), Formulating and Executing 79 % (11.2% A and 67.8% C) and Revising 6.3% (3.15% A and 3.15% C). The analysis described above offers evidence that the effort of applying the proposed process is suitable for small companies.

The main benefits which the firms have reported are the following:

- The companies have moved from a chaotic and unpredictable software process to a tangible one, which is currently being used on development projects. Both the management and the employees of the companies have seen the benefits of this result and, most importantly, they have realized the need to maintain continuous and ongoing improvement, following this same approach for future cycles.
- The firms now keep a registry of the work products related to the improved processes, together with the instancing in the projects applied (for example, E3 and E4 are using collaborative Web applications to support this information). This has allowed them to begin to generate a knowledge base which makes historic data available when decisions are being taken.
- According to E1, the implementation of COMPETISOFT has provided them with an ordered process setting. It has also meant the incorporation of good practices which are progressively implanted, together with the personalization and adaptation of a robust management tool. This has allowed them to systematically implement the activities, work products and level 1 capability responsibilities in 8 processes of the reference model.

- Company E2 has now allotted more man-hours to the RPI, designating him/her as the person in charge of institutional Quality. The quest is to continue improving processes and to implement other strategies to ensure the quality of the company's product and processes.
- The companies have a more specific vision of the organization itself which has helped and motivated it to set out on the road to quality certification. For instance, currently the E1 is conducting an ISO 9001:2000 certification, and the E3 has started to work towards a formal assessment CMMI level 2.

Based on the case studies carried out, the increase of the capability of the processes to be improved, the effort of applying the proposed process and the described benefits we consider that the PmCOMPETISOFT process is suitable to lead the projects of Software Process Improvement in small software enterprises. The results, in terms of effort, increase of capability and benefits, are an indicator that PmCOMPETISOFT can be a practical and useful strategy when facing the difficulty of carrying out SPI in VSEs. Furthermore, from the case studies we have been able to confirm that the proposed process was established and executed properly by the VSEs involved.

5 Conclusions and Future Work

The interpretation and adaptation of COMPETISOFT's methodological framework to fit in with the reality of the four VSEs has opened the way towards a rapid improvement in the chosen processes. Similarly, as in all processes of change, the process of learning and getting to know the system has been necessary, so as to make it fit in with the work style of the whole firm. The VSEs using PmCOMPETISOFT defined certain processes, thus allowing them to have an overview of the way in which they are developing their software. This has meant that VSEs are able to have clear vision of their process, seeing into which of them they should incorporate good practices and pointing the way towards a definition of specific work products and to the designation of the people to be responsible for such tasks. The overall goal for the VSEs is to have tangible processes for software development, with all the advantages that this brings with it. What is being sought is for firms to produce software that has a process-oriented focus, thereby decreasing the high dependence on people that the enterprises have hitherto had. We should emphasize that these VSEs have seen in the improvement work carried out thus far an option that would permit them to take their first steps towards ongoing process improvement. This has, moreover, allowed them to believe that the improvement of software processes, along with the benefits brought with it, can actually become a reality in their companies.

Given that the results of the case studies are encouraging, new improvement cycles are planned for the four organizations, which will take into account the aspects discovered in the first cycles. We shall conduct a follow-up in the companies in order to attempt to determine whether this strategy has made an impact on the companies' success in terms of market attributes.

Acknowledgements. This work has been funded by the projects: INGENIO (PAC08-0154-9262, JCCM of Spain), ESFINGE (TIN2006-15175-C05-05, MEC of Spain). Research fellowships funded by European Regional Development Fund (ERDF).

References

1. Laporte, C., Alexandre, S., Renault, A.: Developing International Standards for Very Small Enterprises. IEEE Computer 41(3), 98–101 (2008)
2. Richardson, I., Wangenheim, C.G.v.: Why are Small Software Organizations Different? IEEE Software 24(1), 18–22 (2007)
3. Staples, M., Niazi, M., Jeffery, R., Abrahams, A., Byatt, P., Murphy, R.: An exploratory study of why organizations do not adopt CMMI. Journal of Systems and Software 80(6), 883–895 (2007)
4. Pino, F., Garcia, F., Piattini, M.: Software Process Improvement in Small and Medium Software Enterprises: A Systematic Review. Soft. Quality Journal 16(2), 237–261 (2008)
5. Oktaba, H., Garcia, F., Piattini, M., Pino, F., Alquicira, C., Ruiz, F.: Software Process Improvement: The COMPETISOFT Project. IEEE Computer 40(10), 21–28 (2007)
6. Oktaba, H.: MoProSoft®: A Software Process Model for Small Enterprises. In: Proceedings of the First International Research Workshop for Process Improvement in Small Settings, pp. 93–101. Carnegie Mellon University, Pittsburgh (2006)
7. Weber, K., Araújo, E., Rocha, A., Machado, C., Scalet, D., Salviano, C.: Brazilian Software Process Reference Model and Assessment Method. In: Yolum, p., Güngör, T., Gürgen, F., Özturan, C. (eds.) ISCIS 2005. LNCS, vol. 3733, pp. 402–411. Springer, Heidelberg (2005)
8. McCaffery, F., Taylor, P., Coleman, G.: Adept: A Unified Assessment Method for Small Software Companies. IEEE Software 24(1), 24–31 (2007)
9. Cater-Steel, A.P., Toleman, M., Rout, T.: Process improvement for small firms: An evaluation of the RAPID assessment-based method. Inf. and Soft. Tech., pp. 1–12 (2005)
10. Scott, L., Jeffery, R., Carvalho, L., D'Ambra, J., Rutherford, P.: Practical Software Process Improvement -The IMPACT Project. In: Proceedings of the Australian Software Engineering Conference, pp. 182–189 (2001)
11. Calvo-Manzano, J.A., Cuevas, G., San Feliu, T., De Amescua, A., Pérez, M.: Experiences in the Application of Software Process Improvement in SMES. Software Quality Journal 10(3), 261–273 (2002)
12. Casey, V., Richardson, I.: A practical application of the IDEAL model. Software Process: Improvement and Practice 9(3), 123–132 (2004)
13. 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: ICSE 2000, Limerick, Ireland, pp. 626–633 (2000)
14. Horvat, R.V., Rozman, I., Györkös, J.: Managing the complexity of SPI in small companies. Software Process: Improvement and Practice 5(1), 45–54 (2000)
15. ISO, ISO/IEC 15504-4:2004 Information technology - Process assessment - Part 4: Guidance on use for process improvement and process capability determination (2004)
16. Pino, F., Garcia, F., Piattini, M.: Key processes to start software process improvement in small companies. In: SAC 2009, Honolulu, Hawaii, U.S.A, pp. 1694–1701 (2009)
17. CYTED, COMPETISOFT Methodological Framework (in Spanish) (2008)
18. Hernández, M., Florez, A., Pino, F., Garcia, F., Piattini, M., Ibargüengoitia, G., Oktaba, H.: Supporting the Improvement Process for Small Software Enterprises through a software tool. In: SES during ENC 2008, Mexicali, México (2008) (in press)
19. Brereton, P., Kitchenham, B., Budgen, D., Li, Z.: Using a protocol template for case study planning. In: EASE 2008, Bari, Italia, pp. 1–8 (2008)
20. Yin, R.K.: Case Study Research: Design and Methods. Sage, Thousand Oaks (2003)

Modeling Software Evolution with Game Theory

Vibha Sazawal and Nikita Sudan

UM Institute for Advanced Computing Studies (UMIACS)
University of Maryland
College Park, MD, 20742 USA
{vibha,nsudan}@umd.edu

Abstract. A wrong design decision at any point in the software lifecycle can lead to cost overruns and competitive disadvantage. We describe how lightweight game theory can help software engineering teams plan for future design and maintenance decisions. To demonstrate our approach, we model the real-world evolution of `java.util.Calendar` using our lightweight Basic Software Evolution Game. The game expressively models both what actually happened as well as circumstances when alternate design decisions would be optimal.

Keywords: Software evolution, game theory, software design decisions.

1 Introduction

Software has a long lifespan. A wrong design decision at any point in the software lifecycle can lead to cost overruns and competitive disadvantage. Unfortunately, complete knowledge of whether a design decision is right or wrong is often possible only in hindsight, after the customer requests changes or a competitor's future product changes the landscape of the market.

To guide software design decision-making, many have proposed using economic theories. Sullivan et al. [1], for example, proposes the use of real options theory. Boehm's seminal book *Software Engineering Economics* [2] explains classic techniques such as net present value (NPV) and value of information (VOI), among others. Denne and Cleland-Huang [3] describe how to sequence design steps to maximize return on investment (ROI). In this paper, we present a complementary approach: *game theory.* We hypothesize that game theory can model software design decision-making because (1) games naturally model the sequence of design decisions that must be made *throughout* the lifecycle, and (2) changes in requirements and customer interest are easily described because customers can be explicit players in a game. However, if game-theoretic approaches are too complex, software engineering teams will not be able to spare time to their adoption. In this paper, we contribute a lightweight application of game theory to software evolution called the Basic Software Evolution Game.

To support our hypothesis that lightweight game theory can model software evolution, we present an example of design decision-making based on the real-world scenario of Sun's `java.util.Calendar` class. We model the alternating

Q. Wang et al. (Eds.): ICSP 2009, LNCS 5543, pp. 354–365, 2009.

sequence of design decisions and customer requests as an instantiation of the Basic Software Evolution Game, an extensive-form sequential game with perfect information. We discuss this case study to demonstrate the potential of lightweight game-theoretic models to align software design and maintenance decisions with business objectives. In the case of `java.util.Calendar`, we found that the Basic Software Evolution Game expressively models what has really happened as well as circumstances when alternate design decisions would be optimal.

In Section 2, we provide a brief primer to game theory. Section 3 introduces our application of game theory to software evolution, called the Basic Software Evolution Game. Section 4 presents the `java.util.Calendar` case study as an instantiation of the Basic Software Evolution Game. In Section 5, we discuss some of the choices we made in the case study. Section 6 presents related work, and Section 7 concludes.

2 Introduction to Game Theory

Game theory describes and prescribes rational behavior for interactive decision problems. An extensive-form sequential game is a tree of decisions. The root node is player A, who acts (moves) first, and her possible decision choices (moves) are edges that emanate from the root. The children of the root represent the player who acts second (player B), and edges emanating from these nodes represent the possible decision choices that B has given the first move of A. In a two-player game, the grandchildren of the root represent the options A has for her second move. Games with more than two players are represented analogously.

The leaves of the tree represent when no more decisions are to be made. Associated with each leaf is a payoff value for each player that represents the value they obtain from the sequence of moves described by the path from root to leaf. An example is shown in Figure 1.

A sequential game is one in which players make decisions following a predetermined order. A game with perfect information occurs when the sequence of past moves is always known by any player about to move. To solve a sequential game with perfect information, we follow backwards induction [4]. The process of backwards induction is described below:

1. Start at the parents of the leaves. This is the last decision to be made in the game. For each leaf parent, select the edge that leads to the highest payoff for the last player to move.
2. Back up to the grandparents of the leaves. Select the optimal edge (move) for that player given the moves selected in Step One.
3. Continue up the tree in this manner until the root is reached. A set of selected edges that forms a path from root to leaf is considered a *solution* to the backwards induction.

The solution to a backwards induction is *sequentially rational.* Rationality in a game theory context means that each player always chooses the decision that

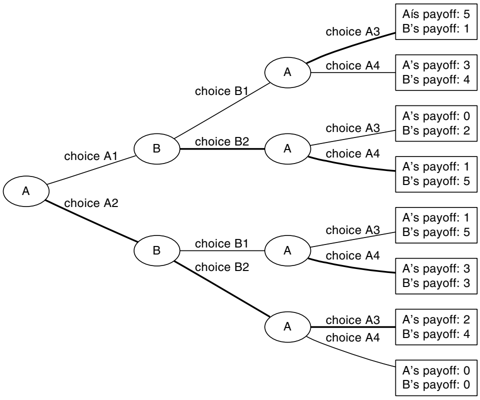


Fig. 1. An example of a two-player game. Edges in bold are selected using backwards induction. The solution to the game is {A2, B2, A3}.

is best for them in terms of personal payoff. Sequential rationality occurs when every player knows that all the other players are rational and takes that fact into account.

Game theory can also accommodate simultaneous moves, incomplete knowledge, and uncertain payoffs. In this paper, we focus on the simplest subset of game theory that can be useful to software teams; incorporating additional complexity while remaining accessible to non-economists is future work.

3 Applying Game Theory to Software Evolution

There are many ways to model decision-making performed throughout the software process as an extensive-form game. Multiple companies, customers, and all sorts of other stakeholders can all be players. Unfortunately, games with large quantities of players can become very complex. In this paper, we present a basic two-player game that we believe can describe an interesting subset of software process scenarios. Just as small formal models can help users with complex requirements issues, the risky portions of a complex software process can often be fruitfully described with one or more small games.

3.1 Basic Software Evolution Game

In our Basic Software Evolution Game, there are two players: (1) the Software Engineering Team (SE Team) that is responsible for constructing and maintaining the software, and (2) the World, who uses the software, buys the software, etc. The game is essentially played between two roles: producer and consumer.

Who makes the first move in the game? It is definitely feasible for the World to make the first move, such as choosing among a set of different initial requirements or choosing among a set of vendors. However, in our Basic Software Evolution

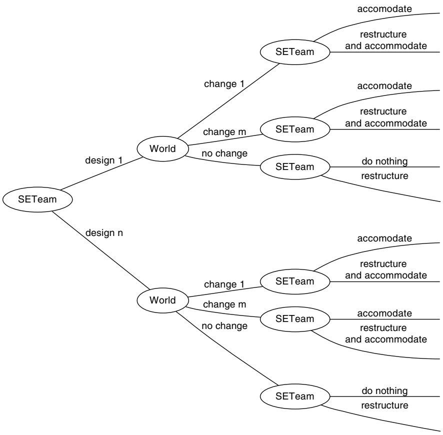


Fig. 2. Basic Software Evolution Game with 2 initial candidate designs and 2 possible change requests

Game, we assume the initial requirements are fixed. Thus, the SE Team has the first move. It is very straightforward, however, to prepend requirements-oriented moves to our basic game.

The SE Team's first move is a choice between n initial design options. What should the second move be? As with the first move, there are many candidate second moves. We can model whether the multiple customers within the World each choose to buy or not buy the product produced by the SE Team. It is also possible for the SE Team to initially choose an incomplete design and then make later moves to finalize the design as additional information arrives. We again simplify the scenario and propose that the second move is a change request by the World. This change can be a modification to the functional or non-functional requirements of the software. The World can choose among m possible requirements changes, and also it can ask for no change.

In response to the change request from the World, the SE Team's options for its next move (the third move overall) are to (1) modify the code minimally in response to the change request or (2..n) accommodate the change *and* restructure the design to any of the n-1 designs not initially chosen. We model options (2..n) as a requirements-preserving restructuring followed by a modification/extension to accommodate the requirements change request. A restructuring might be motivated by the desire to reduce the cost of accommodating future changes.

The fourth move is another change request by the World, and the fifth move is another accommodation and/or restructuring by the SE Team. The game can continue in this manner indefinitely. If an SE Team is using the game to plan for the future, then the game "ends" when the SE Team can no longer realistically predict a candidate set of possible future moves for the World or itself. Figure 2 describes the Basic Software Evolution Game where n=2 and m=2.

We can define the payoffs for the SE Team and the World for each possible path as follows. Let π be payoff, U be utility, P be payment, and C be cost. Let *base* be the implementation of the initial design and Δ be a world-requested change or a requirements-preserving restructuring. For a given Δ_j, P_{Δ_j} is likely to be zero if Δ_j is a requirements-preserving restructuring, but P_{Δ_j} may be nonzero for a World-requested change. Similarly, the utility gained by the World from a requirements-preserving restructuring is likely to be zero, but the utility gained from a World-requested change should be nonzero. We assume that prices are not affected by previous deltas. However, the cost to complete a delta does depend on previous deltas made. $tot\Delta$ is the total number of deltas, and since all moves after the first move are delta-related, $tot\Delta = (|moves| - 1)/2$.

The general formula for the payoff of the SE Team is:

$$\pi_{SETeam} = U(P_{base}) + \sum_{i=1}^{tot\Delta} U(P_{\Delta_i}) - U(C_{base}) - \sum_{i=0}^{tot\Delta} U(C_{\Delta_i}|\Delta_1 \ldots \Delta_{i-1}) \quad (1)$$

The general formula for the payoff of the World is:

$$\pi_{World} = U(base) + \sum_{i=1}^{tot\Delta} U(\Delta_i|\Delta_1 \ldots \Delta_{i-1}) - U(P_{base}) - \sum_{i=1}^{tot\Delta} U(P_{\Delta_i}) \quad (2)$$

How can the utilities in Equations 1 and 2 be estimated? Precise approaches for estimating utility are well described by Keeney [5]. Cost estimation can be performed using techniques such as COCOMO II [6]. Pricing of products can be estimated using existing market prices for similar products and/or input from marketing.

In this paper, we use a lightweight approach that defines utilities relationally. There can be a base utility of the product for the world $U_{World}(base)$ and a base utility of selling the product for the SE Team $U_{SETeam}(P_{base})$. These utilities are the same regardless of the initial design selected.[1] The utilities associated with changes can be described as fractional amounts of $U_{SETeam}(P_{base})$ and $U_{World}(base)$. With this expedient approach, SE teams can easily plug in different fractional estimates and study how the game's solution changes.

3.2 Example of the Basic Game: KWIC

As an example of the expressiveness of the Basic Software Evolution game, consider the canonical KWIC example. In Parnas' seminal paper [7], he proposes an information hiding-based modularization for a key-word-in-context (KWIC) program. Parnas' modularization can accommodate several data representation changes at very low cost. Garlan et al. [8] present an alternative modularization based on tool abstraction or "toolies." A toolie-based modularization can

[1] This is a simplification. For example, the time to build an initial implementation can affect the World's utility. Our approach emphasize the effects of future changes.

accommodate a different set of changes at low cost, such as functional extensions. Given the task to implement KWIC, should an SE team choose Parnas' modularization (PM) or toolies (T)?

Obviously the selection of PM vs. T should be based on which changes are expected to occur in the future. But what if some likely changes are easier to accommodate with PM and some are easier to accommodate with T? Would it ever make sense to start with PM and later evolve to T? Or start with T and later evolve to PM? How would intermediate changes affect a later restructuring decision? All of these potential scenarios can be modeled with the Basic Software Evolution Game.

4 Case Study: Sun's Support for World Calendars

In Java 1.1, Sun offered the class `java.util.Calendar`. `Calendar` interprets `Date` objects according to the rules of a calendaring system. In Java 1.1, the only calendar supported was the Gregorian calendar. However, many parts of the world use non-Gregorian calendars. In Java 1.4, Java added support for the Thai Buddhist calendar. This support consists of adding 543 to the year. Sun acknowledges that their Thai calendar does not support historical calendar system transitions [9]. Other sources report other easy-to-produce bugs with Sun's Thai calendar [10].

In Java 6 (beta released February 14th, 2006), Sun added support for the Japanese Imperial calendar [9]. This support consists of replacing the Gregorian year with an imperial-era based number. Sun acknowledges that the Japanese calendar does not correctly define the first day of the Meiji era.

A review of Java's Bug ID 4609228 [11] reveals many requests for alternate calendar support, including support for the Japanese, Arabic, Persian, and Hebrew calendars. This bug was submitted on December 14th, 2001. While support for the Japanese Imperial calendar was added to Java 6, all other requested calendars remain unsupported. Comments in the bug report describe many challenges in accommodating world calendars with the current codebase.

On February 2005, a commenter to the bug report announced that a Persian calendar was now supported by an open source project that is based on IBM's ICU4J [12] set of calendar-related classes. IBM's ICU4J codebase was first started by Taligent (which was acquired by IBM), and Taligent's calendar code formed part of Java 1.1. Since then, however, IBM and its partners have dramatically improved their support for world calendars. Support for the Japanese Imperial calendar became stable in ICU4J 2.8, which has a timestamp in the ICU4J repository of February 8th, 2005. As of December 5th 2008, `com.ibm.icu.util.Calendar` has support for the Chinese, Coptic, Ethiopic, Gregorian, Hebrew, Indian, Islamic, Thai Buddhist, Japanese Imperial, and Taiwanese calendars. The open-source Persian calendar project mentioned above also extends the ICU4J calendar support. Both Java's and IBM's Calendar support is open-source and available free of cost.

We present this case study to describe how design decisions (such as the design of the `java.util.Calendar` API and implementation) can have *significant business implications*. Since Java 1.1, Sun has only made minor modifications to its Gregorian calendar support, possibly because support for other Calendars would be too costly. However, in the meantime IBM identified a business opportunity and is filling the need for Java world calendar support. Comments from Sun bug report 4609228 suggest that IBM has gained considerable goodwill from this gesture. A business disadvantage for Sun is thus obtained from poor software evolution choices.

We can model this scenario with the Basic Software Evolution Game. At the start of Java 1.1, Sun faced a decision about whether to build in support for multiple calendars or just implement the Gregorian calendar. Sun chose to only support the Gregorian calendar. The World then clamored for additional calendar support. Sun then faced the choice of offering incremental support for one calendar at a time, or restructuring to support multiple calendars. So far, Sun has chosen twice to add incremental support for an additional Gregorian-like calendar.

We assume that Sun is rational; thus, Sun makes its design decisions to maximize its expected payoff. However, modeling the scenario as a game allows us to understand under what circumstances Sun's payoff would be higher with a different evolutionary path.

The players in our game are Sun and the World. Sun begins the game by choosing between two design options: support for only the Gregorian Calendar (GC), or support for multiple world calendars (MC). We define MC as a system that only supports GC at the start but can easily accommodate other calendars at low cost if needed. The World can then choose to request support for additional calendars. In response, Sun can make incremental changes to a GC solution, restructure from GC to MC, or quickly update its MC implementation at low-cost.

The extensive-form game that models the Calendar scenario appears in Figure 3. There are fifteen possible paths in the tree. Nodes have only one child when there is only one decision that makes sense. For example, we assume it would never make sense to restructure code from MC to GC.[2] With regard to edge labels, "GC to GC+1" represents adding one calendar system to the existing GC implementation. Other edge labels are analogous.

Since `java.util.Calendar` is provided free of cost, the monetary payment received for supporting a country's calendar might appear to be zero. However, Sun considers the entire Java language to be strategically valuable despite its free distribution, and the same holds for `Calendar` classes. In particular, we can measure the goodwill obtained from adding support for a country's classes.

To compute the payoffs for each of the fifteen paths, we consider the benefits and costs for both Sun and the World. Sun obtains goodwill for providing `java.util.Calendar`; in addition, it obtains additional goodwill for supporting

[2] It's possible that there is a performance difference between MC and GC but we have found no evidence of this.

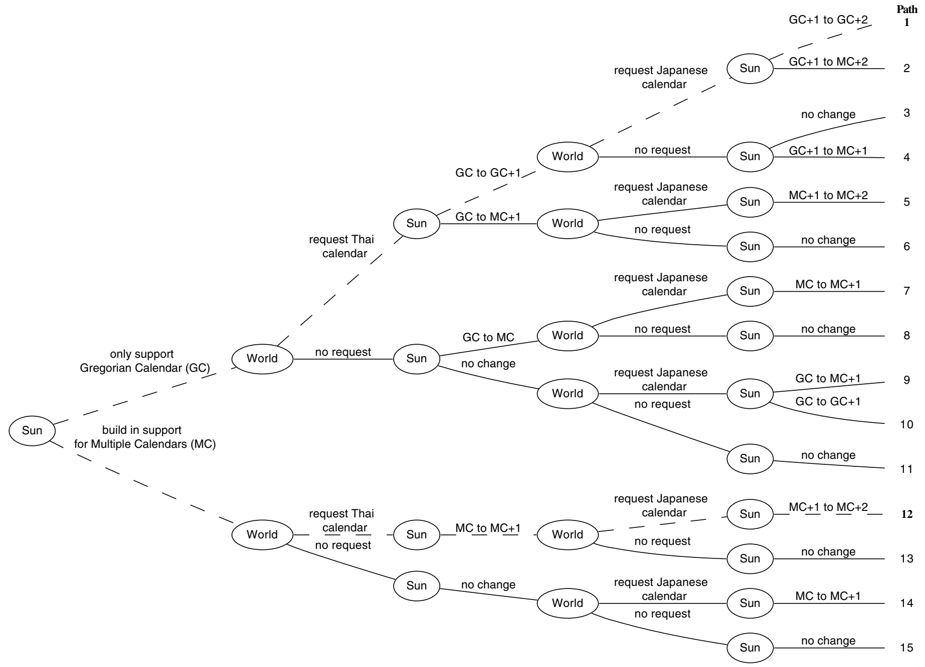


Fig. 3. Two-person game between Sun and the world regarding the design and maintenance of `java.util.Calendar`. Payoffs are listed in Table 1. The solution to the game depends on the utility values used to compute payoffs. Path 1 is the path actually taken by Sun; Table 3 shows some utility estimations where Path 12 is a better choice.

other calendars. However, if IBM comes out with support for a particular calendar before Sun, then Sun's goodwill obtained from supporting that particular calendar is reduced. We model this loss of goodwill using the fractional value α, with $\alpha < 1$. Sun incurs costs for building the Calendar code, and initial support for multiple calendars (MC) costs more than initial support for only the Gregorian Calendar (GC).

Similarly, the world obtains utility from `java.util.Calendar` and any extensions to it. However, the World's utility is reduced if they can already get alternate calendar support from IBM. We represent this loss of utility with the fractional value β, $beta < 1$.

Using these heuristics, we can define payoffs entirely in terms of utility variables. We can assume that the values of these variables have been normalized to the same utility scale and thus can be added together. Let G be the utility obtained from goodwill, C be the utility lost from spending money (cost), and U be the utility obtained by the world from having calendar support available.

The payoffs for each of the 15 paths can then be described in Table 1. To find a solution to backwards induction, we need to simplify the payoffs. Table 2 presents some reasonable assumptions. With these simplifying assumptions all payoffs can be expressed in terms of the goodwill for producing `java.util.Calendar`

Table 1. Payoffs for Sun and the World for each of the 15 paths in Figure 3

	Payoff
	Sun chooses GC, World then requests support for the Thai calendar
1	Sun: $(G_{base} + G_{thai} + \alpha G_{japanese}) - (C_{GC+1\|GC} + C_{GC+2\|GC+1} + C_{GC})$ World: $U_{base} + U_{thai} + \beta U_{japanese})$
2	Sun: $(G_{base} + G_{thai} + \alpha G_{japanese}) - (C_{GC+1\|GC} + C_{MC+2\|GC+1} + C_{GC})$ World: $U_{base} + U_{thai} + \beta U_{japanese}$
3	Sun: $(G_{base} + G_{thai}) - (C_{GC+1\|GC} + C_{GC})$, World: $U_{base} + U_{thai}$
4	Sun: $(G_{base} + G_{thai}) - (C_{MC+1\|GC+1} + C_{GC+1\|GC} + C_{GC})$ World: $U_{base} + U_{thai}$
5	Sun: $(G_{base} + G_{thai} + G_{japanese}) - (C_{MC+1\|GC} + C_{MC+2\|MC+1} + C_{GC})$ World: $U_{base} + U_{thai} + U_{japanese}$
6	Sun: $(G_{base} + G_{thai}) - (C_{MC+1\|GC} + C_{GC})$, World: $U_{base} + U_{thai}$
	Sun chooses GC, World then gives Sun time to restructure if desired
7	Sun: $(G_{base} + G_{japanese}) - (C_{MC\|GC} + C_{MC+1\|MC} + C_{GC})$ World: $U_{base} + U_{japanese}$
8	Sun: $G_{base} - (C_{MC\|GC} + C_{GC})$, World: U_{base}
9	Sun: $(G_{base} + \alpha G_{japanese}) - (C_{MC+1\|GC} + C_{GC})$, World: $U_{base} + \beta U_{japanese}$
10	Sun: $(G_{base} + \alpha G_{japanese}) - (C_{GC+1\|GC} + C_{GC})$, World: $U_{base} + \beta U_{japanese}$
	Sun chooses MC
11	Sun: $G_{base} - C_{GC}$, World: U_{base}
12	Sun: $(G_{base} + G_{thai} + G_{japanese}) - (C_{MC+1\|MC} + C_{MC+2\|MC+1} + C_{MC})$ World: $U_{base} + U_{thai} + U_{japanese}$
13	Sun: $(G_{base} + G_{thai}) - (C_{MC+1\|MC} + C_{MC})$, World: $U_{base} + U_{thai}$
14	Sun: $(G_{base} + G_{japanese}) - (C_{MC+1\|MC} + C_{MC})$, World: $U_{base} + U_{japanese}$
15	Sun: $G_{base} - C_{MC}$, World: U_{base}

(G_{base}), the goodwill for supporting the Thai Buddhist and the Japanese Imperial calendars (G_{thai} and $G_{japanese}$), and the utility gained by the world from `java.util.Calendar` (U_{base}) and support for Thai Buddhist and Japanese Imperial calendars (U_{thai} and $U_{japanese}$).

We solved the game with different values for these goodwill and utility metrics to see how varying estimates of goodwill and utility affect the solution to the game. Once the payoffs are known, solving a game is straightforward using existing software. We used the freely available applet [13] on `http://www.gametheory.net`. We show the solution to our game in Table 3. The values we used for goodwill obtained by Sun and utility obtained by the world are arbitrary and intend to represent a small sample of many possible values. Accurately assessing these utilities is beyond the scope of this paper.

As Table 3 shows, the optimal choice of initial design is heavily affected by Sun's estimation of goodwill. If Sun perceives a large gain from supporting non-Gregorian calendars, then the most cost-effective way to receive that gain is to plan for multiple calendars from the start. If Sun feels that the added value it would receive from supporting non-Gregorian calendars is small, then they would choose to incrementally tack on alternate calendars to their Gregorian base.

There are alternate ways to compute these payoffs. For example, the cost to add an incremental change could increase over time. A switch from GC to MC

Table 2. Simplifying assumptions that support back-of-the-envelope payoff calculations in the `java.util.Calendar` case study

Assumption	Rationale
$C_{MC+1\|MC} = 0$ $C_{MC+2\|MC+1} = 0$	additions of new calendars are trivial when MC is the base design
$C_{MC} = 2C_{GC}$	design support for multiple calendars takes more time and expense then designing only for the Gregorian calendar
$\alpha = \frac{1}{8}, \beta = \frac{1}{4}$	$\beta > \alpha$ because legacy code using Sun's classes won't need to be converted to a competitor's class
$C_{GC+1\|GC} = \frac{1}{8}C_{GC}$ $C_{GC+2\|GC+1} = \frac{1}{8}C_{GC}$	incremental changes are expensive to add to GC
$C_{MC\|GC} = \frac{3}{2}C_{GC}$ $C_{MC+1\|GC} = \frac{3}{2}C_{GC}$	expensive to restructure from GC to MC
$C_{MC+1\|GC+1} = C_{MC\|GC} + \frac{1}{8}C_{GC}$ $C_{MC+2\|GC+1} = C_{MC\|GC} + \frac{1}{8}C_{GC}$	cost of restructuring change is higher because of previous incremental change

Table 3. Solutions to backwards induction for different utility values

Goodwill obtained from		World's utility from		Solution to game
Thai Calendar	Japanese Calendar	Thai Calendar	Japanese Calendar	
$0.01 \times G_{base}$	$0.125 \times G_{base}$	$0.01 \times U_{base}$	$0.125 \times U_{base}$	Path 1: build GC, incrementally extend
$1 \times G_{base}$	$1 \times G_{base}$	$0.01 \times U_{base}$	$0.125 \times U_{base}$	Path 12: build MC initially
$1 \times G_{base}$	$1 \times G_{base}$	$1 \times U_{base}$	$1 \times U_{base}$	Path 12: build MC initially

could involve API changes that dramatically increase the costs for many parties. We limit our presentation of optimal paths to the three variations in Table 3 because we want to emphasize the approach and not our estimations of variables. We feel an SE Team with some knowledge of their customer can provide good-enough estimates of these variables for back-of-the-envelope computations. Put another way, software engineering organizations are regularly making judgment calls about which change requests are important; game theory makes the effects of those judgments more explicit. Using an extensive game, we can see how assumptions affect our choices and can make more informed decisions.

5 Discussion

How well does the Basic Software Evolution Game scale up to more complex scenarios? As the quantity of design choices and change requests increase, the game can grow dramatically in size. In addition, legacy projects may have multiple versions in active use at one time, and all of them may be undergoing multiple restructurings. We see the Basic Software Evolution Game as a tool for modeling the highest-risk portions of a software project at low cost. Certainly game theory supports complex games with more branches, more players, and less certainty, but we believe a lightweight approach will be most appealing to software engineering teams, at least initially.

In the `java.util.Calendar` case study, we chose to model the effect of IBM's competitive offering as a decrease in utility for Sun and the World when Sun lagged behind. We modeled competition in this way because it is a lightweight approach that is straightforward for software engineering teams to estimate. However, there are times when modeling competitors explicitly will provide more accurate decision-making support. This is especially true if a competitor's entry into the market is dependent on a sequence of decisions the software engineering team may make.

6 Related Work

Economic approaches to software process engineering are numerous. A well-known software process approach that is very similar to game theory is Theory W; indeed, Boehm and Ross state that "Theory W also has fruitful connections to game theory" [14, p.907]. Theory W emphasizes software processes in which all stakeholders come out a winner [15]. The primary difference between Theory W and classic game theory is that Theory W encourages negotiation to find a sequence of process steps where everyone wins. In classic game theory, rational actors only look out for themselves. Nonetheless, game theory can be used by negotiating parties to understand if any paths in the game lead to high payoffs for all. Such a spirit of negotiation, however, cannot occur without cooperation from all stakeholders.

Zagal et al. present maintenance-oriented design, which views the software lifecycle as an initial design followed by maintenance changes [16]. We use the same model to form our games. Zagal et al. describe a case study where they essentially follow one path of our Basic Software Evolution Game. They provide excellent arguments for an up-front investment that eases future changes.

Game theory is a well-known component of business strategy. For example, Brandenberger and Nalebuff present a number of real-world business cases that illustrate the benefits of game theory [17]. In the domain of software engineering, Oza applies game theory to client-vendor relationships in outsourcing [18]. We believe software engineering teams need specialized, understandable models to apply game theory to their technical design decision-making. In this paper, we present a specific game, the Basic Software Evolution Game, as a possible model for software engineering teams to use as they plan.

7 Conclusion

Design and maintenance decisions made throughout the software lifecycle can have significant business implications. Lightweight game theory can help software engineering teams plan throughout the software process. To demonstrate our approach, we present the evolution of `java.util.Calendar` as a an instantiation of our Basic Software Evolution Game. The game expressively models both what actually happened as well as circumstances when alternate design decisions would be optimal. We intend to encourage software engineering teams to consider lightweight game theory as a means to explicitly understand the tradeoffs between their various design and maintenance options.

References

1. Sullivan, K., Griswold, W., Cai, Y., Hallen, B.: The structure and value of modularity in software design. In: ESEC/FSE (2001)
2. Boehm, B.W.: Software Engineering Economics. Prentice Hall PTR, Englewood Cliffs (1981)
3. Denne, M., Cleland-Huang, J.: Software By Numbers. Prentice Hall PTR, Englewood Cliffs (2003)
4. Dutta, P.K.: Strategies and Games: theory and practice. MIT Press, Cambridge (1999)
5. Keeney, R.L.: Value-Focused Thinking. Harvard University Press (1996)
6. Boehm, B., Abts, C., Brown, A.W., Chulani, S., Clark, B.K., Horowitz, E., Madachy, R., Reifer, D.J., Steece, B.: Software Cost Estimation with Cocomo II. Prentice Hall PTR, Englewood Cliffs (2000)
7. Parnas, D.: On the criteria to be used in decomposing systems into modules. Communications of the ACM (1972)
8. Garlan, D., Kaiser, G., Notkin, D.: Using tool abstraction to compose systems. IEEE Computer (1992)
9. Sun Microsystems: Supported calendars (2005), http://java.sun.com/javase/6/docs/technotes/guides/intl/calendar.doc.html
10. Phillips, A.: It's about time (2005), http://www.inter-locale.com/demos/about-time/about-time-ext.xml
11. Sun Developer Network: Bug ID: 4609228 (cal) RFE: Provide additional local calendars in Java (2001-2005), http://bugs.sun.com
12. ICU Project: International components for unicode (2000-2008), http://icu-project.org
13. Shor, M.: Extensive form game applet (2001-2006), http://www.gametheory.net/Mike/applets/ExtensiveForm/
14. Boehm, B.W.: Theory-W software project management: Principles and examples. IEEE Transitions on Software Engineering (1989)
15. Boehm, B.W., Bose, P.: A collaborative spiral software process model based on theory W. In: ICSP (1994)
16. Zagal, J.P., Ahués, R.S., Voehl, M.N.: Maintenance-oriented design and development: A case study. IEEE Software (2002)
17. Brandenburger, A.M., Nalebuff, B.J.: The right game: Use game theory to shape strategy. Harvard Business Review (1995)
18. Oza, N.V.: Game theory perspectives on client - vendor relationships in offshore software outsourcing. In: EDSER (2006)

Structural Considerations in Defining Executable Process Models

Borislava I. Simidchieva, Leon J. Osterweil, and Alexander Wise

Laboratory for Advanced Software Engineering Research (LASER)
Department of Computer Science
University of Massachusetts Amherst
Amherst MA, 01003 USA
{bis,ljo,wise}@cs.umass.edu

Abstract. This paper examines the question of how to structure the representation of a process in order to assure that the representation is effective in supporting such diverse activities as process understanding, communication among process participants, and process execution. The paper uses the example of a negotiation process to demonstrate that one process structure (which we refer to as the narrative form) seems to be quite effective in supporting understanding and communication, but then indicates that this structure seems problematic in supporting process execution. The paper indicates that a different structure (which we refer to as the role-oriented form) seems much more appropriate and effective in supporting execution, but may be lacking at supporting communication. In addition to serving different purposes, the two structures seem to represent different underlying models–a static process model, and a similar, but more complex, execution model. The properties of these two complementary structures are then analyzed and evaluated. The paper then uses these observations to raise questions about the underlying needs for effective process representation, suggesting in particular that a single process representation may not be a suitable basis for supporting the range of needs that process representations are expected to address.

1 Introduction

In other papers we have noted that there are many uses to which people wish to put representations of processes [1,2]. Among these uses are the facilitation of communication and coordination, the identification of defects and deficiencies in processes, and the automation of processes. We have previously noted that these differences in motivation have given rise to different notations with which to model and represent processes. Thus, for example, box and arrow diagrams have been popular and successful as devices for facilitating coordination of the efforts of process participants by communicating to them a sense of the juxtaposition of the various roles of process performers. On the other hand, there is growing evidence that the more formal process modeling notations could be effective as bases for supporting process defect detection and process automation [3,4].

Our recent experience indicates that different motivations and prospective uses for process representations seem to suggest not only the value of different process

Q. Wang et al. (Eds.): ICSP 2009, LNCS 5543, pp. 366–376, 2009.

representation notations, but also the value of using different process representation architectures, even when one process representation notation may seem suitable for meeting multiple uses. In particular, this paper describes our experiences in discovering that a process representation architecture that seemed quite suitable for supporting process communication posed problems for supporting desirable infusions of automated support into the performance of the process.

To be specific, the paper will describe the development of a “narrative model” of a specific dispute resolution process, and summarize our experience in using it to help process participants understand their roles and consider improvements. The narrative model emphasizes the intermixing of the activities of the various process participants, thereby elucidating various coordination issues. The paper then goes on to describe the difficulties that we encountered when we attempted to use this representation as a guide in inserting the use of automation in order to address some of the coordination issues. The difficulties we encountered caused us to completely refactor our process model in order to structure it around specifications of the different roles played by different types of process performers. The “role-based model” of the dispute resolution process has been effective as the basis for supporting the desired automated support. On the other hand, the role-based model seems significantly less useful than the narrative model in supporting communication among the process participants.

The paper begins by presenting a summary of the process to be modeled and the initial narrative model used to describe it. The paper then indicates the problems we encountered when we attempted to use this model as the basis for automation. The paper then presents the role-based model resulting from the refactoring of the narrative model. Finally, the paper ponders various issues raised by this experience, some of which indicate the possibility of an inherent need for different process languages and architectures in order to effectively meet various user needs.

2 The Process—Online Dispute Resolution

In earlier papers we have described our efforts to use process definition and analysis technology to facilitate the resolution of disputes [5,6]. In those efforts, we collaborated with the US National Mediation Board (NMB) to develop a process definition of the approach that NMB uses to mediate disputes in the US transportation industries. Our goals in doing this were various, and included 1. helping the NMB to understand their process so that they might improve it, 2. enabling the NMB to involve disputants in arriving at mutually agreeable approaches to the mediation of their disputes, 3. helping the NMB to train new mediators, and 4. supporting NMB’s process with automation in order both to create efficiencies and to create novel mediation approaches made possible through automated support. In pursuing this last goal we were aiming at the development of systems that are commonly described as Online Dispute Resolution (ODR) systems. This term is used to describe systems that exploit computer and communication technologies to facilitate the resolution of disputes.

The vehicle we used for defining NMB’s dispute resolution process was the Little-JIL process definition language [7]. The features of this language are addressed in numerous other papers, and for that reason (and because of space limitations) we omit

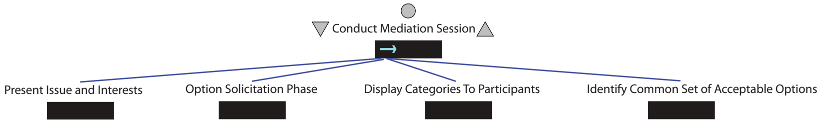

Conduct Mediation Session
Present Issue and Interests
Option Solicitation Phase
Display Categories To Participants
Identify Common Set of Acceptable Options

Fig. 1. Narrative flow of process

descriptions of most of those features here. A central feature of Little-JIL is that it supports the definition of processes as steps (represented as black bars in Figures 1-4) that are organized hierarchically (with child steps connected to the left half of their parent's step by edges), and where each step is annotated with a specification of the type of agent that is responsible for the step's performance. In the case of the NMB's dispute resolution process, some steps are annotated as being the responsibility of the mediator, while other steps are annotated as being the responsibility of a disputant. The NMB's process assumes that there are two sides in every dispute, and that each side may be represented by more than one disputant. In some cases a step may be annotated to mandate that it is to be performed by a disputant from a side that is either the same, or opposite, of the side of a disputant that performed some other step.

As is often the case in creating a model of a process, elicitation of the process became an issue of central importance. Experience in several domains [3,8] has suggested to us that one coherent overarching view of the process is often unavailable, and must be synthesized by putting together different views of the process, each being elicited from a different participant in the process. In the case of the NMB dispute resolution process, however, our elicitation efforts were considerably aided by a domain expert who was a very senior, very experienced mediator. This domain expert was the source of both the high level view of this process, and many of the details needed to support formulation of a coherent view of the process. Our domain expert also indicated places and ways in which different mediators and mediation situations dictated the desirability of creating variants from the baseline mediation process.

From a very high level point of view the process definition that emerged can be viewed as the orchestration of a carefully structured multi-person conversation. The structure specifies first the elucidation of the issues underlying a dispute by the mediator, then the injection of ideas and views by the disputants, then the summarization of these by the mediator, and then the iteration of suggestions for a resolution by the mediator, with responses by the disputants. The process is defined to iterate either until agreement on a resolution has been reached, or until it is agreed that agreement cannot be found. This description of the process strongly suggests that showing the interactions of the mediator and the participants (both individual disputants and their parties) would seem to capture of essence of this process.

Figure 1, for example, shows a small portion of the Little-JIL definition of this process in which the mediator and disputant activities are interleaved (note that the right arrow in the root step's step bar indicates sequential execution of its child steps), with the mediator presenting the issue and interests, the disputants contributing possible options that might address the issue, the mediator then categorizing and presenting the contributed options, and the disputants identifying a common set of acceptable options. Figure 2 shows another portion of the process in which the mediator, after having the

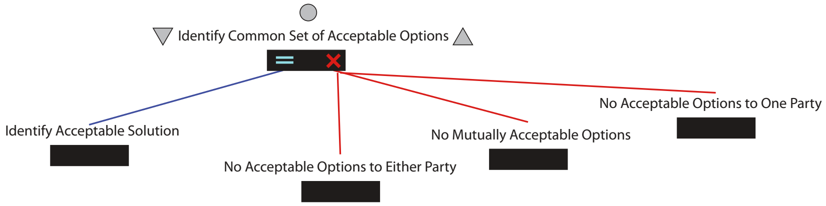

Identify Common Set of Acceptable Options
Identify Acceptable Solution
No Acceptable Options to Either Party
No Mutually Acceptable Options
No Acceptable Options to One Party

Fig. 2. Interruption of normative flow

disputants attempt to identify a common set of acceptable options, decides that it is necessary to interrupt the interactions among the disputants in order to deal with the fact that the process has not identified any mutually acceptable options. Our experience suggests that the process model depicting these interactions helped the mediator to gain a better understanding of the nature of the process, and was indeed useful to him and to the NMB in the training of new mediators. Further, preliminary experience has suggested that disputants should be better able to accept the process because this view helped them to understand why they were being asked to structure their participation as mandated (e.g. why from time to time it is desirable for the mediator to interrupt ongoing discussions). Thus this experience suggested that this model of the process was of considerable value in addressing goals 1, 2 and 3, outlined above.

When we moved on to address goal 4, the provision of automated support for the process, difficulties became apparent.

3 Supporting Execution of the ODR Process

Little-JIL's semantics are rigorously defined by means of finite state machines. The semantics define the behavior of a step to be quite similar to the behavior of a procedure invocation. Thus, each step definition includes a specification of input and output parameters that function in a way that is similar to that of the formal parameters of a procedure. The edge that connects a Little-JIL step to its parent can carry arguments that are bound to the child step's formal parameters at run time. As noted above, each step is annotated with a specification of the type of agent that is to be responsible for the performance of the step, and at runtime a resource manager searches a repository of available resources to identify and then bind a resource instance that matches the step's agent type. After this has been done, the step is ready to be executed.

All that being the case, the execution of a Little-JIL step is left to the resource instance that has been bound as the step's agent. Little-JIL's runtime system allocates to each agent an agenda, which is a list of the steps to which the agent has been assigned. The step's input arguments are passed to the agent by placing them in the agenda item that corresponds to the step. Similarly, once the step has been executed, the resulting output arguments are placed by the agent into the agenda item that corresponds to the completed step, and execution then proceeds.

From this point of view it can be seen that execution of a process defined in Little-JIL is centered largely upon the manipulation of the agendas of the various agents that are participating in the process. It is particularly important to note that there may be many

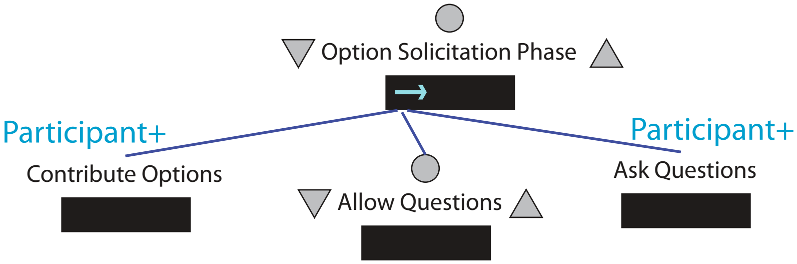


Fig. 3. Option solicitation phase

resource instances of the same type participating in the execution of a process. Thus, for example, there will be many resource instances of type participant (disputant) in an ODR process. Each of these resource instances must have its own agenda, containing agenda items that are the specific steps to be executed by that resource instance. Thus, for example, when a specific resource instance, say Participant1 is assigned the task of submitting a comment, only Participant1 can carry out that task. Moreover, a reply intended for Participant1 must be delivered only to Participant1, not to any participant who might be available.

The need to treat each resource instance individually raised problems in our efforts to use the narrative version of our ODR process as the basis for execution of the process, as the narrative version of the process is essentially a static structure of step types (e.g. steps that are to be bound to any resource instances of a specified type), but the execution of the process results in a more complicated, dynamic structure that requires the management of individual step instances (i.e., the specific steps that each of the resource instances is charged with carrying out).

To be specific, note that Figure 3 defines the way in which the option solicitation phase, one of the bottom level steps referenced in Figure 1, is to be carried out. First, the disputants are asked to contribute options (the Participant+ notation on the step's incoming edge indicates that the subprocess Contribute Options, whose details are not shown here, is to instantiated once for each agent of type Participant), then, after a certain number of options have been suggested, the mediator can choose to allow participants to ask questions by executing the Allow Questions step, after which participants can submit clarifying questions about identified options in the Ask Questions subprocess.

Difficulties with the narrative architecture become clearer when considering the case when only some participants who have submitted options are allowed to ask questions. Since the two subprocesses that are executed by disputants, Contribute Options and Ask Questions, are instantiated once per each participant separately, there is no way of knowing whether a participant for whom Ask Questions is instantiated has actually submitted any options. In order to solve this problem and also to be able to account for disputants on a per-instance basis, the process could be restructured so that the entire Option Solicitation Phase subprocess is instantiated once for each participant. Although this addresses the original concern, it would also result in the mediator having to execute the Allow Questions step multiple times, to allow each single disputant to proceed. This is highly undesirable because it introduces unnecessary work for the mediator and is, moreover, error-prone since it might result in a disputant being overlooked inadvertently.

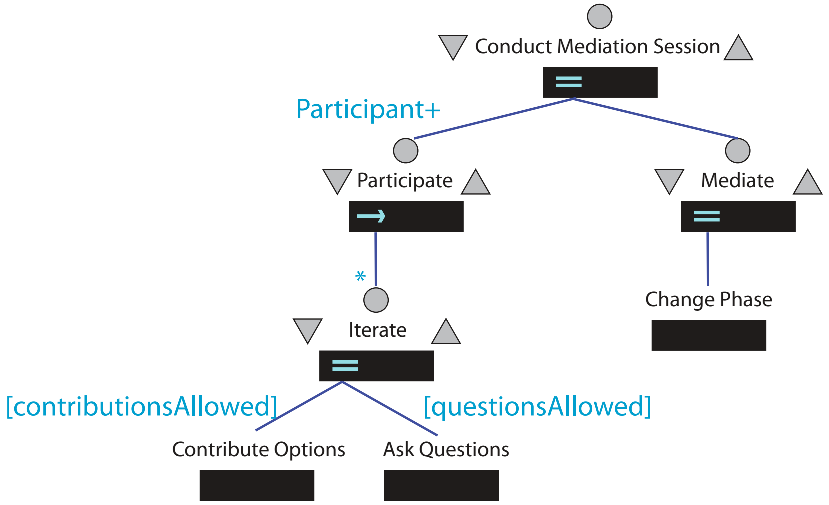


Fig. 4. Role-based architecture

This difficulty is only one example of the way in which the narrative form of the process turned out to be quite unsuitable for specifying important forms of agent interactions clearly, precisely, and tersely. As indicated above, an obvious way in which to address this sort of difficulty is to define such a process in such a way that there is a separate process subtree defining this interaction for each resource instance of type Participant at every point in the process where an interaction between mediator and participants was defined. Ultimately, as noted above, the process definition structured in this way must become large, complex, and increasingly difficult to understand.

Our solution to this problem was to refactor the process definition as a role-based process, in which each of the activities of each of the resource instances was modeled as a separate Little-JIL process tree, with all such trees defined to be executing in parallel with each other. Figure 4 shows the portion of the process needed to represent the process structure from Figure 3 (note that the parallel execution of these trees is denoted by the = sign in the step bar for the step that is the parent of all of these instances). In this process, the left branch (Participate) is performed by each participant, and the right branch (Mediate)–by the mediator. Participation in this process is an iterative activity in which for each iteration, the predicates associated with Contribute Options and Ask Questions consult the process state, which is specified by the mediator's use of Change Phase (and in the case of Ask Questions by the participant's previous actions) to determine which actions can be performed currently.

This role-based process architecture made it straightforward to define the actions of each resource instance, and to separate the actions of the different agents of the same type because this architecture mimics the dynamic execution model closely, unlike the narrative process architecture, which is based on the static model. On the other hand, the role-based process made it correspondingly difficult to indicate the necessary coordination of the activities of the different agents. In the role-based process, a message-passing channel construct in Little-JIL is used to define the transfer of messages and information between the mediator (for example) and each of the separate participants (for example). This use of channels is effective in defining appropriate coordination, but at the expense of the clarity that is a feature of the narrative form of the process definition. Thus we see that the need for executability (goal 4) has given rise to the need for a process definition architecture that does indeed support executability, but at the expense of goals 1, 2, and 3.

4 Discussion

Our experience in developing the executable form of our ODR process has called into question our previous belief that there was a single representation of a process that was equally effective in supporting all of the many desired uses for process representations. Previously, we had believed that it was essential to identify a process definition language that was sufficiently clear, precise, and broad in semantic scope. Although we still believe that this is essential, we now also believe that even such a process definition language may need to be used in different ways in order to represent a process in ways that are effective in supporting different process uses.

This experience highlights particularly clearly that the need to actually execute a process definition raises a set of issues and requirements that are different from the issues and requirements that seem to be foremost in supporting process understanding and communication. One key difference seems to be that process execution requires the creation and maintenance of the dynamic state of the executing process. One particularly important aspect of the dynamic state of a process is its specification of the precise activities that each of the performers of the process is engaged in at any time, and which precise artifacts and other resources are being used in order to carry out these activities. This dynamic model can become very different from the static process definition. Especially in cases where resource instances of the same type are performing activities at the same time as each other, it becomes clear that a structure such as the narrative process, which is a structure of types of resources, lacks an important dimension, namely facilities for specifying the different items of information relevant to each of the different instances. This need to address the different characteristics, activities, and artifact utilization for each different resource instance is met far more successfully and effectively by the role-based process structure exemplified by Figure 4. As noted above, however, this process structure seems notably less clear for the purposes of understanding than the narrative form.

Interestingly, the desire to gain process understanding and communication through process elicitation leads to a similar conclusion. The case described above was unusual in our experience in that a single domain expert had a clear understanding of the role of all involved in the process being described. It is far more usual to find that in processes involving multiple types of agents, each contributer has a clear view of his or her own participation, but a less clear view of what others contribute. Thus each contributor group (corresponding to a type of agent) can be helpful in contributing information needed to define a different parallel branch of the process structure. The role-based process structure is often the logical starting point in dealing with such processes, and it then becomes important to refactor this role-based structure to synthesize a narrative structure from it for communication purposes.

In retrospect, our view that processes are a type of software should have suggested far sooner the need for these types of structures. If a process definition is analogous to the text of an executable program, then the structure of an executing process should be expected to be analogous to the internal structure of an executing program, which is clearly very different from its source text. Indeed a process that integrates and coordinates the actions of different types of participants–and multiple instances of different participant types–seems quite analogous to a software system composed out of components each of which supports multiple execution threads. A visualization of such complex parallel, multi-threaded systems should not be expected to bear much resemblance to the source text of such systems, thus presaging the situation described here.

Indeed, as in the case of complex software systems, different needs dictate the need for two representations. A narrative structure–a static representation that is a structure of types of activities, artifacts, and resources–seems necessary to aid understanding. A role-based structure–a dynamic representation, which is necessarily a structure of instances–however, serves a different set of needs. Although our initial expectation was that the instance structure needed to support representation of the process dynamic state might be patterned closely after the static structure, the experiences in this paper now strongly suggest that this may not be a realistic expectation. The analogy of processes to programs further suggests that this expectation is probably unrealistic.

4.1 Future Directions

The preceding discussion suggests a number of directions for future research. Most immediately, we note that the role-based structure of a process is not itself an actual representation of the dynamic state of an executing process, but rather a suggestion of at least part of its underlying structure. We propose to take the suggestion and use it as the basis for creating just such a clear representation of the dynamic state of a process. We will do this with a recognition of the probable analogy to the problem of representing the state of an executing concurrent, multi-threaded program. Accordingly, we expect to draw upon work from that domain, while also expecting that our work in the process domain might have applicability to the domain of concurrent program visualization.

In previous work [9], we discussed the benefits of considering closely-related processes as variants of one another, and proposed an approach for reasoning about a collection of such variants by defining process families. We described a process family as a group of processes that are the same, or sufficiently similar, at a high level of abstraction, but may exhibit differences, or variations, at lower levels of abstraction. This definition rested on the assumption that all variants within a family share a common process core, and the variations are different elaborations of this common core.

According to this definition, the narrative and the role-based process architectures described in this paper are not members of the same process family because they are not elaborations of a common process core. It is apparent, however, that these two architectures have a lot in common. Contrary to our previous experience with process variants, these two process architectures share low-level functionality (e.g. a disputant contributing an option, or the mediator presenting an issue statement), but have completely different orchestration of events at the high level. They might even be considered to be different projections of the same underlying model since, ultimately, they both define the same process functionality. If these two process architectures can be construed as different views of the same model, they must be members of the same process family.

This clearly indicates that further investigation is needed to determine if the narrative and role-based versions of the negotiation process are variants of one another, and whether our initial definition of a process family needs to be reassessed to accommodate architecture differences.

Finally, in undermining the expectation that there should be only one form of a process representation that supports all possible uses, this work also raises the question of how many different process representation structures may be needed in order to support all of the many varied uses of processes. Indeed, such an understanding of the basic structures needed to support different uses may lead to clearer understandings of the notations best suited to supporting these different representations. This may in turn, then, help to shed new light on the ongoing question of which process representation notations are best suited for which needs.

5 Related Work

There are many approaches to representing processes, and moreover, many of the approaches to representing software systems are also quite applicable to representing processes as well [10,11]. Most approaches, such as UML module interaction diagrams [12] and IDEF diagrams [13] are similar in goals and design to the narrative form described in this paper. Others, such as UML message sequence diagrams (or "ladder charts") [12] are more similar in goals and design to the role-based form described here. It is also interesting to note that scientific workflow systems such as Kepler [14] are more in the style of the narrative form of description, but it has been noted that this form is increasingly inadequate as the scientific processes that it describes are to be used to support process execution [8]. Finally, we note that work on Viewpoints [15] also recognizes the value of representing a system as a collection of the different views of the system by the different participants and observers of the system.

Thus the observations described in this paper are not inconsistent with work that has been done previously in the area of software systems. Our paper, however, indicates that the need for these two different structural approaches is also present in representing processes. Moreover, our work suggests that the desired application may have a particularly important role to play in deciding which structural approach to use. The role-based structure seems particularly useful and necessary in supporting execution, while clear communication of coordination issues seems to indicate the use of the narrative form. Our ongoing work is attempting to determine whether a process family approach may indicate how the two structures may be considered views of a more fundamental form. If so, then this work should also be interesting and applicable in the domain of software system modeling and representation.

Acknowledgments

The authors wish to express gratitude to Stanley M. Sutton, Jr, Barbara Staudt Lerner, Lori A. Clarke, and members of the Laboratory for Advanced Software Engineering Research at the University of Massachusetts Amherst who have participated in this research, and clarified the points made in this paper.

This material is based upon work supported by the US National Science Foundation under Award Nos. CCR-0427071, CCR-0204321 and CCR-0205575. The views and conclusions contained herein are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of The National Science Foundation, or the U.S. Government.

References

1. Zhu, L., Osterweil, L.J., Staples, M., Kannengiesser, U., Simidchieva, B.I.: Desiderata for languages to be used in the definition of reference business processes. International Journal of Software and Informatics 1(1), 37–65 (2007)
2. Osterweil, L.J.: Unifying microprocess and macroprocess research. In: Li, M., Boehm, B., Osterweil, L.J. (eds.) SPW 2005. LNCS, vol. 3840, pp. 68–74. Springer, Heidelberg (2005)
3. Clarke, L.A., Avrunin, G.S., Osterweil, L.J.: Using software engineering technology to improve the quality of medical processes. In: ACM SIGSOFT/IEEE 30th International Conference on Software Engineering (ICSE 2008), pp. 889–898 (May 2008); Invited keynote address by Lori A. Clarke
4. Chen, B., Clarke, L.A., Avrunin, G.S., Osterweil, L.J., Henneman, E.A., Henneman, P.L.: Analyzing medical processes. In: ACM SIGSOFT/IEEE 30th International Conference on Software Engineering (ICSE 2008), pp. 623–632 (May 2008)
5. Osteweil, L.J., Katsh, E., Sondheimer, N.K., Rainey, D.: Early lessons from the application of process technology to online grievance mediation. In: 2006 National Conference on DIgital Government Research (2005)
6. Osterweil, L.J., Clarke, L.A., Gaitenby, A., Gyllstom, D., Katsh, E., Marzilli, M., Sondheimer, N.K., WIng, L., Wise, A., Rainey, D.: A process-driven tool to support online dispute resolution. In: International Conference on Digital Government Research, pp. 356–357. ACM Press, New York (2006)
7. Wise, A.: Little-JIL 1.5 Language Report. Technical report, Department of Computer Science, University of Massachusetts, Amherst, MA (2006)
8. Osterweil, L.J., Clarke, L.A., Podorozhny, R., Wise, A., Boose, E., Ellison, A.M., Hadley, J.: Experience in using a process language to define scientific workflow and generate dataset provenance. In: ACM SIGSOFT 16th International Symposium on Foundations of Software Engineering (FSE16), pp. 319–329 (2008)
9. Simidchieva, B.I., Clarke, L.A., Osterweil, L.J.: Representing process variation with a process family. In: Wang, Q., Pfahl, D., Raffo, D.M. (eds.) ICSP 2007. LNCS, vol. 4470, pp. 109–120. Springer, Heidelberg (2007)
10. Osterweil, L.J.: Software processes are software too. In: 9th International Conference on Software Engineering (ICSE 1987), pp. 2–13 (March 1987)
11. Osterweil, L.J.: Software processes are software too, revisited. In: 19th International Conference on Software Engineering (ICSE 1997), pp. 540–548 (September 1997)

12. Object Management Group: OMG Unified Modeling Language (OMG UML) Superstructure. Technical Report formal/2007-11-02, Object Management Group, Version 2.1.2 (November 2007)
13. US Air Force: ICAM architecture. part II, functional modeling manual (IDEF0). Technical Report AFWAL-TR-81-4023, Materials Laboratory, Wright-Patterson Air Force Base (1981)
14. Ludäscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger-Frank, E., Jones, M., Lee, E., Tao, J., Zhao, Y.: Scientific workflow management and the kepler system. Concurrency and Computation: Practice & Experience 18(10), 1039–1065 (2006)
15. Nuseibeh, B., Kramer, J., Finkelstein, A.: Expressing the relationships between multiple views in requirements specification. In: Proceedings of the 15th International Conference on Software Engineering, pp. 187–196 (May 1993)

Analyzing a Software Process Model Repository for Understanding Model Evolution

Martín Soto, Alexis Ocampo, and Jürgen Münch

Fraunhofer Institute for Experimental Software Engineering
Fraunhofer-Platz 1
67663 Kaiserslautern, Germany
{soto,ocampo,muench}@iese.fraunhofer.de

Abstract. Process models play a central role in the process improvement cycle. Often, large process models evolve in an ad-hoc manner, a fact that may easily have critical implications such as increased maintenance effort. This highlights the need for supporting the control and management of process model evolution, a kind of support that is currently widely missing. Analyzing existing model repositories in order to better understand model evolution can be seen as a first step towards identifying requirements for process model evolution support. This article presents a study that analyzes the evolution history of a large process model with the purpose of understanding model changes and their consequences. Besides the study description, the article provides an overview of related work, and suggests open questions for future work.

Keywords: process modeling, process model change, process model evolution, model comparison, V-Modell XT, Evolyzer.

1 Introduction

Process models play a central role in the process improvement cycle. On the one hand, process analysis activities intended to identify improvement opportunities use models as one of their main inputs. On the other hand, the process model (usually in the form of a process guide) constitutes the main support process actors have in order to enact the process accurately. For this reason, any proposed process improvements will only be enacted if they are added to the model first. The consequence is that process models must be maintained in lockstep with the process itself, in order for controlled process improvement to happen in a sustained fashion.

Given its importance for process improvement, as well as the large size and complexity of many industrial process models, it comes as a surprise that support for managing model evolution is still widely missing. Standard version management tools are generally barely adequate for the task of storing a model's version history, or observing and analyzing the changes that have happened to it. This is in stark contrast to the situation in code evolution, where version management has been practiced for decades, and countless research efforts have been devoted to analyzing the resulting version histories.

It it this lack of proper evolution support for process models that motivated us to perform the empirical study presented in this article. In the study, which is a

Q. Wang et al. (Eds.): ICSP 2009, LNCS 5543, pp. 377–388, 2009.

follow-up to previously published work by the authors [1], we analyzed a set of 604 development versions of a large process model with the purpose of identifying change patterns and understanding their effect over time. The underlying assumption is that, given the size and complexity of the studied model, its development process will have a behavior similar to that of a standard software development process over time. In order to perform the study, we used our *Evolyzer* model comparison system to compare versions along the history pairwise, and produced a database of detailed changes that was, in turn, the subject of graphical and statistical analysis.

We see the main contributions of this work at two different levels. At the level of the concrete study, our results provide some further evidence of the similarity between the evolution of large models and that of industrial software systems. These similarities suggest, in turn, that in order for model-based development to succeed, support for model evolution must be improved until it is at least as good as support for code evolution is now. Furthermore, at a more general level, our study can be seen as a demonstration of the practical feasibility of observing and analyzing the evolution of complex process models, as well as of the suitability of our model comparison tools for this purpose. In this sense, we expect our work to provide a basis for more advanced empirical work on process model evolution in the future.

The rest of the paper is organized as follows: Section 2 describes the execution of the empirical study and analyzes its results, Section 3 briefly surveys related research work, and Section 4 presents the main conclusions of the work and discusses a number of research questions resulting from the present work that we would like to address in the future.

2 An Empirical Study on Model Follow-Up Change

As stated above, the assumption underlying our empirical work on process model evolution is that there exists a strong parallel between model evolution and general software evolution. For this study, we concentrated on one particular aspect, namely, the effect of changes on stability. Our central research question was whether changes made to a model are likely to introduce problems that must be corrected in follow-up changes. A positive answer to this general question would imply that, similar to the case of code development, projects dealing with the development and maintenance of complex models have to plan for a stabilization period when doing extensive changes.

2.1 The German V-Modell XT

We investigated this main research question in the context of a large software process model, the German V-Modell® XT. The V-Modell XT [2] (not to be confused with Royce's V-Model [3]) is a high-level process description that is currently being adopted as the software development standard for the German public administration. It covers such aspects of software development as project management, configuration management, software system development, and change management, among others. In printed form, the latest English version at the time of this writing (version 1.2.1) is 765 pages long and describes about 1500 different process entities. Internally, the V-Modell XT is structured as a hierarchy of process entities interconnected by a complex graph of relationships. This structure is completely formalized, and suitable for automated processing. The actual text of the model is attached to the formalized structure, mainly in the form of entity and relationship descriptions, although a number of documentation items (including a tutorial introduction to the model) are also integrated into the structure in the form of so-called text module entities.

Actual editing of the model is performed with a software tool set created specifically for this purpose. The printed form of the V-Modell XT (a process guide) is generated automatically by traversing the structure in a predefined order and extracting the text from the entities found along the way. The V-Modell XT contents are maintained by a multidisciplinary team of experts, who work, often concurrently, on various parts of the model. In order to provide some measure of support to this collaborative work, the model is stored as a single XML file in a standard code versioning system (CVS). As changes are made by the team members, new versions are created in this system. Being a standard versioning system intended for code, CVS is able to store a version history and maintain a simple change log, but can hardly provide useful information about the actual changes done in each version. In particular, the output of the *diff* program used by CVS to compare versions cannot easily tell which entities were affected by a version or in which way they were changed.

The change logs show that, since its initial inception, the model has been changed often and for a wide variety of reasons. Changes may be as simple as individual spelling or grammar corrections, or as complex as the introduction of a whole set of processes for hardware development and software/hardware integration. The richness and complexity of this change history makes the V-Modell XT a very interesting target for evolution analysis.

The descriptive analysis we performed in our exploratory study [1] showed that much of the changing activity concentrated around public releases of the model and affected some process modules much more than others. While looking at the changes that happened to model entities, both at the aggregated process module level and at the detailed single entity level, one phenomenon was apparent in the graphs, namely, that "bursts" of activity could be observed that tended to calm down after a few versions. The present study concentrates on these bursts, with the aim of determining if they constitute a significant evolution pattern for the V-Modell XT.

2.2 Hypotheses

One possible way to explain the activity bursts (and probably one that would be rather obvious to anyone familiar with software development) is that *primary changes*, that is, changes intended to introduce new features or to restructure the model, often introduce defects that have to be corrected later on, by doing a number of secondary or *follow-up* changes. So, one activity burst would actually consist of a primary change, maybe split into a few versions, and a number of follow-up changes intended to reestablish model correctness. Proving this conjecture is difficult, however, since it would require an objective classification of changes into primary and secondary ones, a task that would most probably require human judgment in many cases.

Still, we can target a weaker form of the conjecture, namely, that changing an area of the model increases the probability of changes happening to the same area in the near future. This would mean that activity bursts observed by visual inspection of the graphs have statistical significance, that is, they cannot be simply explained by chance, or by artifacts of the graphical representation used.

One difficulty that arises here is that of defining what exactly an "area" of the model is. Actually, given the complex structure of the V-Modell XT, there would be a number of potential, reasonable definitions, covering various levels of granularity. As an initial step, we decided to work at a fine level of granularity, and analyze changes at the entity level. The main rationale for this decision is that if we can observe the phenomenon at the entity level, it holds also at least for the larger entity containers, whereas the opposite cannot be stated.

The previous considerations led us to the following two hypotheses:

H1: Changing a process model entity in a particular version increases the probability of changing it again in subsequent versions.

H2: Changing a process model entity at a given date increases the probability of changing it again in the following days.

In the hypotheses, we are not making any statements about the particular way in which the probability of further changes should increase after a change. Our current knowledge of the evolution of this and other models is still too limited to provide a more detailed mathematical model of how a change affects the probability of future changes to the same area.

One conclusion that would immediately follow from the hypotheses above is that changes to an entity in the various versions in the history are not independent events: that is, changes to an entity affect the likelihood of future changes to the same entity. Based on this observation, we formulate our null hypotheses as follows:

$H1_0$: Changes to a process model entity in a particular version are independent from changes to the same entity in all other versions. Moreover, there is a fixed probability p_v of changes occurring to an entity in a particular version, for all versions and for all entities present in each version.

$H2_0$: Changes to a process model entity on a particular day are independent from other changes to the same entity. Moreover, there is a fixed probability p_d of changes occurring to an entity on a particular day, for all days in the studied period and for all entities present in the model during that period.

Notice that, if falsified, these null hypotheses are weaker than the negation of the alternative hypotheses, namely, they would show that there exists a dependency between changes in different versions and at different points in time, but they would not guarantee that the probability of follow-up changes actually increases. We will address this point later.

2.3 Data Preparation

As for the initial study, the first step we took in order to make it possible to analyze the V-Modell's change history statistically was to read a sizable portion of the V-Modell XT's versioning history into our *Evolyzer* model comparison system. Although a description of the internal operation of *Evolyzer* is beyond the scope of this paper (see [4] for details), a short explanation of its workings is in order. The basis of the system is a model database that can contain an arbitrary number of versions of a model. Model versions in the database are represented using the RDF notation [5], and the whole model database can be queried using a subset of the SPARQL [6] query language for RDF.

The main purpose of *Evolyzer* is to allow for comparing model versions from the database. Given two arbitrary versions, the system computes a so-called *comparison model,* which contains all model elements (RDF statements, actually) present in the compared versions, marked with labels indicating whether they are common to both versions or are only present in one of them and, in the latter case, which of the versions they come from. Given the high level of granularity of this comparison, identifying changes in it by direct inspection is generally a difficult task. For this reason, change identification is performed by defining special *change patterns* (see [4] for a detailed explanation) that match particular types of changes in the comparison model. *Evolyzer* provides an efficient interpreter for the pattern language, which can identify instances of a particular pattern in the comparison model of two arbitrary versions.

For the present study, we attempted to read 604 versions from the original versioning repository into our system. These versions were created in a little more than two years' time, with three major and one minor public releases happening during that period. Since *Evolyzer* uses the RDF notation for model representation (this is necessary in order for our comparison technique to work at all), each V-Modell version was mechanically converted from its original XML representation into an RDF model before reading it into the system. This conversion did not add or remove information, nor did it change the level of formalization of the original process description. The conversion process was successful for all but 4 of the 604 analyzed versions. These four versions could not be read into our repository because their corresponding XML files contained syntax errors, and they were replaced by copies of the previous version in order to prevent our system from reporting spurious changes.

After importing the version history, we proceeded to compare the versions pairwise to identify individual changes happening from one version to the next. As changes, we considered the addition or deletion of entities, the addition or deletion of relations between entities, and the alteration of text properties. We identified these changes by defining corresponding change patterns and searching for them in the version comparisons. Information about each of the identified changes, including type, version number, and affected process entities, was encoded in RDF and stored in the repository together with the model versions. This allowed us to easily go from the change information to the actual model contents and back from the models to the changes as necessary for our analysis (see [7] for the details of how this cross-referencing works).

2.4 Data Analysis and Interpretation

In order to test our first hypothesis, we proceeded to look for *pairs of consecutive changes* in the model history. A pair of consecutive changes is defined as a triple (e, v_1, v_2), with e a model entity, and v_1, v_2 version numbers, such that

1. $v_1 < v_2$.

2. v_1 and v_2 contain changes that affect e.

3. no version v, with $v_1 < v < v_2$, contains changes affecting e.

In other words, these are pairs of versions that change the same entity, with none of the versions lying between them affecting the entity.

If our first null hypothesis $H1_0$ holds, the probability of an entity being changed by a particular version has a fixed value p_v. This, in turn, implies that the process of changing an entity over its history can be modeled as a Bernoulli process with probability p_v, where each new version containing the entity is seen as a Bernoulli trial and the trial succeeds if the corresponding version actually changes the entity.

Let us now consider the length $v_2 - v_1$ of a pair of consecutive changes. If the change process is actually a Bernoulli process, the probability *P(l)* of the pair having a particular length *l* would be given by the geometric distribution, that is

$$P(l) = (1 - p_v)^{l-1} \cdot p_v$$

This formula is easily understood as the probability of making $l - 1$ unsuccessful trials followed by one final, successful trial.

In order to determine if this is actually the case in the V-Modell history, we queried our model evolution database to find all text changes (changes to text attributes) affecting entities in the model during the observed period, and, with the help of some simple postprocessing of the query results, identified all pairs of consecutive changes in the history as defined above. We found 2835 individual pairs during the period studied.

Figure 1 is a histogram of the lengths of these pairs, with categories of width 10. 47.3% of the pairs have a length of 10 versions or less, 71% of 50 versions or less, and 90.2% of 170 or less.

As explained above, if the null hypothesis holds, this observed distribution should correspond to the geometric distribution for the probability p_v. In order for a goodness-of-fit test to be possible, it is necessary to estimate a reasonable value for p_v. We tried two different methods for estimating this value.

The first method is based directly on the null hypothesis. If the probability of making changes to any particular entity in any particular version is always the same, we can look at the complete history as a single Bernoulli process in which the individual histories of the various process entities are placed in a single row in some arbitrary order. The total number of trials in that process would then correspond to the sum of the lengths of the individual histories of the entities in the model. Since entities are introduced and deleted along the history, the length history varies from one entity to

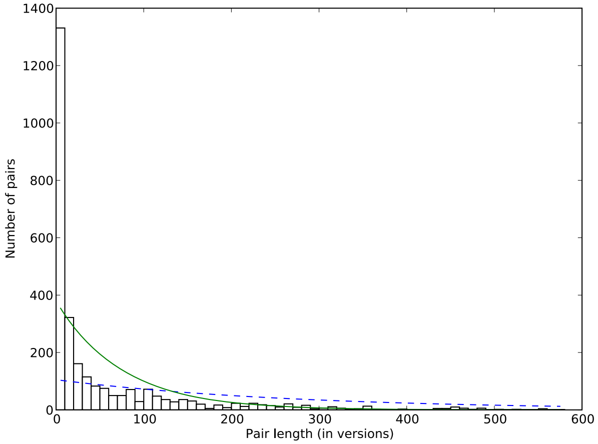


Fig. 1. Distance in versions for consecutive entity changes

the next. Using database queries for the creation and deletion points of entities, we calculated the total number of trials to be approximately 1.150.000 and the total number of changes to be 4248, producing a value of 0.0037 for p_v. The curve for the resulting geometric distribution is compared with the original histogram in Figure 1, where it is shown as a dashed line. A chi-square goodness-of-fit test for this case yielded a p-value smaller than 0.0001.

The second method we tried in order to determine the value of p_v was to use a standard optimization procedure to find a value of p_v that minimizes the chi-square value with respect to the actual data. The p_v value obtained was 0.0133, with a p-value for the goodness-of-fit that is still below 0.0001. The curve for this probability value is shown in Figure 1 as a solid line. Given the very low probability obtained in both cases for the chi-square tests, we can reasonably reject our first null hypothesis, $H1_{0.}$

The direct implication of rejecting the null hypothesis is that we are observing a certain level of dependency among changes. Still, it is not clear if this dependency really implies a *higher* probability of subsequent changes after a change. The comparison in Figure 1, however, shows that the first categories are much higher than those predicted by the estimated geometric distributions. This means that there is a high number of short pairs (indicating changes that are very close to each other) that could not be explained if changes were assumed to happen with a fixed probability. This supports our alternative hypothesis H1.

For the second hypothesis, we extended the previous analysis to also consider the time when changes were made. For each of the identified pairs of consecutive changes, we measured the distance in days between the check-in operations corresponding to the versions v_1 and v_2 in the pair, and discarded those pairs where the changes happened on the same day. This left us with 2324 of the original 2835. Figure 2 contains a histogram of the distances obtained, with the categories corresponding to 10-day intervals.

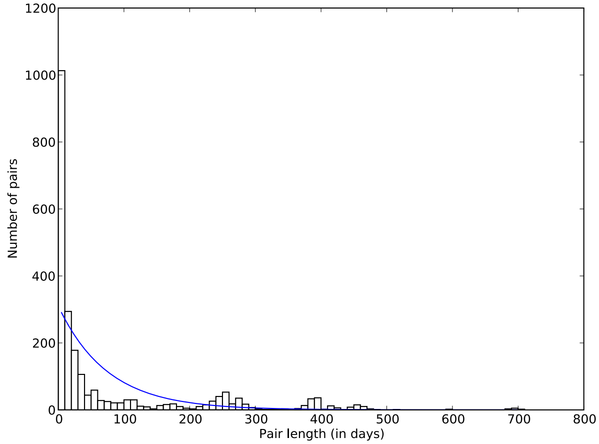


Fig. 2. Distance in days for consecutive entity changes

The first approach used in the previous case to estimate the value of p_v cannot be used here as easily, because the number of entities in the model can vary in the course of a single day. For this reason, we used only the chi-square optimization method to estimate the value of the probability p_d of an entity being changed (at least once) on a particular day. The resulting curve can be seen as a solid line in Figure 2. The actual resulting value was 0.0133 with a goodness-of-fit p-value also below 0.0001. The conclusion is analog to the one for the previous case: The null hypothesis $H2_0$ can also be rejected in this case. Similarly, the pronounced peak in the first category observed in the graph over the theoretical distribution contributes the remaining support to our alternative hypothesis H2.

2.5 Threats to Validity and Limitations

Although the high significance of the previous results clearly shows that the null hypotheses can be rejected, at least in the stated form, the question remains of whether the assumption of a constant change probability over all entities in the model, and for all model versions (or days in the studied period) actually holds in practice. For the time case, for instance, activity increases around releases, which would lead to shorter consecutive change distances in days for the time periods around releases. This effect, however, would be only observable when measuring change distances in days, but not when measuring them in versions, so the fact that the effect is observed in both of them actually speaks against this risk.

It is also quite possible that certain entity types may have higher change probabilities, and this may explain at least some instances of the short change distances we are observing. In particular, certain entity types contain more text attributes, or tend to have longer text attribute contents, thus increasing the probability of changes to them. One option for looking into this in more detail is to identify consecutive changes to the individual attribute instances. If change distances are still short for this case, we can more safely assert that the effect observed at the entity level is not only explained through differences among entity types.

Figure 3 presents the version and time histograms for pairs of consecutive changes to individual text attributes in the model. These pairs are defined in a similar way as for the entity case, with the exception of versions in the pair having to affect both the same entity and the same attribute in the entity. 2749 pairs were found for the version-based analysis, and 2225 for the time-based analysis. The best-fit geometric distributions have probabilities of 0.0182 and 0.0134 respectively. Both of them yielded p-values for the chi-square test below 0.0001. These results are consistent with our analysis for the entity case, namely, that changes to single attributes also increase the probability of future changes to the same attribute.

A larger, much more difficult question is related to external validity. Since model evolution is just starting to be studied, it is premature to say that the results observed for the V-Modell XT can be generalized to other similar models. However, one can assume that the results would apply, at least to some extent, to models such as the Rational Unified Process (RUP) [8], which have a similar purpose and level of complexity. Further studies in this direction would be very valuable.

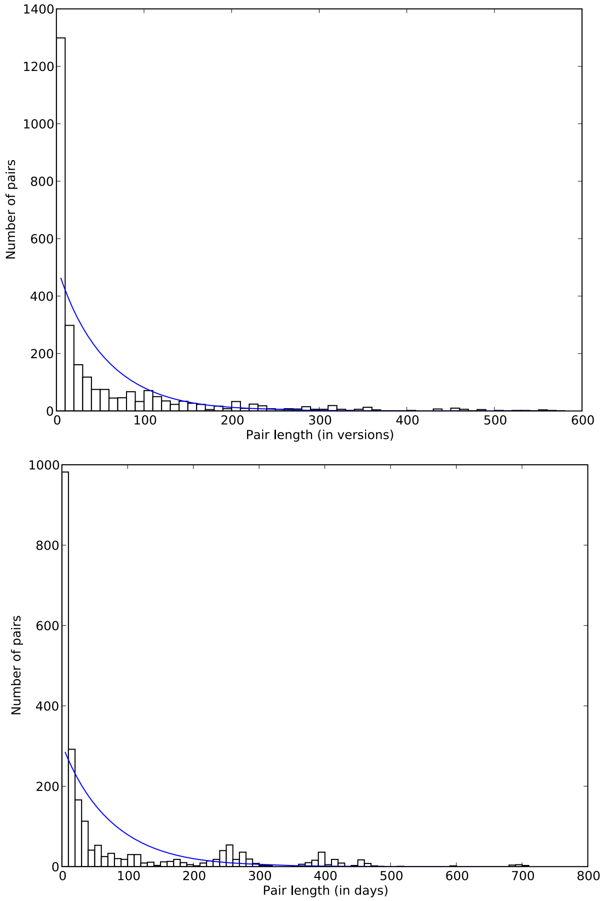


Fig. 3. Distance in versions and days for consecutive text attribute changes

3 Related Work

Much support and several studies have been dedicated to understanding software evolution. Many examples of such work can be found in recent workshops and conferences [9-11]. Most of these studies have concentrated on confirming Lehman's [12] and Parnas' [13] findings, by examining successive source code releases, or examining change data stored in source code control systems. Such studies are frequently performed with the support of advanced data mining techniques [14, 15], as is the case for both the product and process communities. In the product community, studies are performed for purposes such as understanding the evolution of programs and the programs themselves [16], detecting evolutionary coupling between files [17] and model elements (e.g., classes) [18], suggesting and predicting likely changes, preventing errors due to incomplete changes, or detecting coupling undetectable by program analysis [19].

Jazayeri [20] stated that “Individual software products age while our understanding of them and, as a result, their models (and meta-models) evolve”, and encouraged the community to move the focus of studies towards the evolution of models and meta-models. As mentioned in the introduction of this paper, we currently observe a lack of empirical studies on the evolution of large models. In addition to the preliminary study that gave rise to the present work [1], two of the authors performed an exploratory study [21] with the goal of understanding the nature of process model changes in the context of the aerospace industry. That study presented the most important issues that motivated process engineers to change an aerospace software process standard.

4 Conclusions and Outlook

The empirical study presented in this paper had the purpose of determining whether changes to entities in the German V-Modell XT software process standard increased the probability of subsequent changes to the same entities, both in time and in the version sequence. The basic conjecture giving rise to this research question is that certain changes to the model introduce defects that have to be corrected in a number of follow-up changes, thus producing “bursts” of activity that are observable in the model history.

The data used for the study corresponded to a set of 604 consecutive versions, containing changes performed to the V-Modell XT from October 2004 to October 2006. The differences from one version to the next were calculated automatically by means of the focused identification of changes supported by our *Evolyzer* tool. The detailed description of this change identification process, as well as the definition of the change types used here, can be found in [4] and [7], respectively.

The descriptive analyses and the statistical tests presented in this article confirm the hypothesis that changes to an entity increase the probability of further changes in the future. Although this does not prove that our underlying model of primary and follow-up changes holds, it is a first step towards providing evidence in this direction.

In this sense, also, the results of the present study support our assumption that there are clear behavior similarities between code and model evolution (see [22] for an example of a study that yields similar results for software systems). This assumption has a number of consequences. On the one hand, it suggests that existing mechanisms that facilitate code maintenance can be potentially applied to model maintenance. For instance, encapsulation mechanisms help to isolate stable code parts from constantly changing parts and, therefore, help to gain better control of maintenance activities. In the case of models, the same encapsulation can be applied by (re-)structuring them in such a way that the contents of stable model entities are reused without change. There are also implications for project planning, since, as in the case of code, complex changes seem to be very likely to “destabilize” the model by introducing defects of various types. This would mean that project managers must plan for a stabilization period after introducing complex changes.

We see model evolution studies as valuable input for better support of model maintenance in the future. Particularly in the area of version management, which is clearly related to our work on model comparison, we believe that models can benefit to a large extent from existing code version management techniques and tools. However, the implementation of such techniques for models presents a number of theoretical and technical challenges that are not present in standard code version management. Our techniques for focused change identification, and the *Evolyzer* tool that realizes them, constitute a new proposal for this kind of support. The viability of this proposal can be seen in the fact that these techniques and tools provided us with the capabilities to perform the study presented in this paper.

The similarities observed between model and code evolution are also a motivation for performing future studies based on research questions already posed in the latter area. For example, in a recent code evolution study [19], Zimmermann et al. attempted to find hidden dependencies in a large software system by looking for pairs of program elements that have a strong tendency to be changed simultaneously, e.g., when one of them is changed in a given version, there is a high probability that the other one is also changed in the same version. In our opinion, a similar study could be viably extended to models, with the goal of discovering hidden evolution connections between entities that could not be found using the techniques that we have applied so far.

Acknowledgments

We especially thank our colleagues Marcus Ciolkowski and Jens Heidrich from Fraunhofer IESE, who provided many valuable ideas and enriching discussions during the preparation of this paper. We would also like to thank Sonnhild Namingha, also from Fraunhofer IESE, for proofreading the paper.

During our work with the V-Modell XT, we had support from several members of the V-Modell development team. We would particularly like to mention Professor Andreas Rausch, Christian Bartelt, Michael Deynet, and Thomas Ternité, all currently at the Technical University of Clausthal, Germany.

This work was supported in part by SoftDiff, a project financed by the Fraunhofer Challenge Program. This work was also partially supported by the *Stiftung Rheinland-Pfalz für Innovation* through the Q-VISIT project (*Qualitätsorientierte Visuelle Software Inspektion*).

References

1. Soto, M., Ocampo, A., Münch, J.: The secret life of a process description: A look into the evolution of a large process model. In: Wang, Q., Pfahl, D., Raffo, D.M. (eds.) ICSP 2008. LNCS, vol. 5007, pp. 257–268. Springer, Heidelberg (2008)
2. V-Modell® XT, `http://www.v-modell.iabg.de/` (last checked 2007-12-20)
3. Royce, W.W.: Managing the development of large software systems: concepts and techniques. In: Proceedings of the 9th International Conference on Software Engineering. IEEE Computer Society Press, Los Alamitos (1987)
4. Soto, M., Münch, J.: Focused Identification of Process Model Changes. In: Wang, Q., Pfahl, D., Raffo, D.M. (eds.) ICSP 2007. LNCS, vol. 4470, Springer, Heidelberg (2007)
5. Manola, F., Miller, E. (eds.): RDF Primer. W3C Recommendation (2004), `http://www.w3.org/TR/rdf-primer/` (last checked 2007-12-20)

6. Prud'hommeaux, E., Seaborne, A. (eds.): SPARQL Query Language for RDF. W3C Working Draft (2006), `http://www.w3.org/TR/rdf-sparql-query/` (last checked 2006-10-22)
7. Ocampo, A., Soto, M.: Connecting the Rationale for Changes to the Evolution of a Process. In: Münch, J., Abrahamsson, P. (eds.) PROFES 2007. LNCS, vol. 4589, pp. 160–174. Springer, Heidelberg (2007)
8. RUP. Rationale Unified Process, `http://www-306.ibm.com/software/awdtools/rup/` (last checked 2008-08-06)
9. Di Penta, M., Lanza, M.: Ninth international workshop on Principles of software evolution (IWPSE) (2007) ISBN:978-1-59593-722-3
10. 8th International Workshop on Principles of Software Evolution (IWPSE 2005), Lisbon, Portugal, September 5-7, 2005. IEEE Computer Society, Los Alamitos (2005) ISBN 0-7695-2349-8
11. 7th international Workshop Principles of Software Evolution. IWPSE, September 06 - 07, 2004, vol. 8. IEEE Computer Society, Washington, `http://dx.doi.org/10.1109/IWPSE.2004.15`
12. Lehman, M.M., Belady, L.A. (eds.): Program Evolution: Processes of Software Change. Academic Press Professional, Inc., London (1985)
13. Parnas, D.L.: Software Aging. In: Proceedings of the 16th International Conference on Software Engineering (ICSE 1994), Sorrento, Italy, pp. 279–287 (1994)
14. van der Aalst, W.W.T., Maruster, L.: Workflow Mining: Discovering Process Models from Event Logs. IEEE Transactions on Knowledge and Data Engineering 2004 16(9), 1128–1142 (2004)
15. Eiben, A.E., Smith, J.E.: Introduction to Evolutionary Computing. In: Natural Computing. Springer, Berlin (2003)
16. Ball, T., Kim, J.M., Porter, A.A., Siy, H.P.: If Your Version Control System Could Talk. In: Proc. ICSE Workshop Process Modelling and Empirical Studies of Software Eng. (1997)
17. Gall, H., Hajek, K., Jazayeri, M.: Detection of Logical Coupling Based on Product Release History. In: Proc. Int'l Conf. Software Maintenance (ICSM 1998), pp. 190–198 (November 1998)
18. Bieman, J.M., Andrews, A.A., Yang, H.J.: Understanding Change-Pronenes. In: OO Software through Visualization. In: Proc. 11th Int'l Workshop Program Comprehension, pp. 44–53 (May 2003)
19. Zimmermann, T., Zeller, A., Weissgerber, P., Diehl, S.: Mining version histories to guide software changes. IEEE Transactions on Software Engineering 31(6), 429–445 (2005) Digital Object Identifier 10.1109/TSE.2005.72
20. Jazayeri, M.: Species evolve, individuals age Invited Keynote Talk. In: 8th International Workshop on Principles of Software Evolution (IWPSE 2005), Lisbon, Portugal, September 5-7, 2005. IEEE Computer Society, Los Alamitos (2005)
21. Ocampo, A., Münch, J.: Process Evolution Supported by Rationale: An Empirical Investigation of Process Changes. In: Wang, Q., Pfahl, D., Raffo, D.M., Wernick, P. (eds.) SPW/ProSim 2006. LNCS, vol. 3966, pp. 334–341. Springer, Heidelberg (2006)
22. Burd, E., Munro, M.: Evaluating the evolution of a C application. In: proceedings International Workshop on Principles of Software Evolution, `http://dontaku.csce.kyushu.ac.jp/IWPSE99/Proceedings`

Process Trustworthiness as a Capability Indicator for Measuring and Improving Software Trustworthiness

Ye Yang, Qing Wang, and Mingshu Li

Lab for Internet Software Technology,
Institute of Software Chinese Academy of Sciences
#4 South 4th Street, ZhongGuanCun, Beijing 100190, China
{ye,wq,mingshu}@itechs.iscas.ac.cn

Abstract. Due to increasing system decentralization, component heterogeneity, and interface complexities, many trustworthiness challenges become more and more complicated and intertwined. Moreover, there is a lack of common understanding of software trustworthiness and its related development methodology. This paper reports preliminary results from an ongoing collaborative research project among 6 international research units, which aims at exploring theories and methods for enhancing existing software process techniques for trustworthy software development. The results consist in two parts: 1) the proposal of a new concept of Process Trustworthiness, as a capability indicator to measure the relative degree of confidence for certain software processes to deliver trustworthy software; and 2) the introduction of the architecture of a Trustworthy Process Management Framework (TPMF) toolkit for process runtime support in measuring and improving process trustworthiness in order to assess and assure software trustworthiness.

Keywords: Software Trustworthiness, Process Trustworthiness, Process Trustworthiness Model, Measurement Model, Risk Management.

1 Introduction

In his ICSE 2006 Keynote speech [1], Boehm pointed out the increasing trend of software criticality and dependability as one of the eight surprise-free software trends. Over the past 50 years, different strategies such as formal methods, security assurance techniques, defect prediction, failure mode and effects analysis, testing methods, and software assurance techniques have been proposed to address different aspects of trustworthy challenges. Based on these studies, numerous quality categories and attributes have been studied as major factors influencing on software trustworthiness. Among them are included functionality, reliability, safety, usability, security, portability, and maintainability, etc. Though there is a growing consensus that a "Trustworthy" software system is characterized as one that satisfies a collection of critical quality attributes, there is a lack of common understanding of software trustworthiness.

As software has been playing an ever-increasing role in many domains and systems, assessing and improving software trustworthiness becomes more and more complicated and difficult. For example, Nelson et. al. [2] reported the increasing trend

Q. Wang et al. (Eds.): ICSP 2009, LNCS 5543, pp. 389–401, 2009.

of US weapon system's software dependencies from less than 10% in 1960's to 80% in 2000, and consequently the projects experienced seriously challenges in delivering trustworthy software. The author concluded that two areas of research are of major importance: requirement engineering and cost estimation, which demand for enhanced theory and methods.

In January 2008, a collaborative research project among 6 international research units was initiated to develop of a Trustworthy Software Development Methodology with enhanced theories and methods to support trustworthy software development, and the implementation of a management supporting toolkit. This paper reports preliminary results from this ongoing collaborative research project. First, it examines trustworthiness-related definitions in existing literature and proposes the concept of **Process Trustworthiness** as a capability indicator to measure the relative degree of confidence for certain software process to deliver a trustworthy product. Second, a **Trustworthy Process Management Framework** is introduced to demonstrate the conceptual structure in establishing and managing the process trustworthiness for assessing and improving software trustworthiness.

The paper is organized as follows: Section 2 discusses related work; Section 3 introduces the context and basic working definitions of our study; Section 4 presents the process trustworthiness model; Section 5 presents the major components in the Trustworthiness Process Management Framework from our ongoing work; and Section 6 is the conclusions.

2 Related Work

Over the past few decades, many studies mentioned aspects like security, reliability, safety, maintainability, usability, etc. Some of the discussions also included measurement and evaluation on a particular aspect along with support technologies to ensure implementation. Several works discuss the use of formal methods and model validation techniques to enhance the trustworthiness about a particular aspect. However, the understanding of the concept of software trustworthiness is not uniform among researchers.

For instances, Trusted Computer National Evaluation Criteria (TCSEC) restricted trustworthiness to security as the only attribute to consider [3]. Parnas et al. defines software trustworthiness as level of appropriateness of using software engineering techniques to reduce failure rates, including techniques to enhance testing, reviews, and inspections [4]. Common Criteria [7] provides a complete framework of evaluating software with a mere focus on security.

Inspired by traditional manufacturing industry, software process-oriented methodologies such as Trusted Software Methodology (TSM) [5], CMMI [6], SSE-CMM [11], and ISO 9001[14] all focus on the enhancement of software processes as a means for enhancing resultant software. TSM was developed and evolved by US National Security Agency and three other organizations, to define a note for software trustworthiness and provide a means for assessing and improving it. It provides 44 generally-accepted trust principles and practical process guidance in examining and evaluating trustworthiness threats during the development process. CMMI is the most widely adopted process management framework with accumulated body of knowledge on various development disciplines. However, neither CMMI nor ISO 9001 do explicitly address any security or trustworthiness related issues. While SSE-CMM [11] extends the original CMM to support process improvement, capability evaluation and assurance in any organization involving security engineering, it does not address other aspects of software trustworthiness.

3 Context and Basic Definitions

3.1 Motivations and Objectives

There are a number of issues that motivated our study, including:

- Currently there is lack of common understanding on the definitive scope of software trustworthiness;
- Quality attributes such as security, reliability, and dependability etc., are studies individually;
- Development guidelines are related to single quality attributes and few instructions are available when multiple attributes are considered;
- Few studies have been conducted towards better understanding of trustworthy software development.

Our collaborative research project aims at exploring theories and methods for enhancing, improving, and innovating software process techniques, supporting the development and production of large-scale complex trustworthy software. As the preliminary working progress from this project, the work reported in this paper mainly addresses the following objectives:

- Investigate different aspects of trustworthy software methodologies;
- Propose common definition for trustworthiness of software development;
- Try to provide an overall scale of process trustworthiness as an indicator for assuring software trustworthiness;
- Stimulate discussions from both researchers and practitioners on related topics.

3.2 Previous Work

We have conducted a literature review to determine major trustworthiness influencing factors from existing literature on software trustworthiness or software assurance. There were three aspects that we inspected in a particular research study: 1) focused quality attributes; 2) process or product related guidelines; and 3) metrics used to help in providing a complete analysis on the corresponding trustworthiness attributes. Table 1 summarizes the evaluation results from 8 major resources including [15]:

- Trusted Software Methodologies (TSM)[5]
- Common Criteria (CC)[7]
- Software Assurance for Project Management (SA-PM)[8]
- Software Security Assurance (SSA)[9].
- DoD/FAA Security & Safety Extension to CMMI and iCMM (SSE-FAA)[10]
- System Security Engineering for CMM (SSE-CMM)[11]
- ISO 9126 [12]
- ISO 27000 [13]

Table 1. Summary of results from our literature review

Name	Attribute(s)	Process/Products	Metrics/Evaluation
TSM	All possible quality attributes	44 Trust Principles	6 Trust Classes
CC	Security	Target of Evaluation, Security Requirements to Classes & Families	7 Evaluated Assurance Level (EAL1-7)
SA-PM	No clearly identified attribute	Activities needed to maintain a trusted workflow, such as risk management and configuration management	COSECMO for security cost estimation
SSA	Security	Many security enhanced methodology such as MS Trustworthy Computing SDL and Seven Touchpoints.	Different metrics & models such as Practical Security Measurement
SSE-FAA	Security and Safety	Extension of Application Area on Security and Safety: 4 Application Practice Goals	Use the same evaluation as in iCMM or CMMI
SSE-CMM	Security	22 Security Process Areas	6 Capability Levels (0-5)
ISO 9126	All possible quality attributes	6 Major Characteristics	Part 2, 3, and 4 define internal, external, and quality in user metrics
ISO 27000	Security	6 different sets of standards,, only the first two are established	Not available

As introduced in Section 2 and summarized in Table 1, TSM defines trust principles and provides guidance in evaluating trustworthiness threats in the development process, however, it does not provide a runtime management framework for assuring software trustworthiness. SSE-CMM raises a similar idea of "Capability-based assurance, that is, trustworthiness based on confidence in the maturity of an engineering group's security practices and process" [11]. However, with an intensive focus on security and/or safety, CC, SSE-CMM, SSA, SSE-FAA, ISO 27000 do not address the evaluation and assurance of any other quality attributes.

Based on our analysis in [15], we concluded that TSM is in the closest alignment with our research objectives. It provides the basis for us to build our working definitions for software trustworthiness. And we believe that there is an opportunity in integrating the trust principles into process management methodologies such as CMMI to enable software managers to make reasonable decisions that have positive impact on their development approaches.

3.3 Basic Definitions

3.3.1 Software Trustworthiness and Trustworthy Software

We adapted our first working definitions of trustworthiness as "*level of confidence or degree of confidence*", and **Software Trustworthiness** as *"degree of confidence that the software satisfies its requirements*" from TSM [5]. Furthermore, we refer to ISO 9126 for a complete list of quality characteristics/attributes which should be taken into consideration when planning and managing for a specific set of trustworthiness attributes for a particular trustworthy software development project.

In other words, software trustworthiness is a circumstance-dependent measure of how the delivered product satisfies the stakeholders' set of expectations. This definition emphasizes that software trustworthiness is highly dependent on the prescribed, yet evolving, set of requirements, technical decisions, and management decisions throughout the development process life cycle. It may include functionality, reliability, safety, usability, security, portability, and maintainability etc. according to stakeholder's value propositions and negotiation results [1]. Each of these abilities represents a dimension of trustworthy software and also introduces a source of vulnerabilities that can threaten the normal functions of a software product.

Additionally, we define a **Trustworthy Product (i.e. work product, software)** as a product *(i.e. work product, software) that satisfies a range of its trustworthiness objectives established based on its requirements.*

3.3.2 Process Trustworthiness and Trustworthy Process

In CMMI [6], a widely recognized notion of process capability has been defined as "the range of expected results that can be achieved by following a process." This term is a means to quantitatively denote and compare the performance of software processes in satisfying product quality and process performance objectives with respect to the 22 process areas. While CMMI assembles the best practices on what to do to assure the delivered product quality, it does not tell how to do. Nor does it emphasize trustworthy software development.

To that end, we extend the concept of process capability and propose the term of **Process Trustworthiness** as an overall scale to measure process capability in developing trustworthy software. It is defined as: *Process trustworthiness is the degree of confidence that the software process produces expected trustworthy work products that satisfy their requirements.*

Meanwhile, our working definition for a **Trustworthy Process** is *a capable process that produces a range of trustworthy products.*

By arriving at these definitions, we set a common point of view for us to continue with our research in drafting the scope that guides us to gather sufficient information to help design a software trustworthiness framework. However, we do believe that this draft version will be refined many times as we find out more about the different aspects related to this subject.

4 Modeling Process Trustworthiness

4.1 Meta-model of a Trustworthy Process

Fig.1 illustrates the meta-model of a trustworthy process based on the terms proposed in Section 3. As shown in the Fig.1, a trustworthy process is modeled as a capable process that demonstrates certain level of process trustworthiness in producing a trustworthy product. The "Depends on" association between a trustworthy work product/software and its requirements indicates that, to assess the trustworthiness of certain product, the evaluation criteria should be established based on software requirements or work product requirements.

As our research goal is to develop a process-oriented Trustworthy Process Management Framework, there are two key issues for elaborating and assessing the applicability and effectiveness of existing process technologies: 1) how to model the dependencies among trustworthy software, trustworthy work product, and trustworthy process, and 2) how to model the dependencies between requirements and trustworthy product.

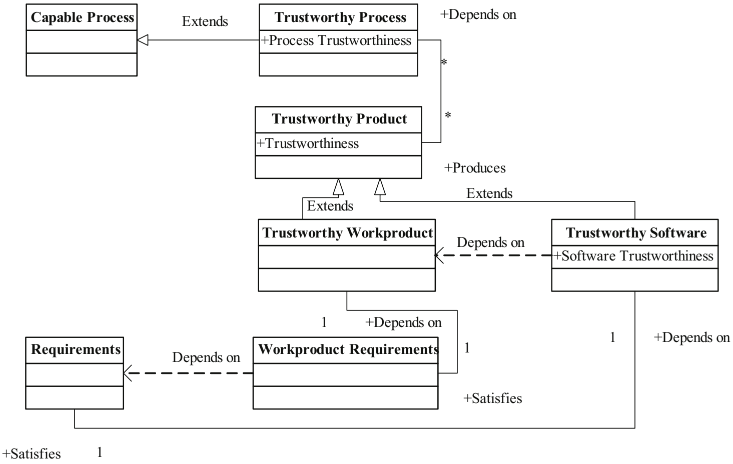


Fig. 1. Meta-model of a trustworthy process

The rest of this paper will focus on the introduction of our proposal in addressing the first issue, and the solution to address the second issue is planned as in-progress study and is not in the scope of this paper.

4.2 Modeling Software Trustworthiness

As depicted in Fig.1, requirements are the basis to establish criteria for evaluating the level of satisfaction of software trustworthiness. In particular, the establishment of software trustworthiness objectives may consist of understanding stakeholders' expectations, selecting critical trustworthy attributes, and setting appropriate target profiles for the trustworthiness objectives. We model software trustworthiness as follows:

Software trustworthiness = ({Attribute}, {Traceability}, {Objective}, {Priority}), where

- Attribute set is a subset of critical quality attributes extracted from software requirements, e.g. a subset tailored from the list of characteristics or sub-characteristics defined in ISO 9126;
- Traceability set reflects the mapping between an attribute and the originating requirements;
- Objective set is a target profile of trustworthiness level(either qualitative or quantitative) for each attribute defined in {Attribute};
- Priority set captures the relative degree of importance for each attribute of the end product meeting its trustworthiness objective.

This software trustworthiness model can be applied to both end software product and intermediate work products in order to set trustworthiness objectives based on software requirements and work product requirements. Also as indicated by Fig.1, whether software can be evaluated as trustworthy depends on requirements, process trustworthiness, and work product requirements.

4.3 Modeling Processes Trustworthiness

We extend the Process Area (PA) concept from CMMI, defined as "*a cluster of related practices that, when implemented collectively, satisfy a set of goals considered important for making improvement in that area* [6]", to integrate trust principles, trust levels, and evaluation methods from TSM [5] into Trustworthy Process Area. The benefits of this integration are two-fold: 1) to include provision for a specific development process to deal with untrustworthy process factors, e.g. less capable or malicious developers; and 2) to provide an organizational improvement infrastructure for process trustworthiness.

Trust principles are a collection of familiar, generally accepted software engineering and security principles summarized in [5]. We match these trust principles with relevant process areas defined in CMMI. Each identified PA is attached with a relevant set of trust principles and extended into a corresponding Trustworthy Process Area (TPA). Moreover, we create new TPA to include some important trust principles that can not be easily mapped to any PAs in CMMI. With this extension the Process trustworthiness of an instantiated TPA can then be evaluated according to definitions, selection and evaluation criteria of 6 trust classes (i.e. T0-T5) defined in [5]. Fig.2 (A) shows the structure of a TPA.

Typically, there are three dimensions to be taken into consideration when planning and managing for a trustworthy process, each introducing certain major threats to the software trustworthiness during the development life cycle [30]. These include:

- Process Entities: An entity of the process is the person responsible to perform the process or any organization assets used in performing the process;
- Process Behaviors: Process behaviors are those manners which plan, monitor, measure, review, evaluate and execute the process; and
- Process Products: A process product is an artifact produced during the development process. It could be either a work product (e.g. document, checklist, record, etc.) or a component (e.g. source file, executable package).

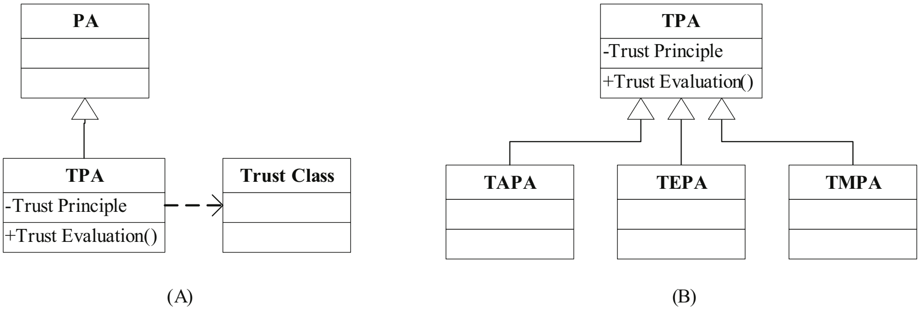


Fig. 2. Trustworthy Process Area Structure

Therefore, as shown in Fig.2 (B), we further model the TPAs into three categories with a focus on addressing threats from each of above mentioned dimensions:

- Trustworthiness Assurance Process Area (TAPA): The purpose of TAPA is to assure the organization to have the proper process entities to execute tasks in the proper way with the proper infrastructure support. This process area provides necessary trustworthy goals, which describes characteristics of a trusted work environment, effective policy, and capable personnel as well as other assurance measures in setting up the organization for its trustworthy process [29].

- Trustworthiness Monitoring Process Areas (TMPAs): Management and supporting-related process areas from CMMI are further extended into TMPAs, including project management, risk management, configuration management, etc., with necessary extension to ensure the trustworthiness of process behaviors and trustworthy process execution, e.g. to monitor the progress of the project, cost, quality and other attributes; to confirm the correct realization and evolution of the planned trustworthy attributes; to control project risks, etc.
- Trustworthiness Engineering Process Areas (TEPAs): TEPAs are proposed to involve more rigid activities to ensure work products satisfying trustworthiness requirements. An example set of TEPAs include trustworthy requirements, trustworthy design, trustworthy code, trustworthy test, and so on. These enhancements embody additional content and amount of engineering activities, as well as corresponding restrictive conditions. For example, a higher security requirement level will dramatically increase the scale (requirement, design, code, and test) of software product compared to a lower level. Frequently, formal methods will be applied to verify requirement, design, and code to ensure trustworthiness.

4.4 Trustworthiness Measurement Model

To serve as the basis for planning and managing a software process targeting at certain trustworthiness level, the measurement model provides an infrastructure to facilitate the composition of process measures as valid indicators for trustworthy evidence and performance objectives established in the Process Trustworthiness Model, as well as the collection and analysis of metrics data for assessing and improving process trustworthiness and software trustworthiness. The structure of the Trustworthiness Measurement Model is shown in Fig. 3.

The Evidence Model in Fig.3 establishes and maintains the relationship between a set of process measures and a valid indicator, provided as basis for analyzing whether valid evidences exist for assessing the implementation of a particular trust principle. While the Performance Model establishes and maintains the quantitative performance baselines and control limits for selected processes with respect to the selected trustworthiness requirements.

We have also developed a set of direct and derived metrics to measure software trustworthiness from process entity, process behavior, and process product dimensions [16], as summarized in Table 2. It does not mean to be a comprehensive list, but integrates our previous work and results from a thorough literature review on software measurement [17, 18] and trustworthy quality characteristics [12].

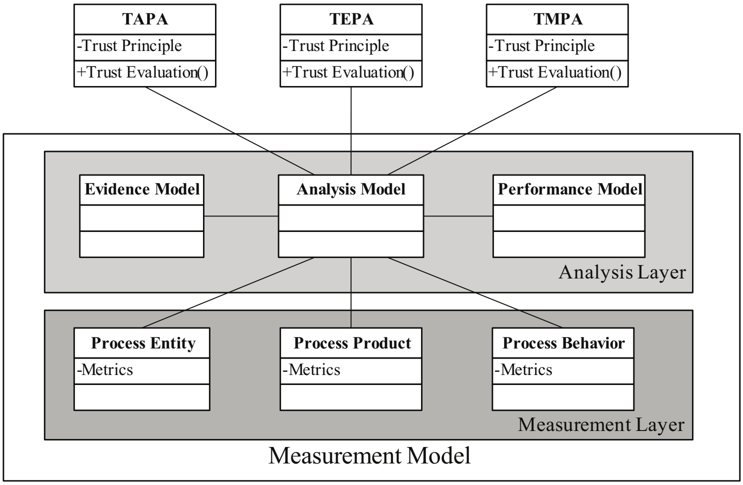


Fig. 3. Structure of Trustworthy Measurement Model

Analysis Models captures the relationships between different combinations of metrics from the Measurement Layer and the evidence indicators or performance indicators established in the Analysis Layer. Our previous work has led to the development of a series of Analysis Models, e.g. quantitative testing process management models [19, 20] and personnel process and ability management [21]. These methods can be extended in appropriate ways to support software organizations to establish their process quantitative management models based on historic data and use these models to measure and assess their process trustworthiness and software trustworthiness.

Table 2. Summary of trustworthiness measures in the Trustworthiness Measurement Model

Dimensions	Quality	Cost	Schedule	Example Measures
Process Entity	8	25	0	Capability, experience, defect detection efficiency
Process Behavior	33	25	12	For each phase/activity: defect injection ratio, defect elimination ratio, productivity
Process Product	37	14	0	Complexity, reliability, number of defects

5 Towards a Trustworthy Process Management Framework

This section, we will present the initial architecture of a Trustworthy Process Management Framework (TPMF), to facilitate the management and assurance for trustworthy software development. This reflects part of results from our in-progress work. TPMF extends a previous toolkit called SoftPM [22], which a platform for software process management, to incorporate the Software Trustworthiness Model, Process Trustworthiness Model and Measurement Model introduced in Section 3. Fig.4

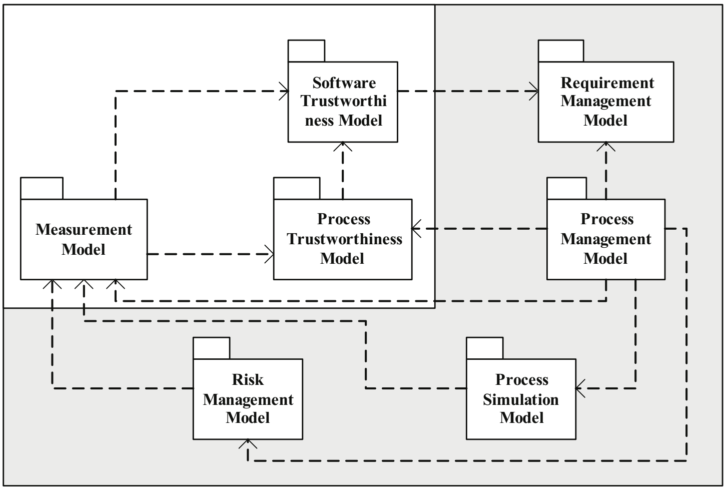

Software Trustworthi ness Model
Requirement Management Model
Measurement Model
Process Trustworthiness Model
Process Management Model
Risk Management Model
Process Simulation Model

Fig. 4. Trustworthy Process Management Framework

illustrates the dependencies among them and four other infrastructural modules in TPMF. Next, we will briefly introduce these infrastructural modules (as shown in the shaded area) and their functions in supporting software organizations for assuring trustworthy software development.

5.1 Requirement Management Model

In TPMF, Requirement Management Model is the basis for establishing software trustworthiness, and process management model also depends on it in establishing and managing project development plans.

Our research is focused on requirement evolution management to support the definition and maintenance of the software trustworthiness model. Different technologies such as WikiWinWin [23], dynamic requirements traceability [24], and empirically-based requirement quality assessment method [25] are being developed and evolving to facilitate the requirement negotiation, requirement traceability, and requirement change impact analysis.

5.2 Process Management Model

Any process models would need to be appropriately assembled and instantiated with more detail information before they can be followed for execution. Such information includes resource, schedule, work products, constraint condition and trustworthy requirement and related quality criteria. The purpose of Process Management Model is to provide such runtime instantiation and management support in order to enable the process trustworthiness model meaningful and applicable.

Our work in this direction is to study and develop mechanisms and methods to extend SoftPM [22] to support the assembly, tailoring, integration of the specific instantiation from the trustworthy process model with respect to particular project circumstance.

5.3 Risk Management Model

A proactive risk management strategy is a key component in the Trustworthy Process Management Framework. Early identification of key risks and their relationships has a very crucial role ensuring the trustworthy realization of the project objectives. Through studying the mechanisms behind risk transfers and risk accumulation, we can establish a set of key risk and corresponding rules in support of decision-making.

In our study, we are developing mechanisms to identify key risk indicators and major risk patterns based on the measurement model, through data mining techniques such as feature selection and support vector machines to reveal patterns and interrelationships between certain risk and process/product metrics [28].

5.4 Process Simulation Model

Process simulation model is another critical component. On the one hand, we intend to integrate existing software process simulation modeling techniques, and develop methods and tools to support the change impact analysis given certain process conditions and trustworthiness objectives [26]. On the other hand, we investigate the process optimization methods as risk mitigation strategies with respect to actual situation and status, in recognition of the key risk factors measured on the basis of the project life cycle and the key risk control point [27].

6 Conclusions

Nowadays, software has been widely applied and plays an increasingly key role in almost every business domain. Trustworthiness requires a process-oriented view as first noted in TSM, where the behavior and state of software products to be predictable and controllable in its life cycle.

This paper proposed a set of trustworthiness-related definitions, and a Trustworthy Process Management framework as preliminary results from an ongoing international collaborative project. The definitions are adopted and derived from a literature review on a number of related studies; and the TPMF aims at, from a software process-oriented viewpoint, better supporting the measurement and assurance of software trustworthiness through the measurement and assessment of process trustworthiness of the development process.

Acknowledgements

This work is supported by the National Natural Science Foundation of China under Grant No. 90718042; the National Hi-Tech R&D Plan of China under Grant Nos. 2007AA010303; the National Basic Research Program (973 program) under Grant No. 2007CB310802.

References

1. Boehm, B.: A view of 20th and 21st century software engineering. In: International Conference on Software Engineering. Proceedings of the 28th international conference on Software Engineering, pp., 12–29 (2006)
2. Nelson, M., Clark, J., Spurlock, M.A.: Curing the Software Requirements and Cost Estimating Blues. PM: November/December (1999)
3. Department of Defense, National Computer Security Center: Trusted Computer System Evaluation Criteria. DoD 5200.28-STD (1985)
4. Parnas, D., et al.: Evaluation of Safety-Critical Software. CACM 33(6), 636–648 (1990)
5. Amoroso, E.C.T., Watson, J., Weiss, J.: A process-oriented methodology for assessing and improving software trustworthiness. In: Proceedings of the 2nd ACM Conference on Computer and communications security, Virginia, USA, pp. 39–50 (1994)
6. Capability Maturity Model Integration Version 1.2 Overview, `http://www.sei.cmu.edu/cmmi/adoption/pdf/cmmi-overview07.pdf`
7. Common Criteria Portal, `http://www.commoncriteriaportal.org/`
8. DACS, Software Project Management for Software Assurance: A State-of-the-Art-Report, September 30 (2007)
9. DACS and IATAC, Software Security Assurance: A State-of-the-Art-Report, July 31 (2007)
10. United States Federal Aviation Administration, Safety and Security Extension for integrated Capability Maturity Model (September 2004)
11. CMU, Systems Security Engineering Capability Maturity Model SSE-CMM: Model Description Document, Version 3.0, June 15 (2003)
12. International Standards Organization, ISO 9126, Ist edn. (2001)
13. International Standards Organization, ISO 27000, Ist edn. (2005)
14. International Standards Organization, ISO 9001, 2nd edn. (2005)
15. Tan, T., He, M., et al.: An Analysis to Understand Software Trustworthiness. Accepted by the 2008 International Symposium on Trusted Computing, China (November 2008)
16. Shu, F., Jiang, N., Gou, L.: Technical Report: A Trustworthiness Measurement Model. ISCAS/iTechs Technical Report #106 (November 2008)
17. Jones, C.: Applied Software Measurement: Assuring Productivity and Quality. McGraw-Hill, New York (1997)
18. Boehm, B.W., et al.: Software Cost Estimation with COCOMO II. Prentice-Hall, NY (2000)
19. Wang, Q., Gou, L., et al.: An Empirical Study on Establishing Quantitative Management Model for Testing Process. In: Wang, Q., Pfahl, D., Raffo, D.M. (eds.) ICSP 2007. LNCS, vol. 4470, pp. 233–245. Springer, Heidelberg (2007)
20. Gou, L., Wang, Q., et al.: Quantitatively Managing Defects for Iterative Projects: An Industrial Experience Report in China. In: Wang, Q., Pfahl, D., Raffo, D.M. (eds.) ICSP 2008. LNCS, vol. 5007, pp. 369–380. Springer, Heidelberg (2008)
21. Zhang, S., Wang, Y., et al.: Capability Assessment of Individual Software Development Processes Using Software Repositories and DEA. In: Wang, Q., Pfahl, D., Raffo, D.M. (eds.) ICSP 2008. LNCS, vol. 5007, pp. 147–159. Springer, Heidelberg (2008)
22. Wang, Q., Li, M.: Measuring and improving software process in China. In: 2005 International Symposium on Empirical Software Engineering (2005)
23. Yang, D., Wu, D., et al.: WikiWinWin: A Wiki Based System for Collaborative Requirements Negotiation. In: HICSS (2008)

24. Li, Y., Li, J., et al.: Requirement-Centric Traceability for Change Impact Analysis: A Case Study. In: Wang, Q., Pfahl, D., Raffo, D.M. (eds.) ICSP 2008. LNCS, vol. 5007, pp. 100–111. Springer, Heidelberg (2008)
25. Li, J., Hou, L., et al.: An Empirically-Based Process to Improve the Practice of Requirement Review. In: Wang, Q., Pfahl, D., Raffo, D.M. (eds.) ICSP 2008. LNCS, vol. 5007, pp. 135–146. Springer, Heidelberg (2008)
26. Liu, D., Wang, Q., Xiao, J., Li, J., Li, H.: RVSim: A Simulation Approach to Predict the Impact of Requirements Volatility on Software Project Plans. In: Wang, Q., Pfahl, D., Raffo, D.M. (eds.) ICSP 2008. LNCS, vol. 5007, pp. 307–315. Springer, Heidelberg (2008)
27. Dai, J., Xiao, J., Wang, Q., Li, M., Li, H.: Dynamically Optimize Process Execution Based on Process Agent. Accepted by 2008 International Conference on Software Engineering and Knowledge Engineering (SEKE 2008) (2008)
28. Li, J., Chen, Z., Wei, L., Xu, W.: Feather Selection via Least Squares Support Feature Machine. International Journal of Information Technology & Decision Making 6(4) (2007)
29. Du, J., Tan, T., He, M., et al.: Technical Report: A Process-Centric Approach to Assure Software Trustworthiness. ISCAS/iTechs Technical Report #106 (September 2008)
30. Wang, Q., Yang, Y.: Technical Report: A Process-Centric Methodology to Software Trustworthiness Assurance. ISCAS/iTechs Technical Report #105 (June 2008)

A System Dynamics Model That Simulates a Significant Late Life Cycle Manpower Increase Phenomenon

Douglas Buettner

The Aerospace Corporation, 2350 E. El Segundo, El Segundo, CA USA
Douglas.J.Buettner@aero.org

Abstract. Available software metrics data from an initially schedule and cost-driven satellite flight software project contains a late life cycle staff spike from a lack of initial rigor in inspection and unit testing. In order to study the effects on the number of staff from varying the degree of inspection and unit test rigor, Madachy's inspection-based system dynamics model was modified to add unit testing and an integration test feedback loop. This modified Madachy model generated a similar late life cycle manpower rate increase from a parametric lack of up-front rigor in these same defect removal techniques.

Keywords: system dynamics, inspections, peer reviews, unit testing, software defects, feedback loops, satellite software.

1 Introduction

We describe research performed at The Aerospace Corporation (Aerospace) to fundamentally understand inadequate inspection and unit test rigor issues uncovered on one of our mission critical flight software projects. A summary with examples of the case study research from the author's Ph.D. dissertation (reference [1]) showing the inadequacy in these defect discovery methods is provided. Further, the paper will include a high-level overview of a model derived from Madachy's inspection-based system dynamics model that simulates this phenomenon.

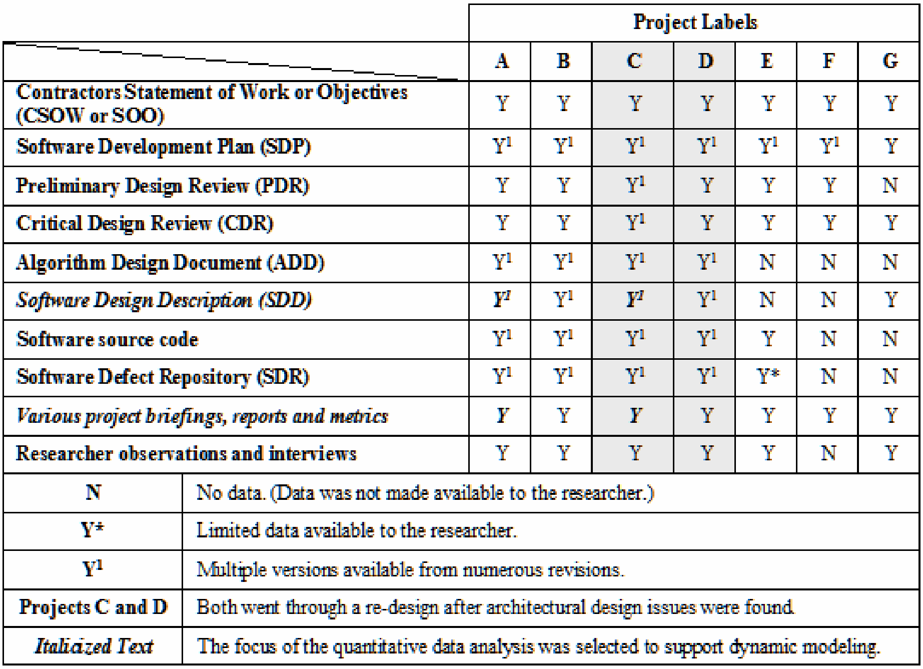
The paper will conclude with a discussion of threats to the validity of the model and will suggest avenues for further research for confirmation of the phenomenon and the proposed model.

2 Case Study Research

Aerospace has recently (in the past 5 years) started to maintain a software reliability research database containing information from flight and ground software projects. To date, the database contains seven flight software projects, five of which have provided text files with data extracted from their project's defect report (DR) database. Additional supporting information about the projects came from interviews of Aerospace employees involved in the technical oversight of the flight software projects found in

Q. Wang et al. (Eds.): ICSP 2009, LNCS 5543, pp. 402–410, 2009.

the database, supporting comments from contractor employees involved in the development and testing of those products, and the author's personal observations from his six years of support to the various projects and while on staff as a systems director responsible for overseeing all the flight software in a program office supported by Aerospace. Fig. 1 (below) provides a summary of the kinds of information available in the database, or where the researcher had personal observations, interviews with other Aerospace staff supporting these projects, or had performed informal interviews with contractor staff.



	Project Labels						
	A	B	C	D	E	F	G
Contractors Statement of Work or Objectives (CSOW or SOO)	Y	Y	Y	Y	Y	Y	Y
Software Development Plan (SDP)	Y^1	Y^1	Y^1	Y^1	Y^1	Y^1	Y
Preliminary Design Review (PDR)	Y	Y	Y^1	Y	Y	Y	N
Critical Design Review (CDR)	Y	Y	Y^1	Y	Y	Y	Y
Algorithm Design Document (ADD)	Y^1	Y^1	Y^1	Y^1	N	N	N
Software Design Description (SDD)	*Y*^{1}	Y^1	*Y*^{1}	Y^1	N	N	Y
Software source code	Y^1	Y^1	Y^1	Y^1	Y	N	N
Software Defect Repository (SDR)	Y^1	Y^1	Y^1	Y^1	Y*	N	N
Various project briefings, reports and metrics	*Y*	Y	*Y*	Y	Y	Y	Y
Researcher observations and interviews	Y	Y	Y	Y	Y	N	Y

N	No data. (Data was not made available to the researcher.)
Y*	Limited data available to the researcher.
Y^1	Multiple versions available from numerous revisions.
Projects C and D	Both went through a re-design after architectural design issues were found
Italicized Text	The focus of the quantitative data analysis was selected to support dynamic modeling.

Fig. 1. The figure contains a description of the data available from seven flight software projects in The Aerospace Corporation's Software Reliability Research Database as of the Summer of 2008

While, qualitative and quantitative case study results for the projects in Fig. 1 are reported in detail in reference [1], a summary of the qualitative and quantitative research results for the five projects where defect data are available is provided here:

- Project-A had no initial government oversight and was initially severely cost and schedule constrained forcing personnel, and test equipment cutbacks. Developers reverse engineered the structural design from the code, did not maintain algorithm documents, and cut quality corners under schedule pressure by discontinuing peer reviews and not doing 100% unit testing. The software qualification test team (a team of independent testers charged with verifying the software's requirements) spent a significant amount of time attempting to just get the software to work in the hardware in the loop test facility. The contractor's first attempt at qualifying the software in front of government technical witnesses failed this maturity gate. Following which, significant unit test thoroughness issues were subsequently uncovered from a review of the contractor's attempt at re-unit testing functionality. A third attempt at re-unit testing all of the software consisted of significant government oversight and team retraining. This third attempt found an additional 141 defects, 18 of which required procedural or database changes to avoid the newly identified issues in operation. The second more thorough attempt at qualifying the software identified numerous functionality disconnect issues with the requirements, design, and algorithm documents and uncovered additional issues that needed to be operationally worked around. Project-A's ability to perform in parallel development and test was limited by the cost decision to build only a single hardware in the loop test system. An effort to build a software in the loop simulator had been discontinued, but then reconstituted after the software test issues were uncovered. The software quality assurance representative in this case was embedded within the team.

- Project-B transferred lessons learned from project-A, and designed their code using the software development plan required Unified Modeling Language (UML), with significant contractor and customer visibility. Software peer reviews were superior to those in project-A as the government funded independent technical support to attend these reviews. The contractor used third party unit testing and performed thorough integration testing to vastly improve the quality of the software for qualification and system testing. Project-B used two hardware in the loop test systems for development and testing. Ultimately few defects were found by the qualification test team members.
- Project-C had no initial government oversight and eventually suffered from the early discovery of fundamental design flaws, following which significant government oversight and a corporate commitment to building a quality product led to a thorough design, and a rigorous defect prevention process. Government technical oversight was embedded with the contractor's software developers effectively keeping them from doing code while the project was still in the re-design phase. The project used two hardware in the loop systems. However, there were cost constraints that kept them from building a comprehensive test system that could proactively identify all of the errors. As a consequence, there were issues that were found later on in system testing on the flight hardware. The team effectively used a commercial unit test tool to achieve a high level of branch and path test coverage.

- Project-D began as a project with no government oversight and did not use requirements engineering or a design process that included UML use cases or sequence diagrams. The product had to be re-unit tested and reached coverage close to 100%. The lack of an up front design process included no design for functionality like the fault detection and management system and eventually led to the discovery of four significant design flaws and numerous minor ones during system testing and subsequent analysis. The result was the need to completely rewrite significant portions of the flight software. The rewriting effort did not follow a design then code approach. Instead they used the approach of modifying software from another program and then reverse engineering the design from the resulting product. The second effort utilized a software development team with significant years of experience writing space flight software.
- Project-E used a significant amount of re-use software and an incremental development approach that did not use thorough UML design products. One example of corner cutting in the limited amount of defect database available to the researcher indicated that the system tester had found a defect which proved to him that the software developer had obviously not done <u>any</u> unit testing at all.

Additionally, reported management strategies (for schedules that were unattainable) included; recommending less effort colleague reviews for mission critical software instead of the more rigorous inspections, providing the government over-parallelized schedules with no detail and overloaded personnel (tasking beyond 200% or more in some cases), schedules that lacked detail and assigned resources, dismissing[1] the results of widely used parametric modeling tools that indicated the contractor's schedules were to short, and provided schedule completions that used coding rates that exceeded anything the contractor had ever executed at in the past, or used overly optimistic integration and qualification test efforts. Finally, they suggested that software testing could be completed with little effort, while management simultaneously instructed testers to reduce integration testing (by a factor of 10 in one case) because it did not fit into the schedule.

Cumulative defect distributions from four of the case study projects in the Aerospace research database are provided below in Fig. 2.

[1] "Dismissing" was the contractor's management claim that the parametric modeling tools did not apply in "this case", among other claims that these tools were insufficiently capturing "their software development situation", or that these tools were just plain wrong. Nevertheless, in all of the situations in this case study research, the parametric modeling tools provided estimates that were more pessimistic than the optimistic schedules used by the contractor's management; ultimately, the parametric modeling tools provided schedules that were more accurate. Furthermore, these tools slightly underestimated the actual schedules by a few months as we usually tried to give the contractors the benefit of the doubt when setting parameters.

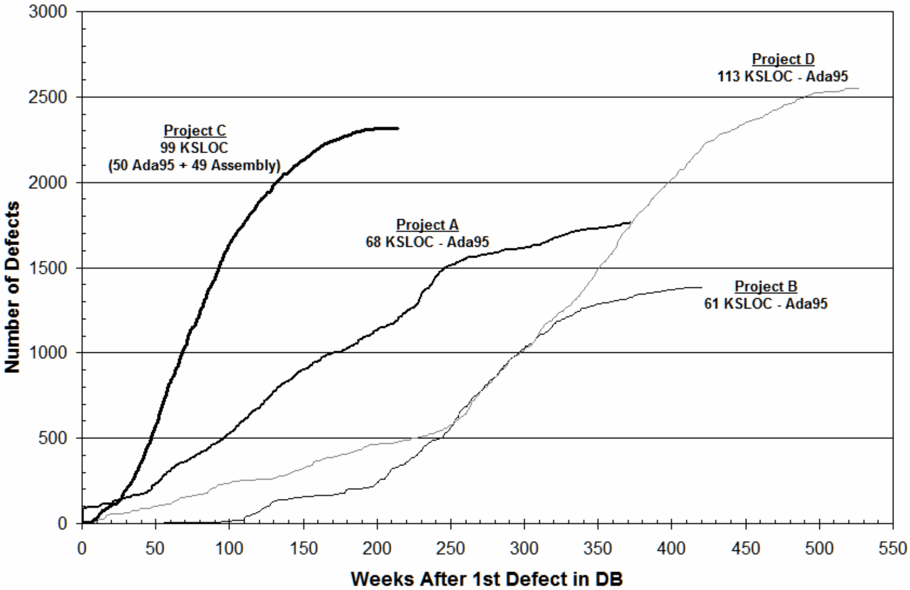


Fig. 2. Cumulative plots of all the reported defects covering software documentation, test products and flight code for the case study data that were included in ASCII text data dumps from project defect databases found in Aerospace's database

Buettner's research goal (see reference [2]) was to understand the fundamental reasons driving defect removal issues uncovered on project-A. He decided to compare project-A with project-C, which had better defect removal characteristics. Recall that even though project-C had initially begun as a troubled project, a corporate commitment to fix the situation in addition to significant government oversight provided a significantly better schedule and superior defect discovery dynamics than was seen on project-A. Thus, project-C provided a counter example for modeling.

Peer review findings (co-plotted in Fig. 3) from projects-A and C show a significant disparity between the total numbers of findings identified. Review of the archived information from project-A also indicated that at some point peer reviews had been dropped all together due to schedule pressure, which was substantiated from interviews of project and Aerospace personnel. Not plotted on Fig. 3 for project-A are the peer review findings from the second staff spike (seen in Fig. 4).

Staffing data for project's A and C (co-plotted in Fig. 4 above) were extracted from available monthly status reports and organization charts. Project management task data identifying exactly what the development team's tasks were, would have been preferred however this information was not available in the team's software development folders (SDFs). In addition, data gaps are from not keeping up the data in the project's SDFs. Furthermore, these two specific projects appeared to not use project management tools to manage and coordinate the software development tasks. (The archive contains an exact snapshot for all of the appropriate files in the project's completed SDF, where evidence of managements' use of these tools should have been found. It is possible, that the detailed software development project plans were maintained by somebody outside of the immediate software development team.)

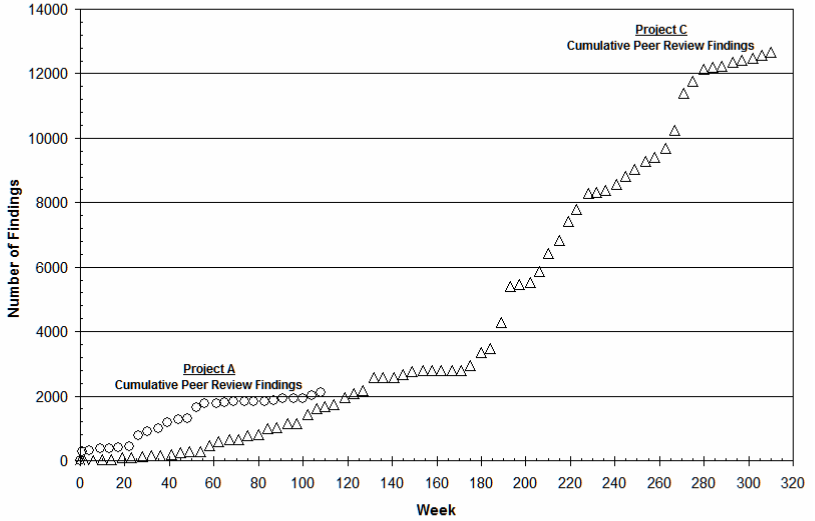


Fig. 3. Co-plotted peer review data from projects A and C

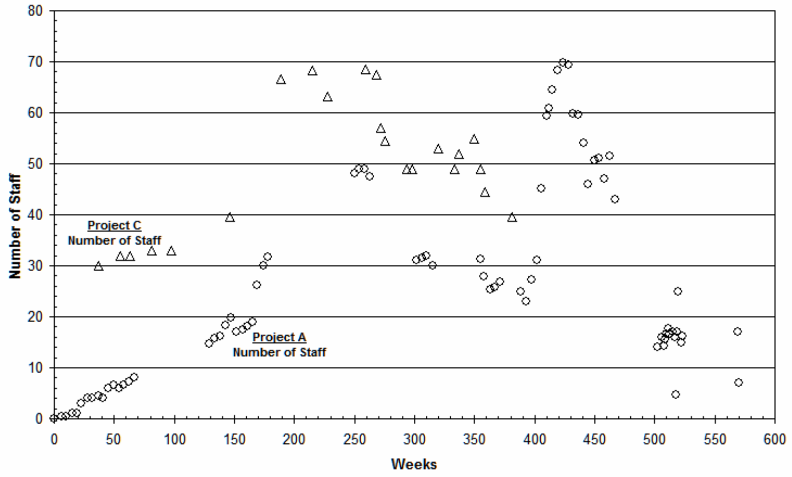


Fig. 4. Co-plotted available project staffing data from projects A and C

Hence, project-A provides an example where both inspections and unit testing were initially cut to some degree during the development, while project-C (despite the initial design issues) provides us with a counter example of the application of a more rigorous approach towards inspections and unit testing.

3 System Dynamics Modeling

Our modeling situation requires a temporal dependency on the effort that is expended on inspections and unit testing, allowing this effort to be ‘pushed down stream’ when it is not performed in the appropriate software development phase. System dynamics modeling provides a temporal modeling capability, and furthermore, Commercial-Off-The-Shelf (COTS) tools are readily available with easy to learn integrated development environments. In addition, system dynamicists are using these tools to provide an ever increasing wealth of models of software process dynamics, and thus is a testament to the popularity of the method.

An existing model, Madachy's inspection-based model (reference [3]), provides one that has been thoroughly tested and already includes a task and error sequence that is based on the COCOMO (reference [4]) calibrated effort expended during the software development process. Our goal is to create a modified model to determine if we can simulate the observed late life cycle increase in staff. Furthermore, the form of the inspection model in Madachy's work allows for the straightforward addition of a unit-testing sub-model. Moreover, an integration test feedback loop implementation, which is a representation of the methodology described in the software development plans for these two projects, was incorporated with a modest amount of effort by the author.

The interested reader is directed to reference [5] for the modified Madachy model's details, while the model's extent makes it difficult to properly document here, Fig. 5 is included below to provide an example of one of the sub-models.

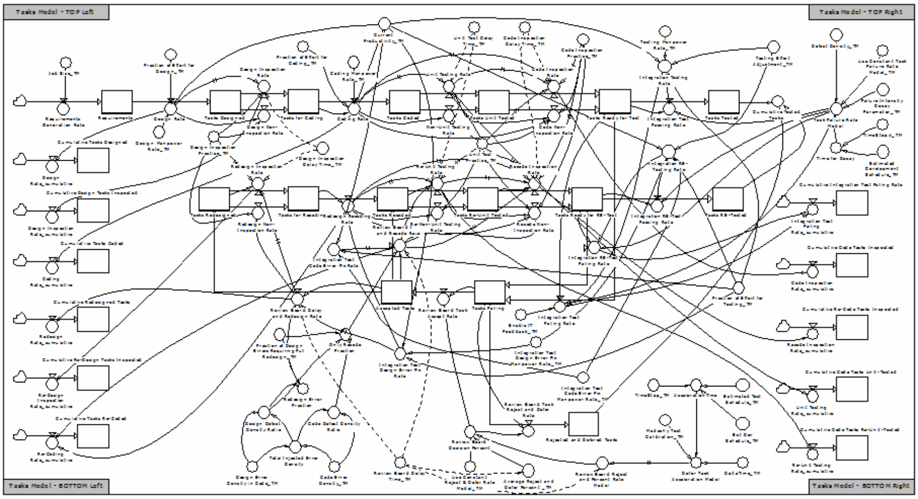

Tasks Model - TOP Left
Tasks Model - TOP Right
Tasks Model - BOTTOM Left
Tasks Model - BOTTOM Right
Job Size_TM
Requirements Generation Rate
Requirements
Fraction of Effort for Design_TM
Design Rate
Design Manpower Rate_TM
Tasks Designed
Design Inspection Rate
Design Non-Inspection Rate
Design Inspection Practice_TM
Design Inspection Delay Time_TM
Tasks for Coding
Fraction of Effort for Coding_TM
Coding Manpower Rate_TM
Coding Rate
Current Productivity_TM
Tasks Coded
Unit Testing Rate
Non-Unit Testing Rate
Unit Test Delay Time_TM
Code Inspection Delay Time_TM
Code Inspection Practice_TM
Code Inspection Rate
Code Non-Inspection Rate
Tasks Unit Tested
Unit Test Practice_TM
Tasks Ready for Test
Testing Manpower Rate_TM
Integration Testing Rate
Integration Test Passing Rate
Tasks Tested
Testing Effort Adjustment_TM
Cumulative Tested Tasks
Defect Density_TM
Use Constant Task Failure Rate Model_TM
Task Failure Rate Model
Failure Intensity Decay Parameter_TM
TimeStep 4_TM
Time for Decay
Estimated Development Schedule_TM
Cumulative Tasks Designed
Design Rate_cumulative
Cumulative Design Tasks Inspected
Design Inspection Rate_cumulative
Cumulative Tasks Coded
Coding Rate_cumulative
Cumulative Redesigned Tasks
Redesign Rate_cumulative
Cumulative ReDesign Tasks Inspected
ReDesign Inspection Rate_cumulative
Cumulative Tasks ReCoded
ReCoding Rate_cumulative
Redesign Inspection Rate
Tasks Redesigned
Redesign Non-Inspection Rate
Tasks for Recoding
Redesign Recoding Rate
Tasks Recoded
Review Board Recode Rate
ReUnit Testing Rate
ReNon-Unit Testing Rate
Recode Inspection Rate
Recode Non-Inspection Rate
Tasks ReUnit Tested
Tasks Ready for RE-Test
Integration RE-Testing Rate
Integration RE-Test Passing Rate
Tasks RE-Tested
Integration RE-Test Failing Rate
Integration Test Code Error Fix Rate
Accepted Tasks
Review Board Task Accept Rate
Tasks Failing
Integration Test Failing Rate
Enable IT Feedback_TM
Review Board Delay and Redesign Rate
Fraction of Design Errors Requiring Full Redesign_TM
Only Recode Fraction
Integration Test Design Error Fix Rate
Integration Test Design Error Fix Manpower Rate_TM
Integration Test Code Error Fix Manpower Rate_TM
Redesign Error Fraction
Design Defect Density Ratio
Code Defect Density Ratio
Total Injected Error Density
Design Error Density in Code_TM
Code Error Density_TM
Review Board Delay Time_TM
Use Constant Reject & Defer Rate Model_TM
Average Reject and Defer Percent_TM
Review Board Decision Percent
Review Board Task Reject and Defer Rate
Rejected and Deferred Tasks
Review Board Reject and Percent Rate Model
Fraction of Effort for Testing_TM
TimeStep_TM
Acceleration Time
Estimated Test Schedule_TM
Madachy Test Calibration_TM
Std Dev Schedule_TM
Defer Task Acceleration Model
DeltaTime_TM
Cumulative Integration Test Failing Rate
Integration Test Failing Rate_cumulative
Cumulative Code Tasks Inspected
Code Inspection Rate_cumulative
Cumulative ReCode Tasks Inspected
Recode Inspection Rate_cumulative
Cumulative Code Tasks Unit-Tested
Unit Testing Rate_cumulative
Cumulative Code Tasks ReUnit-Tested
ReUnit Testing Rate_cumulative

Fig. 5. A full view of the modified version of Madachy's Task sub-model is shown. This modified model contains additions for unit testing and the integration testing task feedback loop to account for failing tasks.

4 Modeling Results

We show in Fig. 6 (below) the differences between Madachy's baseline model with and without unit testing for a 100 KSLOC embedded mode project using his method to accentuate the difference in the manpower rate (or staffing) results with only this addition.

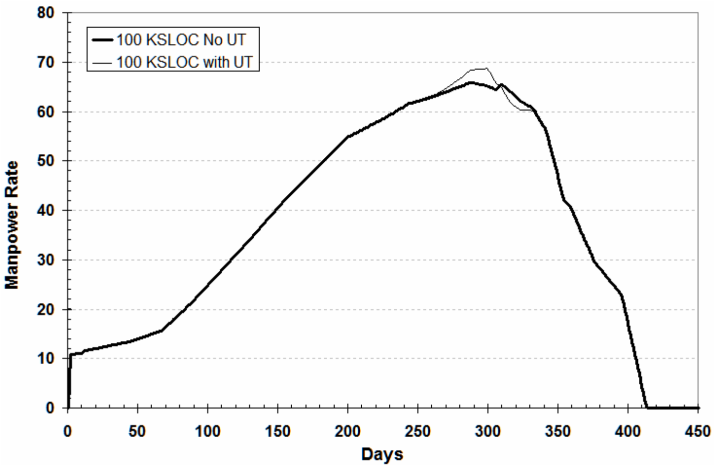


Fig. 6. The bold black line is Madachy's baseline model without unit testing, while the light black line is Madachy's model with unit testing for a 100 KSLOC embedded mode project

The addition of unit testing increases the staff's effort prior to staff day 300, while decreasing slightly the effort after staff day 310. The minimal baseline integration effort at staff day 310 is removed, and both lines merge with minimal differences following staff day 330.

Results from adding in the integration test feedback loop with various parametric levels of inspection and unit testing for a 64 KSLOC embedded mode project are provided below in Fig. 7. These results simulate the existence of a late life cycle effort increase from lax early life cycle inspection and unit testing processes.

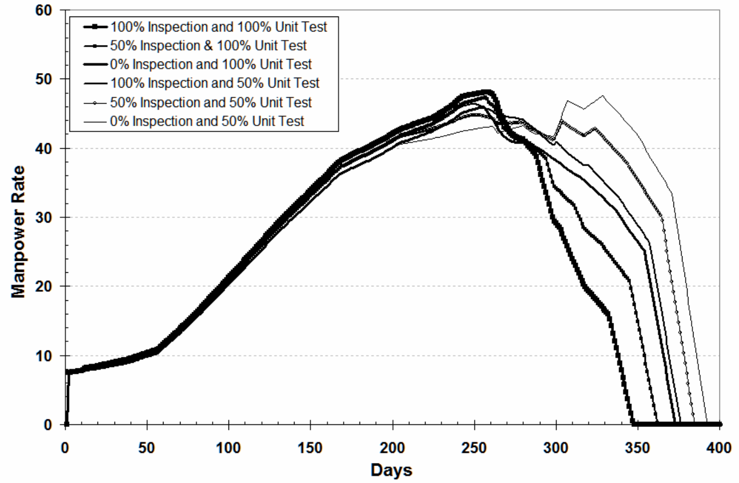


Fig. 7. Varying levels of inspection and unit testing with an integration test feedback loop added to Madachy's model for a 64 KSLOC embedded mode project demonstrate a shift in development effort to a late in the life cycle find and fix effort from the discovery of defects

The models validity of course needs to be verified and validated against similar examples. In our case, the customer chose to expend additional funds to redo the life cycle quality steps usually completed early in the project, thus allowing the manpower increase. Projects with insufficient funds or un-informed oversight likely would not have redone the important unit tests, inspections of those unit tests and the requirements qualification tests which would have resulted in a higher risk for the late discovery of defects on orbit.

5 Conclusions

An implementation of a late life cycle error discovery feedback loop using a modified version of Madachy's inspection-based system dynamics model demonstrates a shift in effort resulting from parametrically reduced early life cycle inspection and unit testing defect discovery and removal processes. These results simulate a similar late life cycle staff increase that was found in one of the space flight software projects overseen by Aerospace Corporation personnel.

References

1. Buettner, D.J.: Designing An Optimal Software Intensive System Acquisition: A Game Theoretic Approach, Ph.D. Dissertation, Astronautics and Space Technology Division, Viterbi School of Engineering, USC, 60–100 (September 2008)
2. Buettner, D.J.: 4–5
3. Madachy, R.J.: A Software Project Dynamics Model For Process Cost, Schedule And Risk Assessment, Ph.D. Dissertation, Department of Industrial and Systems Engineering, USC, 53–58 (December 1994)
4. Boehm, B.W.: Software Engineering Economics, pp. 35–39. Prentice-Hall, Inc., Englewood Cliffs (1981)
5. Buettner, D.J.: 106–125, 289–305

Author Index