

Jürgen Münch
Pekka Abrahamsson (Eds.)

LNCS 4589

Product-Focused Software Process Improvement

8th International Conference, PROFES 2007
Riga, Latvia, July 2007
Proceedings



Springer

Commenced Publication in 1973

Founding and Former Series Editors:

Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

David Hutchison

Lancaster University, UK

Takeo Kanade

Carnegie Mellon University, Pittsburgh, PA, USA

Josef Kittler

University of Surrey, Guildford, UK

Jon M. Kleinberg

Cornell University, Ithaca, NY, USA

Friedemann Mattern

ETH Zurich, Switzerland

John C. Mitchell

Stanford University, CA, USA

Moni Naor

Weizmann Institute of Science, Rehovot, Israel

Oscar Nierstrasz

University of Bern, Switzerland

C. Pandu Rangan

Indian Institute of Technology, Madras, India

Bernhard Steffen

University of Dortmund, Germany

Madhu Sudan

Massachusetts Institute of Technology, MA, USA

Demetri Terzopoulos

University of California, Los Angeles, CA, USA

Doug Tygar

University of California, Berkeley, CA, USA

Moshe Y. Vardi

Rice University, Houston, TX, USA

Gerhard Weikum

Max-Planck Institute of Computer Science, Saarbruecken, Germany

Jürgen Münch Pekka Abrahamsson (Eds.)

Product-Focused Software Process Improvement

8th International Conference, PROFES 2007
Riga, Latvia, July 2-4, 2007
Proceedings

 Springer

Volume Editors

Jürgen Münch

Fraunhofer Institute for Experimental Software Engineering

Fraunhofer-Platz 1, 67663 Kaiserslautern, Germany

E-mail: juergen.muench@iese.fraunhofer.de

Pekka Abrahamsson

VTT Electronics

Kaitovayla 1, 90570 Oulu, Finland

E-mail: pekka.abrahamsson@vtt.fi

Library of Congress Control Number: 2007929634

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

LNCS Sublibrary: SL 2 – Programming and Software Engineering

ISSN 0302-9743

ISBN-10 3-540-73459-7 Springer Berlin Heidelberg New York

ISBN-13 978-3-540-73459-8 Springer Berlin Heidelberg New York

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

Springer is a part of Springer Science+Business Media

springer.com

© Springer-Verlag Berlin Heidelberg 2007

Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper SPIN: 12086863 06/3180 5 4 3 2 1 0

Preface

The Eight International Conference on Product-Focused Software Process Improvement (PROFES 2007) brought together researchers and industrial practitioners to report new research results and exchange experiences and findings in the area of process and product improvement. The focus of the conference is on understanding, learning, evaluating, and improving the relationships between process improvement activities (such as the deployment of innovative defect detection processes) and their effects in products (such as improved product reliability and safety). Consequently, major topics of the conference include the evaluation of existing software process improvement (SPI) approaches in different contexts, the presentation of new or modified SPI approaches, and the relation between SPI and new development techniques or emerging application domains.

This year's conference theme focused on global software development. More and more products are being developed in distributed, global development environments with many customer-supplier relations in the value chain. Outsourcing, off-shoring, near-shoring, and even in-sourcing aggravate this trend further. Supporting such distributed development requires well-understood and accurately implemented development process interfaces, process synchronization, and an efficient process evolution mechanisms. Overcoming cultural barriers and implementing efficient communication channels are some of the key challenges. It is clear that process improvement approaches also need to consider these new development contexts.

A second key focus of PROFES 2007 was on agile software development. Market dynamics require organizations to adapt to changes of the development environment and to enforce innovations better and faster. This often results in process changes that impose risk challenges for SPI approaches. Advanced SPI is required to support the assessment of the impact of process changes such as the introduction of agile methods. Due to the fact that software development processes are human-based and depend heavily on the development context, process changes and their resulting effects should be considered carefully. We consider the development context to include at least the domain-specific characteristics, the workforce capabilities, and the level of work distribution.

The technical program was selected by a committee of leading experts in software process modeling and SPI research. This year, 56 papers from 21 nations were submitted, with each paper receiving at least three reviews. The Program Committee met in Riga for one full day in February 2007. The Program Committee finally selected 30 technical full papers. The topics indicate that SPI remains a vibrant research discipline of high interest for industry. Emerging technologies and application domains, a paradigm shift to global software and system engineering in many domains, and the need for better decision support for SPI are reflected in these papers. The technical program consisted of the tracks global software development, software process improvement, software process modeling and evolution, industrial experiences, agile software development, software measurement, simulation and decision support, and processes and methods. We were proud to have four distinguished keynote speakers,

Carol Dekkers, Dieter Rombach, Jari Still, Guntis Urtāns, as well as interesting tutorials and a collocated workshop.

We are thankful for the opportunity to serve as Program Co-chairs for this conference. The Program Committee members and reviewers provided excellent support in reviewing the papers. We are also grateful to the authors, presenters, and session chairs for their time and effort that made PROFES 2007 a success. The General Chair, Pasi Kuvaja, and the Steering Committee provided excellent guidance. We wish to thank the University of Latvia, the Fraunhofer Institute for Experimental Software Engineering (IESE), VTT, the University of Oulu and all the other sponsors and supporters for their contributions and making the event possible. We would especially like to thank the Organizing Chairs Darja Šmite and Juris Borzovs and the Local Organizing Committee for their highly engaged organization of the conference in Riga. Last but not least, many thanks to Timo Klein at Fraunhofer IESE for copyediting this volume.

April 2007

Jürgen Münch
Pekka Abrahamsson

Conference Organization

General Chair

Pasi Kuvaja, University of Oulu (Finland)

Program Co-chairs

Jürgen Münch, Fraunhofer IESE (Germany)

Pekka Abrahamsson, VTT Technical Research Centre (Finland)

Organizing Co-chairs

Juris Borzovs, University of Latvia (Latvia)

Darja Šmite, University of Latvia (Latvia)

Local Organizing Committee

Dainis Dosbergs, PR-Latvia (Latvia)

Krišs Rauhvargers, University of Latvia (Latvia)

Program Committee

Pekka Abrahamsson, VTT Electronics, Finland

Bente Anta, Simula Research Laboratory, Norway

Andreas Birk, Software Process Management, Germany

Mark van den Brand, HvA & CWI, The Netherlands

Gerardo Canfora, University of Sannio at Benevento, Italy

Reidar Conradi, NTNU, Norway

Torgeir Dingsøy, Sintef, Norway

Tore Dybå, SINTEF, Norway

Jens Heidrich, Fraunhofer IESE, Germany

Martin Höst, Lund University, Sweden

Frank Houdek, DaimlerChrysler, Germany

Tua Huomo, VTT Electronics, Finland

Hajimu Iida, Nara Institute of Science and Technology, Japan

Katsuro Inoue, Osaka University, Japan

Yasushi Ishigai, IPA and Mitsubishi Research Institute, Japan

Janne Järvinen, Solid Information Technology, Finland

Erik Johansson, Q-Labs, Sweden

Philip Johnson, University of Hawaii, USA

Natalia Juristo, Universidad Politecnica de Madrid, Spain
Tuomo Kähkönen, Nokia, Finland
Haruhiko Kaiya, Shinshu University, Japan
Kari Käsälä, Nokia Research Center, Finland
Masafumi Katahira, JAXA, Japan
Pasi Kuvaja, University of Oulu, Finland
Makoto Matsushita, Osaka University, Japan
Kenichi Matsumoto, NAIST, Japan
Maurizio Morisio, University of Turin, Italy
Mark Müller, Bosch, Germany
Jürgen Münch, Fraunhofer IESE, Germany
Paolo Nesi, University of Florence, Italy
Risto Nevalainen, STTF, Finland
Mahmood Niazi, Keele University, UK
Hideto Ogasawara, Toshiba, Japan
Dietmar Pfahl, University of Calgary, Canada
Teade Punter, LAQUSO, The Netherlands
Karl Reed, La Tobe University, Australia
Günther Ruhe, University of Calgary, Canada
Ioana Rus, Honeywell Aerospace, USA
Outi Salo, VTT Electronics, Finland
Kurt Schneider, University of Hannover, Germany
Carolyn Seaman, UMBC, Baltimore, USA
Michael Stupperich, DaimlerChrysler, Germany
Markku Tukiainen, University of Joensuu, Finland
Rini van Solingen, LogicaCMG, The Netherlands
Matias Vierimaa, VTT Electronics, Finland
Hironori Washizaki, National Institute of Informatics, Japan
Claes Wohlin, Blekinge Institute of Technology, Sweden
Bernard Wong, University of Technology Sydney, Australia

External Reviewers

Nicola Boffoli, Software Engineering Research Laboratory, Italy
Kyohei Fushida, Software Design Laboratory, Japan
Ahmed Al-Emran, University of Calgary, Canada
Maria Alaranta, Turku School of Economics, Finland
Martin Solari, ORT University, Uruguay

Table of Contents

Keynote Addresses

Software Development and Globalization (Abstract)	1
<i>H. Dieter Rombach</i>	
Software Development Globalization from the Baltic Perspective (Abstract)	2
<i>Guntis Urtāns</i>	
Experiences in Applying Agile Software Development in F-Secure (Abstract)	3
<i>Jari Still</i>	
People Side of IT Globalization (Abstract)	4
<i>Carol Dekkers</i>	

Global Software Development

An Industrial Survey of Software Outsourcing in China	5
<i>Jianqiang Ma, Jingyue Li, Weibing Chen, Reidar Conradi, Junzhong Ji, and Chunnian Liu</i>	
Understanding Lacking Trust in Global Software Teams: A Multi-Case Study	20
<i>Nils Brede Moe and Darja Šmite</i>	
Utilization of a Set of Software Engineering Roles for a Multinational Organization	35
<i>Claude Y. Laporte, Mikel Doucet, Pierre Bourque, and Youssef Belkébir</i>	

Software Process Improvement

Software Verification Process Improvement Proposal Using Six Sigma . . .	51
<i>Tihana Galinac and Željka Car</i>	
Software Development Improvement with SFIM	65
<i>René Krikhaar and Martin Mermans</i>	
SPI-KM - Lessons Learned from Applying a Software Process Improvement Strategy Supported by Knowledge Management	81
<i>Gleison Santos, Mariano Montoni, Sávio Figueiredo, and Ana Regina Rocha</i>	

Organisational Readiness and Software Process Improvement	96
<i>Mahmood Niazi, David Wilson, and Didar Zowghi</i>	
Software Process Improvement Through Teamwork Management	108
<i>Esperança Amengual and Antònia Mas</i>	
De-motivators of Software Process Improvement: An Analysis of Vietnamese Practitioners' Views	118
<i>Mahmood Niazi and Muhammad Ali Babar</i>	

Software Process Modeling and Evolution

Defining Software Processes Through Process Workshops: A Multicase Study	132
<i>Finn Olav Bjørnson, Tor Stålhane, Nils Brede Moe, and Torgeir Dingsøy</i>	
Improving an Industrial Reference Process by Information Flow Analysis: A Case Study	147
<i>Kai Stapel, Kurt Schneider, Daniel Lübke, and Thomas Flohr</i>	
Connecting the Rationale for Changes to the Evolution of a Process	160
<i>Alexis Ocampo and Martin Soto</i>	

Industrial Experiences

Use of Non-IT Testers in Software Development	175
<i>Vineta Arnicane</i>	
Requirements Management Practices as Patterns for Distributed Product Management	188
<i>Antti Välimäki and Jukka Käriäinen</i>	
SPI Consulting in a Level 1 Company: An Experience Report	201
<i>Tomas Schweigert and Michael Philipp</i>	

Agile Software Development

On the Effects of Pair Programming on Thoroughness and Fault-Finding Effectiveness of Unit Tests	207
<i>Lech Madeyski</i>	
An Agile Toolkit to Support Agent-Oriented and Service-Oriented Computing Mechanisms	222
<i>Asif Qumer and Brian Henderson-Sellers</i>	
Achieving Success in Supply Chain Management Software by Agility . . .	237
<i>Deepti Mishra and Alok Mishra</i>	

Software Measurement

Software Measurement Programs in SMEs – Defining Software Indicators: A Methodological Framework	247
<i>María Díaz-Ley, Félix García, and Mario Piattini</i>	
Smart Technologies in Software Life Cycle	262
<i>Zane Bičevska and Jānis Bičevskis</i>	
Convertibility Between IFPUG and COSMIC Functional Size Measurements	273
<i>Juan Jose Cuadrado-Gallego, Daniel Rodríguez, Fernando Machado, and Alain Abran</i>	
A Framework for Measuring and Evaluating Program Source Code Quality	284
<i>Hironori Washizaki, Rieko Namiki, Tomoyuki Fukuoka, Yoko Harada, and Hiroyuki Watanabe</i>	
Software Fault Prediction with Object-Oriented Metrics Based Artificial Immune Recognition System	300
<i>Cagatay Catal and Banu Diri</i>	

Simulation and Decision Support

Operational Planning, Re-planning and Risk Analysis for Software Releases	315
<i>Ahmed Al-Emran and Dietmar Pfahl</i>	
Project Cost Overrun Simulation in Software Product Line Development	330
<i>Makoto Nonaka, Liming Zhu, Muhammad Ali Babar, and Mark Staples</i>	
E-Service Architecture Selection Based on Multi-criteria Optimization	345
<i>Edzus Zeiris and Maris Ziema</i>	

Processes and Methods

A Component-Based Process for Developing Automotive ECU Software	358
<i>Jin Sun Her, Si Won Choi, Du Wan Cheun, Jeong Seop Bae, and Soo Dong Kim</i>	
A Systematic Approach to Service-Oriented Analysis and Design	374
<i>Soo Ho Chang and Soo Dong Kim</i>	

Improving the Problem Management Process from Knowledge Management Perspective 389
Marko Jäntti, Aki Miettinen, Niko Pyllkkänen, and Tommi Kainulainen

Workshop

Experience on Applying Quantitative and Qualitative Empiricism to Software Engineering (Workshop Description) 402
Marcus Ciolkowski and Andreas Jedlitschka

Tutorials

Using Metrics to Improve Software Testing (Tutorial Description) 405
Alfred Sorkowitz

Increase ICT Project Success with Concrete Scope Management (Tutorial Description) 407
Carol Dekkers and Pekka Forselius

Agile Software Development: Theoretical and Practical Outlook (Tutorial Description) 410
Pekka Abrahamsson and Jari Still

Author Index 413

Software Development and Globalization

H. Dieter Rombach

Chairman, ICT Group, Fraunhofer Gesellschaft e.V.,
Executive Director, Fraunhofer IESE, Kaiserslautern,
Software Engineering Chair, CS Dept., University of Kaiserslautern

Developing software across borders has become an emerging area of software engineering. It is one of the important competitive advantages in today's industry. However, the increased globalization of software development creates many challenges brought by distribution of software life cycle activities among teams separated by various boundaries, such as contextual, organizational, cultural, temporal, geographical, and political.

Software Development Globalization from the Baltic Perspective

Guntis Urtāns

President of SWH Technology

Future predictions say that together with its neighbors Estonia and Lithuania, Latvia will be a major outsourcing center for Northern / continental Europe. The Baltic region is known for its well-educated and multinational workforce, one of the most efficient tax systems in Europe, a liberal economy, great affinity with Nordics, and identical legislation. So, where have we been 15 years ago and where are we now?

Experiences in Applying Agile Software Development in F-Secure

Jari Still

F-Secure Oy's Oulu Office site manager

To develop security software is clearly one of the most challenging software development areas. The challenges are both technical and business based. From the business point of view, the security market is mature and highly competitive, although market needs can change even daily and there is no room for mistakes. Technically, the challenge is the ability to find and catch the threats as soon as they arise.

F-Secure has been one of the leading companies in applying agile software development methods, even though F-Secure works with most challenging requirements like security criticality, short time-to-market, frequently changing requirements, and high quality. At the moment, F-Secure has a software product life cycle process, which is built on agile methods. The difference to the earlier process, which was based on "mature" models, is significant. This keynote speech will address those differences and describe the experiences F-Secure has made with agile methods.

People Side of IT Globalization

Carol Dekkers

PMP, CMC, P.Eng.

President, Quality Plus Technologies, Inc.

IT today consists of myriad combinations of outsourcing, insourcing, offshoring, global development teams, and project teams scattered across multiple time zones. Technology abounds, integration and on-time delivery become mission critical, and process improvement demands increase. But... what's the impact of all this advancement on the people involved? As the technical demands for faster, better, and cheaper software increase, so, too, do the needs for effective communication and cultural intelligence. This presentation examines the people side of IT globalization and provides insights and recommendations for succeeding with project teams in a truly global world.

An Industrial Survey of Software Outsourcing in China

Jianqiang Ma¹, Jingyue Li², Weibing Chen¹, Reidar Conradi², Junzhong Ji¹,
and Chunnian Liu¹

¹ Beijing Municipal Key Laboratory of Multimedia and Intelligent Software Technology,
College of Computer Science and Technology,
Beijing University of Technology, Beijing 100022, China
{jianqiang.ma, weibingchen}@gmail.com

² Department of Computer and Information Science,
Norwegian University of Science and Technology (NTNU)
NO-7491 Trondheim, Norway
{jingyue, conradi}@idi.ntnu.no

Abstract. Most studies of software outsourcing focus on how to manage the outsourcing from the perspective of the outsourcer, i.e., a company issuing a subcontract. There are few studies of outsourcing presented from the viewpoint of the supplier, i.e., a company receiving a subcontract. Since more and more Chinese software companies are getting software outsourcing subcontracts from all over the world, it is important to investigate how software outsourcing projects are actually performed in China, and to identify possible enhancements. Our study has collected data by a questionnaire-based survey from 53 finished projects in 41 Chinese software suppliers. The results show that: 1) Differences in natural languages may not be the barrier of Chinese software suppliers. 2) Email is most used to discuss development related issues, while face-to-face meetings are mainly used to discuss management and requirements issues. 3) The main reasons for overtime work at the Chinese suppliers are design or requirements changes initiated by the outsourcers and the suppliers' initial underestimation of the effort.

Keywords: Software Outsourcing, Empirical Study.

1 Introduction

Software outsourcing is gaining more and more attentions. It can help software companies to save costs and to focus on their core businesses [1, 5]. Along with the China's policy of opening to the outside world, China is becoming one of the biggest software supplying countries, together with India, Ireland, Russia, and so on. Although the volume of Chinese software exports in 2005 is \$3590 million, which has grown from \$720 million in 2001, it is still only 1/6 of India's and is 0.5% of the total volume of Chinese exports [2, 15].

Comparing to in-house software development, there are several factors that may impact the effectiveness of software outsourcing, such as infrequent and ineffective

communications between decentralized teams [6], time zone differences [4], and cultural differences [7, 8, 13]. Many studies have been performed on these issues, but mainly from the outsourcers' perspective. Since China is emerging as a major player in software outsourcing, it is important to study and make guidelines on such issues, in order to help Chinese software suppliers to improve their businesses.

Our questionnaire-based survey has investigated seven software engineering issues in Chinese software suppliers. Due to page limitation, we present only three of them in this paper, namely differences in natural languages, effective communication, and overtime work. We have used membership lists from a national Chinese Software Organization (called CSO¹ in this paper) to achieve a representative subset of software companies. We have gathered information from 53 finished projects in 41 Chinese suppliers. The results show that differences in natural languages may not be the critical factor to affect the success of Chinese software suppliers. Email is the most common mean of communication in development related issues between the outsourcers and suppliers, while face-to-face meetings are mainly used to discuss management and requirements issues. Design or requirements changes initiated by the outsourcers and the suppliers' initial underestimation of the effort are the main reasons for suppliers' overtime work.

The remainder of this paper is structured as follows: Section 2 presents the related work and research questions. Section 3 describes the research design. Section 4 presents results and discussions. Section 5 contains a general discussion. Conclusion and ideas for future work are presented in Section 6.

2 Related Work and Research Questions

There are two main participators in software outsourcing: the outsourcer and the supplier. Software outsourcers are the organizations that give software development subcontracts to other organizations, called the suppliers. Most previous studies focus on managing the software outsourcing from the outsourcers' point of view, such as how to manage the cultural differences [7, 8, 13], how to evaluate and select capable suppliers [10], and how to manage contractual related issues [3]. However, only few studies [11, 12] focus on facilitating the efficiency of software outsourcing from the suppliers' perspective, such as how to communicate and cooperate with outsourcers.

Since China has one of the largest software supplier industries in the world, it is opportune to investigate how Chinese software suppliers can improve their practices to communicate and cooperate with software outsourcers all over the world. The possible lack of English skill can pose constraints for Chinese software suppliers in communicating with outsourcers from Western countries [7]. As it is much harder to follow the plans in projects across organizational and geographic boundaries than projects within the same place and organization [16], the geographical differences between China and Western countries may also introduce additional project overruns [11, 14].

¹ The name of this organization was omitted for confidential reasons.

2.1 Differences in Natural Languages

In software outsourcing projects, the natural language being used in the documents is an important factor, because language differences may cause misunderstandings. For instance, word “more than” in Japanese means equal and more (\geq), but in Chinese it means more ($>$). The misunderstandings in requirements specifications or other development documents may cause project overruns.

A previous study [4] proposed that *“the language factor is one of the reasons for the success of software outsourcing in countries with strong English language capabilities, such as Philippines and Singapore”*. The study [9] concluded that the major Chinese market for software outsourcing is Japan, because of a similar culture. On the other hand, India’s major market is in US and UK because of a shared English language. So, the biggest barrier to Chinese software suppliers towards Western companies seems to be the English language. However, Japan and China also speak different natural languages, but the outsourcing projects between them seem to work well. So our first research question is:

RQ1: Are differences in natural languages the barrier to Chinese software suppliers?

2.2 Effective Communication

Compared with in-house software development, the communication shared in software outsourcing is affected by distribution of both space and time. The study [4] concluded that synchronous communication, such as telephone meetings or video conferences, can resolve misunderstandings and small problems before they become bigger problems. On the other hand, asynchronous communication like email often delays or complicates problem resolution. However, the results of another study [11] show that email is used much more than synchronous channels in a Latvian subcontractor company. To improve suppliers’ current practices, it is important to know which communication channel is mostly used between foreign outsourcers and Chinese suppliers. So our second research question is:

RQ2: What is the most common mean of communication between foreign outsourcers and Chinese suppliers?

2.3 Overtime Work

According to a survey [18], around 60% to 80% software development encountered effort and/or schedule overruns. The average overrun is about 30% to 40%. Schedule and budget risks are the highlighted risk factors in software outsourcing [12]. Comparing with in-house development, more issues, such as time zone differences and organizational differences, may also cause project overruns [11]. To keep the project schedule, overtime work is often used as a remedy. Most Chinese software suppliers get contracts from Japan, which has only one hour time difference with China. It seems that time zone difference is not the major reason of overtime work of

Chinese suppliers. We wonder there are other reasons that cause the project overrun. So, our third research question is:

RQ3: What are the main reasons for overtime work at Chinese software suppliers?

3 Research Design

To answer the research questions, we have used a questionnaire-based survey to collect data. First, a preliminary questionnaire with both open-ended and close-ended questions was designed by reading literatures. Second, a pre-study was performed to verify the quality of questions in the preliminary questionnaire, and to get answers on the open-ended questions. Based on the results of the pre-study, most open-ended questions in the preliminary questionnaire were redesigned into close-ended questions. In addition, the problematic questions in the preliminary questionnaire were revised. Then, the revised questionnaire was used to collect data in a main study.

3.1 The Preliminary Questionnaire

The preliminary questionnaire was designed to study seven issues of software outsourcing from the suppliers' perspective and had 10 sections. Sections 1 and 10 contain questions to collect background information of projects, companies and respondents. Questions in sections 2, 5 and 7 are related to the research questions in this paper. The remaining sections investigate four issues, which are related to technology management (e.g., how is source code integrated and updated) and business relationship management (e.g., how did the outsourcer and supplier get to know each other), are not reported in this paper.

3.2 The Pre-study to Verify and Refine the Preliminary Questionnaire

The pre-study included two steps, where individual interviews were followed by a workshop.

Step 1 – Individual interviews. We interviewed 5 project managers from 5 different software suppliers. All the interviewees had solid experiences in software outsourcing. Each interview was conducted by two authors of this paper. One was responsible for conducting the interview and the other recorded answers and asked for clarification if needed. The interviews were also taped for later use.

Step 2 – A workshop discussion. After the individual interviews, we revised most open-ended questions in the preliminary questionnaire into close-ended questions and made a second version of the preliminary questionnaire. We then organized a group discussion (a workshop) with more than 30 industrial experts to verify and comment the second version of the questionnaire. Based on inputs from the workshop, we revised the questionnaire into a final version. The final questionnaire includes 66 questions and takes about forty minutes to be filled in.

3.3 The Main Study to Collect Data

In the main study, the data was collected by cooperating with the CSO. As mentioned, we got 53 questionnaires from 41 companies. The sample selection and data collection process were as follows:

1. **Establish the target population.** We randomly selected 2,000 companies from a database of the CSO, which included about 6,000 Chinese software companies.
2. **Send invitation letters by email to obtain possible participants.** We sent invitation letters by email to the 2,000 selected companies. The invitation letter introduces the survey and specifies that the survey participant will be rewarded with either the final report of this survey or an annual membership of the CSO worth 500 Chinese Yuan. We got responses from 300 software outsourcing companies. These companies were later used as the original contact list.
3. **Send questionnaires by email to possible participants.** The unit of our study is defined as a finished software outsourcing project. We sent questionnaires (as word files) by email to the 300 companies and asked them to select one or more projects to answer the questionnaire. Since we cannot get the complete list of relevant projects in each company, the projects selection within the companies were decided by the respondents themselves, i.e., convenience sample.
4. **Collect filled-in questionnaires with follow up.** From the 300 companies, we first got 40 questionnaires back. To ensure the quality of the data, we excluded 10 questionnaires answered by respondents with less than three years working experiences. For the remaining 30 questionnaires, we contacted the respondents again by telephone to clarify possible misunderstandings and to fill-in the missing data. At the same time, we contacted the rest of 260 companies through telephone to persuade them to fill in the questionnaire. By doing this, we got 23 more questionnaires back.

4 Results and Discussions of Research Questions

In this section, we first present an overview of the collected questionnaires. After that, we show the results of each research questions followed by detailed discussions.

4.1 Overview of Collected Questionnaires

Participating companies. According to the number of employees, the participating companies include 7 small, 22 medium, 8 large, and 4 super large software companies, as shown in Fig. 1. Comparing with the profile of the number of employees in Chinese software industry [2], it shows that most of the participating companies in our survey are medium and large companies.

Human respondents. Most respondents are IT managers, project managers, or software architects. More than 70% of the respondents have at least 5 years experiences of software development. All of them have at least a Bachelor's degree in computer science or telecommunication.



Fig. 1. The distribution of companies

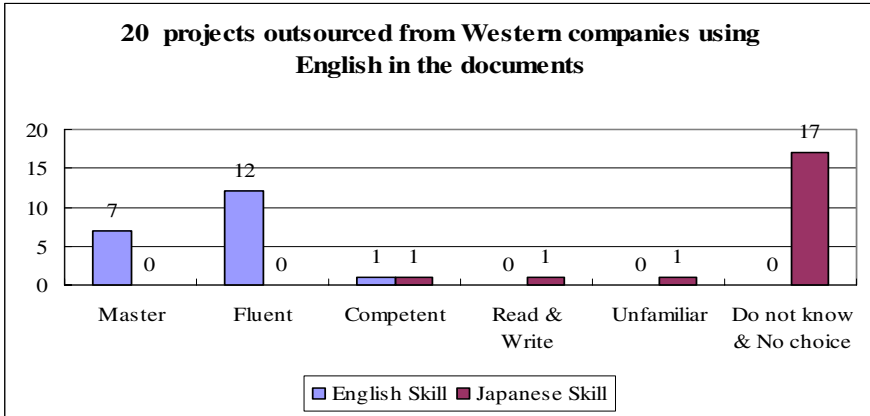
Participating projects. All of the 53 projects are finished software outsourcing projects. The mean duration of these projects (before the first delivery) is 161 days, and mean project effort is 1758 person-days. There are 21 projects outsourced from the US or European companies. One of these 21 projects used French in the document, and the other ones used English. There are 32 projects outsourced from Japanese companies. One of the 32 projects used Chinese in the documents, 8 of them used both English and Japanese, and the other 23 projects used Japanese.

4.2 Results and Discussions

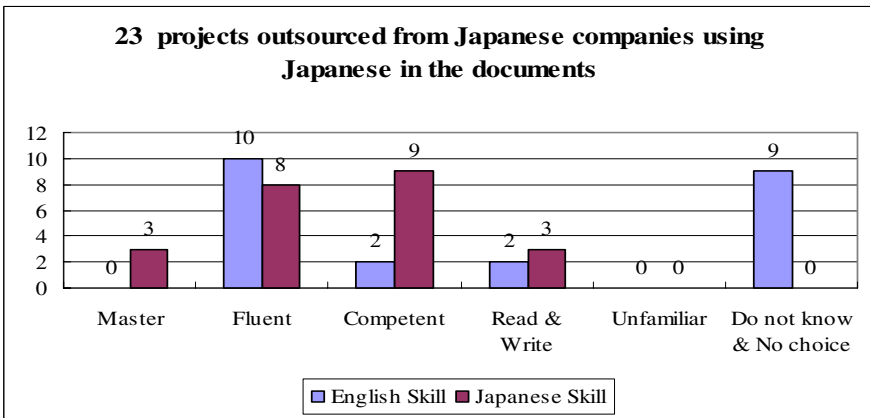
4.2.1 Investigating RQ1: Are Differences in Natural Languages the Barrier to Chinese Software Suppliers?

Results on RQ1. For this research question, we investigate how well the suppliers' employees understand the non-native natural languages used in the documents (e.g., business agreements, requirements specifications, and design specifications) provided by the outsourcers. The scales used to measure people's language skills are: master (near native), fluent, competent (can speak, read, and write, but not fluently), read&write only, unfamiliar, and do not know. The results of RQ1 are presented in Fig. 2 and Fig. 3.

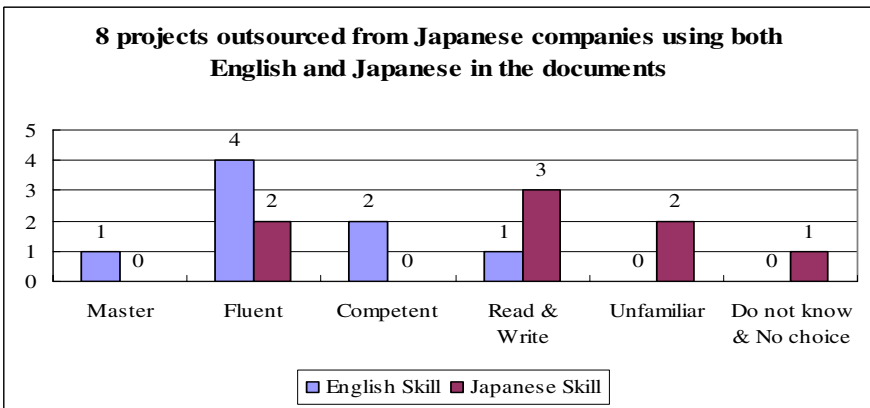
- Fig. 2(a) shows that respondents working on projects outsourced from American and European companies (with only English in documents) have good knowledge of English. However, their Japanese language skills are poor.
- Fig. 2(b) shows that respondents working on projects outsourced from Japanese companies with only Japanese in the documents are good at Japanese. Some of them have solid knowledge about English, while the others have poor English skills.
- Fig. 2(c) shows that respondents working on projects outsourced from Japanese companies with both English and Japanese in documents have much better skills in English than in Japanese.
- Fig. 3 shows that employees of the suppliers on average have better language skills in English than in Japanese.



(a)



(b)



(c)

Fig. 2. The respondents' knowledge of the different natural languages used in documents

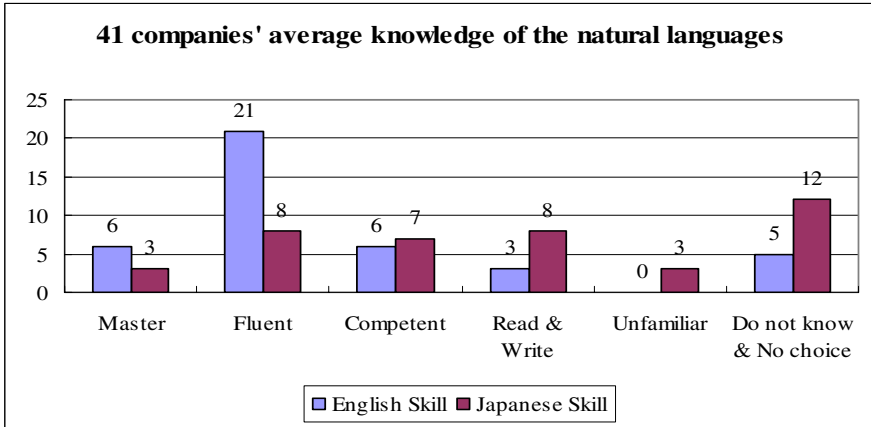


Fig. 3. The 41 companies' average knowledge of the natural languages

Discussion on RQ1. Kshetri [9] concluded that “The US and the UK are India’s major markets, thanks to its English competence. On the other hand, China’s cultural similarity to Japan and South Korea has facilitated its software export.” However, our results in Fig. 2(c) and 3 show that the employees of Chinese software suppliers on average have better knowledge in English than in Japanese. Moreover, results of Fig. 2(b) show that some respondents working with documents written in Japanese also have good English language skills. The possible reason is that most employees in the investigated Chinese software suppliers have a Bachelor’s or higher degree. In order to get such degrees, they must pass an English certification called CET-4

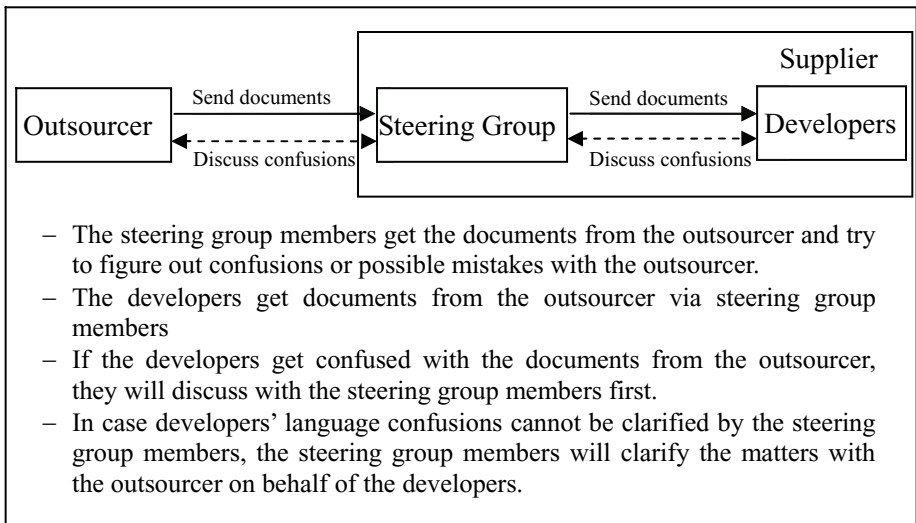


Fig. 4. The steering group process

(College English Test band 4) or CET-6 to qualify their competence of English. Since Chinese software suppliers have successfully developed large software systems for Japanese outsourcers, their equivalent or better English skills indicate a big potential to develop software for outsourcers from English speaking countries. In addition, our talks with interviewees in the pre-study reveal that some companies are using a **steering group** with languages competent people as a buffer between developers and outsourcers (as shown in Fig. 4). In this case, the Chinese software suppliers managed to resolve the confusions caused by the possible poor language skills of the developers.

Thus, our conclusion on *RQ1* is: *Differences in natural languages may not be the barrier for Chinese software suppliers to develop software for English speaking outsourcers. Having a steering group with good English language skills may help to reduce the language impact.*

4.2.2 Investigating RQ2: What is the Most Common Mean of Communication Between Foreign Outsourcers and Chinese Suppliers?

Results on RQ2. To study this research question, we listed several possible channels of communication, such as email, video conferences, and face-to-face meetings, which could be used to discuss issues between outsourcers and suppliers. For each communication type, we asked whether it has been *mainly* used to specify, clarify, or negotiate issues, such as requirements, business, and schedule. If the communication type has been used for more than one issue, the respondents can mark several alternatives.

We calculated the number of projects using a certain communication type to a specific issue vs. the total number of projects (i.e., 53). For example, 28 projects mainly used email to discuss requirements related issues. Thus, the result is 28/53 or 53%. The summary of all results stand in Table 1. Since the maximum number value in Table 1 is 60%, we divided them into four categories to simplify the presentation. We excluded numbers less than 30%. Numbers more than 30% and 40% are replaced with one plus. Numbers between 40% and 50% are replaced with two plusses. Numbers more than 50% are replaced with three plusses.

The data in Table 1 show that email correspondence was used on almost all issues, such as discussing requirements, harmonizing the schedule, handling changes, development, and testing. The popularly used synchronous communication types are telephone meetings and face-to-face meetings, where the face-to-face meetings are most used to discuss the requirements and business related topics. On the other hand, net meetings and video conferences are less used than others.

Discussion on RQ2. The previous study [4] proposed that synchronous communication means are more suitable to be used in the outsourcing contexts than asynchronous communication means. However, few empirical studies have investigated how the synchronous and asynchronous communication types are used in practice. Our results show that both the synchronous, such as telephone meetings and face-to-face meetings, and the asynchronous communication type, such as email, have been used all over. However, different communication types have been used to solve different issues. The advantage of email is that it is much cheaper than synchronous communication channels, and is not affected by the distribution of both space and

time. Chinese software suppliers have used email to discuss almost everything with outsourcers. On the other hand, face-to-face meetings are mainly used in requirements and business related topics. A case study in one Latvian software supplier [11] shows that email is used much more than other communication channels, while net meetings and video conferences are almost never used. Our results from Chinese software suppliers show the same trend. However, different time zones and holidays between the suppliers and outsourcers may affect their available communication types. If we divide these projects into two groups based on having or not having time zone problems (i.e., Japanese vs. Western), the two groups show similar results as in Table 1. Thus, the different time zones and holidays between the suppliers and outsourcers may not affect the results of our study.

Our conclusion on *RQ2* is: *Email is mainly used for development issues, while face-to-face meetings are popularly used for management and requirements issues. Net meetings and video conferences are the least used communication types.*

Table 1. Means of communications and issues to be discussed (N = 53, Multiple Choice)

	Asynchronous	Synchronous			
	Email	Net meeting	Phone meeting	Video conf.	Face-to-face meeting
Requirements	+++	+	++	+	+++
Development	++	+	++	+	+
Testing	++				
Maintenance					
Staff arrangement		+			
Harmonizing schedule	+++	+	++	++	++
Handling changes	+++		++	+	++
Business related issues					+++

4.2.3 Investigating RQ3: What are the Main Reasons for Overtime Work at Chinese Software Suppliers?

Results on RQ3. To investigate this research question, we asked whether there was overtime work in Chinese software suppliers. Results show that 51 out of the total 53 projects had overtime work, usually unpaid. We also asked the respondents their average working hours per week, the percentage of employees in their companies involved in overtime work, and their outputs during overtime work vs. their total outputs. The results show that, the average work hours per week are about 48, meaning the average overtime is 8 hours per week (20% more). Since 70% of the full-time employees are involved in overtime work, the expected relevant outputs from the overtime hours should be 14% (20% * 70%) of that from normal hours. However, the real “output” production from overtime work is 35%/65% (35% outputs are from the overwork hours and 65% outputs are from the normal working hours), or 54%, which

reveals that overtime work is 3 times more efficient than work during normal working time.

To know the reasons for overtime work, we listed several possible alternatives from the literature [12] and from the feedbacks of the pre-study. The respondents were asked to select one or more of the alternatives. More than 70% of the respondents point out that design or requirements changes are the major reason for working overtime (see Table 2). In addition, nearly 60% of the respondents think overtime work is caused by initial underestimation of project effort and duration.

Table 2. Reasons for overtime work (N=51, Multiple Choice)

	Number of projects	Percentage
Different time zones	7	14%
Different holidays	10	20%
Design or requirements changes	37	73%
Required by the outsourcers	15	29%
Insufficient local competence	16	31%
Over-optimism of schedule or effort	30	59%
Resolve some specific problems	1	2%

Discussion on RQ3. We found that the major reason for overtime work in Chinese software suppliers is design or requirements changes, rather than time zones differences. Furthermore, initial underestimation of effort and duration is also an important reason for overtime work. Some of the projects were underestimated because of lack of experience, but many were also victim of tactical bidding [12]. That is, to get a contract, a supplier indicated an unrealistic time schedule or effort to the outsourcer. This then leads to later overtime work for keeping the deadline or budget. If the overtime work (20% more) in our investigated projects is paid, which is usually 1.5 time of normal payment, the real budgets of our investigated software outsourcing projects should be increased by 30%. This means that software outsourcing projects suffer from similar under-budgeting, which is about 30% to 40% percentage [18], as other software projects.

In our pre-study, some of the interviewees attributed the frequent and unpaid overtime work in their companies to companies' encouragement on overworking. As a result, the employees may not have tried to work as effectively as they could during normal hours and may have postponed certain work into the overtime hours. This may explain our finding that the overtime work is more productive than normal time work. Also the less interruption from other employees could be another reason for more productive overtime work. Since our results show about 70% full-time employees worked overtime, it is hard to conclude that the interruptions in the overtime are less than in the normal time.

Our conclusion on *RQ3* is: *The main reasons for overtime work at Chinese software suppliers are design or requirements changes and initial over-optimism of the budget. The cultural encouragement of overtime work may also reduce the efficiency in the normal work hours.*

5 Final Discussion

5.1 General Discussion

Based on our results of **RQ1** to **RQ3**, we suggest the following strategies for Chinese software suppliers to improve their efficiency.

Manage the possible language differences with a steering group

The language factor is one of the reasons for the success of software outsourcing. The successes of software industries in India, Philippines and Singapore are accepted examples [4, 9]. Before our study, few studies have systematically examined the language skills of employees of Chinese software suppliers. The outsourcers from English speaking countries may still be skeptical to the English language skills of their Chinese suppliers. However, our results show that the differences in natural languages are not the barrier to Chinese software suppliers. For the Chinese suppliers, one possible method to manage the language effects is to build a steering group that is comprised by language competent project managers and senior developers. The members of the steering groups can work as a bridge between the external outsourcers and internal developers. Since most communication between the internal developers and the external outsourcers need to go through such a steering group, we later need to study how to improve the performance of such a group.

Do not overuse synchronous communication methods

A previous study [4] shows that synchronous communication channels usually have more advantages than asynchronous communication channels. However, our results and another previous study [11] show that email is still used frequently by the outsourcers and suppliers. On the other hand, synchronous channels, such as net meetings and video conferences are rarely used. To improve the efficiency of communications and to reduce unnecessary communication costs, we recommend that suppliers should avoid excessive synchronous communications. Face-to-face meetings and telephone meetings should be mainly used to discuss the business related issues and requirements. For issues related to design, development, and testing, email may work as an efficient communication type.

Manage the requirements changes and avoid unnecessary encouragement on overtime work

In software development, the requirements evolution can be substantial, e.g., 1% per month or 50% over 3 years [17]. Our results show that the main reason for overtime work in Chinese software suppliers are design or requirements changes initiated by the outsourcer. Another reason is the initial underestimation of effort, which is often caused by tactical bidding. In addition, we observed that there is an unnecessary and perhaps cultural encouragement for overtime work, which may cause employees to postpone their work from normal work hours to overtime hours. We recommend the Chinese software suppliers to appoint a change management group to systematically discuss and negotiate changes with outsourcers. Moreover, the Chinese suppliers need to improve their work efficiency in normal hours and try to avoid tactical bidding. An overall remedy to most of these issues is incremental development to reduce the risks

for overruns, but this may be hard to combine with the existing outsourcer-supplier contracting process.

5.2 Threats to Validity

Construct validity. In this study, most variables and alternatives are taken directly, or with little modification from existing literature. We did a pre-study to ensure the quality of the questionnaire, and nearly 10% of the questions and alternatives in the final questionnaire were revised based on the pre-study. With respect to **RQ2**, one possible threat to construct validity is that we asked only the respondents whether they used some types of communication on each topic or not (i.e., binary scale) without asking for the frequency of usage (e.g., how often do they use a certain communication channel). To investigate the **RQ3**, we have tried to use quantitative metrics, such as lines-of-code, to measure project outputs in the overtime hours. However, results of the pre-study show that most companies did not record such data. Our currently used subjective measurements may bring threats to conclusions of this research question.

Internal validity. We promised respondents in this study a final report or the annual membership of the CSO which worth of 500 Chinese Yuan. Most respondents took part in this survey as volunteers and selected the report as reward. We therefore generally believe that the respondents answered the questionnaire truthfully. However, a possible threat is that the respondents might have failing memory on past projects, because our unit of study is a finished project.

External validity. There were more than 11,550 software companies registered in China in 2005 [2]. However, the initial database we used contained only about a half of them. Although we have put a lot of effort on collecting data, we only get from 41 companies out of randomly selected 2000 companies. For the remaining 1959 companies, we could not know their reasons for not participating. The respondents answered the questionnaires based on finished projects, which were selected based on convenience. All the above issues may bring external threats to the conclusion of this study.

6 Conclusion and Future Work

This paper has presented results of a state-of-the-practice survey on Chinese software outsourcing in industrial projects. The main conclusions of our survey are:

- Differences in natural languages are not the critical factor for the success of Chinese software suppliers. Chinese suppliers have sufficient English skills and routines (e.g., steering groups) to resolve languages differences.
- Email is the most used communication channel for resolving development issues, while face-to-face meetings are popularly used for management and requirements issues. Net meetings and video conferences are not much used.
- The main reasons for overtime work at Chinese software suppliers are design or requirements changes and initial over-optimism of the effort or duration.

The current solution, i.e., unpaid overtime work, however, may reduce the efficiency in the normal work hours.

- Finally, since China has no comprehensive, national database of IT companies, it is difficult to select a random sample of participants in such surveys.

The results of this study discover several important issues that we are going to investigate in the future.

- We illustrate the most common communication channels without knowing their efficiency for discussing a specific topic. Further studies on the efficiency of different communication means are needed.
- We discover that using a steering group may be a proper strategy to mitigate the effect of languages or cultural differences. However, how to improve the performance of such steering group needs to be further studied.
- Although we find that overtime work is more efficient than work in normal hours, a further case study to verify our conclusion with a more reliable quantitative metrics is needed.
- Our results show that it is important to control requirements changes to avoid overruns of the outsourcing projects. How to use alternative processes, such as incremental or agile ones, in projects across geographic and company boundaries needs future investigation.

Acknowledgements

This study was a joint research effort between BJUT and NTNU, partially funded by the Norwegian SEVO project with grant 159916/V30. We would like to thank the CSO for data sampling and questionnaire collection. We also thank all participants in the survey.

References

1. Ahmed, R.E.: Software Maintenance Outsourcing: Issues and Strategies. *Computers and Electrical Engineering* 32(6), 449–453 (2006)
2. Ministry of Information of the People's Republic of China & Chinese Software Industry Association: Annual Report of China Software Industry (2006), <http://www.soft6.com/news/detail.asp?id=15759>
3. Aubert, B.A., Rivard, S., Patry, M.: A Transaction Cost Model of IT Outsourcing. *Information and Management* 41(7), 921–932 (2004)
4. Carmel, E., Agarwal, R.: Tactical Approaches for Alleviating Distance in Global Software Development. *IEEE Software* 18(2), 22–29 (2001)
5. Donahoe, D.N., Pecht, M.: Are U.S. Jobs Moving to China? *IEEE Transactions on Components and Packaging Technologies* 26(3), 682–686 (2003)
6. Ferguson, E., Kussmaul, C., McCracken, D.D., Robbert, M.A.: Offshore Outsourcing: Current Conditions and Diagnosis. In: *Proceedings of the Thirty-Fifth SIGCSE Technical Symposium on Computer Science Education*, Norfolk, Virginia, USA, March, vol. 36(1) pp. 330–331 (2004)

7. Kankanhalli, A., Tan, B.C.Y., Wei, K., Holmes, M.C.: Cross-cultural Differences and Information Systems Developer Values. *Decision Support Systems* 38(2), 183–195 (2004)
8. Krishna, S., Sahay, S., Walsham, G.: Managing Cross-cultural Issues in Global Software Outsourcing. *Communications of the ACM* 47(4), 62–66 (2004)
9. Kshetri, N.: Structural Shifts in the Chinese Software Industry. *IEEE Software* 22(4), 86–93 (2005)
10. Lacity, M.: Lessons in Global Information Technology Sourcing. *Computer* 35(8), 26–33 (2002)
11. Smite, D.: Global Software Development Projects in One of the Biggest Companies in Latvia: Is Geographical Distribution a Problem? *Software Process Improvement and Practice* 11(1), 61–76 (2006)
12. Taylor, H.: Critical Risks in Outsourced IT Projects: The Intractable and the Unforeseen. *Communications of the ACM* 49(11), 74–79 (2006)
13. Walsham, G.: Cross-cultural Software Production and Use: A Structural Analysis. *MIS Quarterly* 26(4), 359–380 (2002)
14. Herbsleb, J.D., Grinter, R.E.: Splitting the Organization and Integrating the Code: Conway's Law Revisited. In: *Proceedings of the 21st International Conference on Software Engineering*, Los Angeles, CA, USA, pp. 85–96 (May 1999)
15. Wong, J.: China's Economy in 2005: At a New Turning Point and Need to Fix Its Development Problems. *China & World Economy* 14(2), 1 (2006)
16. Herbsleb, J.D., Paulish, D.J., Bass, M.: Global Software Development at Siemens: Experience from Nine Projects. In: *Proceeding of the 27th International Conference on Software Engineering*, St. Louis, Missouri, USA, pp. 524–533 (May 2005)
17. Madhavji, N.H., Fernandez-Ramil, J., Perry, D.: *Software Evolution and Feedback: Theory and Practice*. John Wiley & Sons, West Sussex (2006)
18. Moløkken, K., Jørgensen, M.: A Review of Surveys on Software Effort Estimation. In: *Proceedings of the 2003 International Symposium on Empirical Software Engineering*, Rome, Italy, pp. 223–231 (September/October 2003)

Understanding Lacking Trust in Global Software Teams: A Multi-case Study

Nils Brede Moe¹ and Darja Šmite²

¹ SINTEF Information and Communication Technology

NO-7465 Trondheim, Norway

Nils.B.Moe@sintef.no

² University of Latvia

LV-1050, Raiņa bulv.19, Rīga, Latvia

Darja.Smite@lu.lv

Abstract. Many organizations have turned toward globally distributed software development in their quest for higher-quality software delivered cheaply and quickly. But this kind of development has often been reported as problematic and complex to manage. One of the fundamental factors in determining the success and failure of globally distributed software teams is trust. The aim of our work has therefore been to describe the key factors causing lack of trust, and the main effects of lacking trust in such teams. From studying 4 projects, all located in two different countries, with trust problems we found the key factors to be poor socialization and socio-cultural fit, lack of face-to-face meetings, missing conflict handling and cognitive based trust, increased monitoring and too little communication. The effect of lacking trust was a decrease in productivity, quality, information exchange, feedback and morale among the employees; the monitoring increased and the employees doubted negative feedback from manager.

Keywords: Trust, global software development, global software teams, virtual teams, multi-case study.

1 Introduction

1.1 Global Software Development – Different, Complex, Urgent

Several organizations have turned toward globally distributed software development (GSD) in their quest for higher-quality software delivered cheaply and quickly. Today, more software projects are run in geographically distributed environments, and global software development is becoming a norm in the software industry [4].

GSD is said to have significant challenges with respect to communication, coordination and control issues, because of the temporal, geographical and socio-cultural distance between members of the joint development team [27]. For this GSD is recognized as considerably more complex to manage than even the most complex in-house projects [3, 17].

What distinguishes globally distributed projects from in-house projects are the environmental properties, also called global factors [23], that even capable project managers often overlook. However, the reason for failure of global projects is not the lack of capability, but a lack of awareness of issues, problems, and barriers associated with global work [6]. Likewise Sahay and Nicholson describe that the unpredictable nature of the risks in a global environment heightens the potential for unintended consequences [21]. The characteristics of GSD can be defined as follows [23]:

- Multisourcing – multiple distributed member involvement in a joint project, characterized by a number of collaboration partners.
- Geographic distribution – partners are located far away from each other.
- Temporal diversity – characterized by the level of working hours overlay.
- Socio-cultural diversity – level of social, ethnic, and cultural fit.
- Linguistic diversity – characterized by the level of language skills.
- Contextual diversity – level of organizational fit (diversity in process maturity and work practices).
- Political and legislative diversity - effect of cross border collaboration due to political threats or threats associated with incompatibility of laws.

Threats caused by the diversity that exists among the distributed teams involved in a project are seen as unavoidable conditions. These threats can lead to unexpected costs, considerable time delays and undermined morale of the collaborating teams.

The body of knowledge on global software development has been crafted over time, but there is still significant understanding to be achieved, methods and techniques to be developed, and practices to be evolved before it becomes a mature discipline [4, 21].

1.2 GSD Teams and Trust

A GSD team is a team whose members collaborate on a common software project while working across geographic, temporal, cultural, and relational boundaries to accomplish an interdependent task. A GSD team can also be characterised as a Virtual Team [18]. Organizations are driven to virtual forms in order to be more flexible, agile, responsive, and inexpensive [3]. One of the fundamental factors that are believed to be important in determining the success and failure of virtual teams is trust [12, 16, 18]. We define trust as “*the shared perception by the majority of team members that individuals in the team will perform particular actions important to its members and that the individuals will recognize and protect the rights and interests of all the team members engaged in their joint endeavour*” [25]. Virtual teams that exhibit a high degree of trust experience significant social communication as well as predictable communication patterns, substantial feedback, positive leadership, enthusiasm, and the ability to cope with technical uncertainty [14]. Trust functions as the glue that holds and links virtual teams together [16].

Jarvenpaa et al [15] argue that trust in a virtual team has a direct positive effect on cooperation and performance, and an increase in trust in a team with a weak structure is likely to have a direct, positive impact on team members’ attitudes and perceived outcomes.

Since trust is a fundamental factor for virtual teams it is reasonable to believe that trust is also important for GSD teams. Vanzin et al and Davidson and Tay [5, 24] argue that trust is a recurring problem in GSD teams, because of geographical, temporal, organizational, cultural and political differences among the team members. Carmel in his book „Global Software Teams” argues that distance is an impediment to building relationships of trust [3]. However, due to the cost benefit of outsourcing versus in-house and other cost saving strategies, most of the team members never meet.

To understand the importance of trust in GSD and to increase the level of trust in such a team it is essential to understand what leads to lacking trust, and the effect of lacking trust in a GSD team. From our overview of the research literature we believe that the existing literature does not describe this, which is also confirmed by Edwards and Sridhar [11]. Therefore our research questions have been: What are the key factors causing lacking trust in a GSD team located in two countries? What are the main effects of lacking trust in a GSD team located in two countries?

In the next sections we first use previous research on virtual teams and GSD to understand the different threats against establishing trust in a GSD team. We then use theory from virtual teams and teamwork to describe the possible effect of lacking trust in a GSD team, before we apply these reasons and effects to a multi-case study to understand the effect of mistrust on GSD team performance.

2 Key Factors Causing Lack of Trust

Trust in virtual teams needs to be developed quickly because teams may only interact for a short period of time or may be working on a task that is of great importance and urgency [14, 16]. Earlier work on trust in the virtual environment has found that short-lived teams are in fact able to develop high trust but they do so by following a swift trust model rather than the traditional model of trust development [13, 14].

Virtual teams need to focus on the maintenance as well as the development of trust, but there are many threats against achieving a high trust level in a virtual team:

- *Cognitive-based trust.* Virtual teams need to focus on the cognitive dimension of trust (e.g. competence, reliability, professionalism) [16]. Therefore it is important to provide task-relevant background information to virtual team members so that members can quickly develop cognition-based trust. If the remote team does not deliver what is expected this will decrease the cognitive-based trust.
- *Poor socialization.* Socialization strategies may help managers develop trust also in virtual teams [14, 16]. Team members should travel to remote sites to engage in a team-building activity to engender lasting trust [20].
- *Missing face-to-face meetings.* Such meetings are considered irreplaceable for both developing and repairing trust in virtual teams [2, 3, 19]. Carmel [3] argues that “trust needs touch”. If there is no face-to-face communication in a virtual team, this tends to hinder effective communication. E.g. when team members communicate about mutual responsibility and obligations, different perceptions of their commitments may develop, creating a potential for trust decline [19].
- *No conflict handling.* Conflicts in a global development are inevitable [17], and it is often difficult to maintain trust when conflicts among team members emerge. So missing conflict handling is a threat against building and maintaining trust in a virtual team [14, 16].

- *Too little communication.* Virtual teams in a low trust situation need frequent communication to increase the trust level. The frequent communication is important for providing constant confirmation that team members are still there and still working [15].
- *Unpredictability in communication.* The frequency and predictability of communication, and the extent to which feedback is provided on a regular basis, improve communication effectiveness leading to higher trust and improving team performance [13, 14]. Inexperienced virtual team members may experience anxiety or trust decline due to negative interpretations of silence or delays associated with time dispersion [19].
- *Increased monitoring* (behavioural control and too much communication). The use of behavioural controls, such as having members file weekly reports and assigning specific tasks, has been found to be associated with a decline in trust among virtual team members [19]. Also too much communication might cause members of a team to be suspicious that others are monitoring them and this decreases the trust [15].
- *Poor socio-cultural fit.* Duarte and Snyder distinguish three types of culture – national, organizational and functional – and claim that they constitute one of a virtual team’s most significant boundaries [10]. Furthermore, they describe that being hidden like an iceberg, culture affect people’s assumptions, behaviours, and expectations about leadership practices, work habits, and team norms [10] pp.54.

3 The Effect of Lacking Trust

Based on a literature review, Salas et al. [22] argue that it is possible to condense the teamwork knowledge into five core components, which they call the “Big Five” of teamwork, and three coordinating mechanisms. The 5 components affecting the team effectiveness are:

- Team leadership;
- Mutual performance monitoring;
- Backup behaviour;
- Adaptability;
- Team orientation.

The 3 coordination mechanisms are: shared mental models, closed-looped communication, and mutual trust. They are called coordination mechanisms because they are necessary facilitators of the 5 components.

In this model trust is needed to make the team members work interdependently, they must be willing to accept a certain amount of risk to rely on each other to meet deadlines, contribute to the team task, and cooperate without subversive intentions.

Dirks and Fern's [9] review of the literature on the role of trust in organizational settings demonstrates that trust has either direct or moderating effects on a variety of desired performance and behavioural outcome variables. In their view, trust facilitates the effects of other determinants on performance or behavioural outcomes because trust provides conditions, under which certain outcomes are more likely to occur.

Bandow [1] argues that a lack of trust within the group may interfere with how effectively individuals contribute to teams, may reduce overall team performance, increase cycle time, create higher costs and potentially impact product quality.

The effect of lacking trust can be described as [1, 9, 22]:

- *Decreased information exchange and feedback* – A low level of trust is associated with suspiciousness of information, and therefore decreased information exchange and feedback [1, 9, 22].
- *Competition and not cooperation* - If one does not trust a partner, it might be difficult to work toward the joint goal and it is likely that the employees will pay more attention to competitive motives and not to cooperation [9], and even withdraw from participation because they feel insecure [1].
- *Self-protection* - If one does not trust the manager, the individual finds it worrisome to behave as expected; and the management's request is likely to exert a much weaker effect on the individual's behaviour, as the individual diverts resources to self-protection [9]. This will hinder the team leader from effectively managing the team.
- *Doubt negative feedback from manager* - When there is a negative feedback from a manager with low trust, it is likely that the employee will doubt the accuracy of the feedback [9, 22].
- *Relationship conflict* - Under low trust, task conflict within a group is interpreted negatively and subsequently results in relationship conflict [9, 22].
- *Individual goals over group goals* - Under low trust the individuals in a group will direct their efforts toward individual goals, instead of the group's goals [9].
- *Team not self-correcting* - Low trust will decrease the mutual performance monitoring, which means the ability to develop common understandings of the team environment and possibility to accurately monitor team member performance. This is essential to identifying mistakes and lapses in other team members' actions, and providing feedback regarding team member actions to facilitate self-correction [22].
- *Not shifting workload among members* - Decrease in the mutual performance monitoring will again affect the backup-behaviour. This is the ability to anticipate other team members' needs through accurate knowledge about their responsibilities. This includes the ability to shift workloads among members to achieve balance during high periods of workload or pressure [22].
- *Productivity and quality decrease* - Since the lack of trust reduces team performance [1, 9, 22] this reduces the productivity and quality.

4 Research Context and Method

The context for this research is the Latvian software development company LatSoftware (the company name is changed for confidentiality reasons), situated in Riga. The company was established in the late 80s and changed its owners and/or structure several times. It has been oriented towards the international market, focusing on providing software development outsourcing services for the public sector, telecommunications, insurance and banking, as well as tourism and logistics. LatSoftware has successfully accomplished more than 200 projects both in Latvia, Western Europe and Scandinavia. At the present time the company represents a joint venture with over 380 employees.

The work reported in this paper is a multi case study [26] to understand reasons and effect of lacking trust in global software development within LatSoftware. This is a multi case holistic study, in which we study one phenomenon in several projects in

one company. Since we are studying the reasons and the effects of lacking trust we picked four global software development projects that all reported trust problems.

4.1 Data Sources

We have used multiple data sources (see Table 1) in the analysis as described in the following. In the analysis, we rely mainly on qualitative interviews, as these provide a rich picture of the reason and effect of lacking trust. We have also used results from postmortem meetings [7] held during and at the end of the project. A postmortem meeting focuses on describing what went well and what did not work in the project, and then a root-cause analysis is performed on the main issues. Using postmortem meetings it was possible to find the root-causes of problems related to trust.

Project problems were also recorded using previously developed problem checklists that have been developed from an extensive literature review and from former project experience. Postmortem analysis data was recorded with the help of a camera during the meetings and later transcribed by the researchers in the postmortem analysis document, which was afterwards sent to the participants for approval.

In this study we have focused on exploring the investigated company's problems, whose employees are acting as suppliers in the studied projects. Due to the limited availability of information about project customers, we do not present data about their team size in Table 1.

Table 1. Data sources

Pro name	Duration	Project type	Supply chains Location* and Team size	Effort	Data collection
A	1995 – present	SW product development and maintenance	DE→DE (3)→LV (5)	46080 hours	Interviewed current project manager, previous project manager and one developer Problem checklists
B	2002 – 2006	SW product development	UK→UK (13)→LV (16)	40480 hours	Interviewed project manager and 3 team leaders. Postmortem analysis Problem checklists
C	2006	SW pilot product development	SE → LV (3)	320 hours	Interview with project manager Problem checklists
D	2005	SW product development	NO (2)→LV (6)→LV (5)	1460 hours	Interview with project manager Postmortem analysis Problem checklists

* DE - Germany, LV - Latvia, UK - the United Kingdom, SE - Sweden, NO - Norway

4.2 Data Analysis

Data analysis was performed in several steps. First, we read all interviews and postmortem analysis data, and coded interesting expressions of opinions related to trust in the text. Then we assigned the expressions to the categories of “the reasons of lacking trust” and “the effect of lacking trust” found in the literature on Global Software Development, Teams and Virtual Teams. For example, “unwillingness to collaborate caused by threat of being fired due to switching to outsourcing mode” was coded as “the reasons of lacking trust” and linked with “competition and not cooperation”. To avoid bias and misunderstanding, the conclusions from our coding was sent back to the interviewees for approval.

5 Results

In this chapter we describe the four global projects run in the investigated software house. We present each project, followed by a description of why it was lacking trust, and the effects of this.

5.1 Project A

Overview. Project A is a long-term ongoing software enhancement project with close collaboration between 5 Latvian developers and 3 representatives from a German company that build a software product for their customer. The German team performs project management and systems analysis, while outsourcing coding activities. Recent changes in project management from the Riga side didn't get much appreciation from the customer side due to increasing costs. The customer has moved part of the work to a lower price partner from another country, and signalled that future project problems can lead to cancellation of the project

Reasons for lacking trust. The project extensively uses modern collaboration tools such as video conferencing and instant messaging. However, the Riga team argues that this doesn't compensate for the lack of face-to-face meetings. The Riga team also sense that their German partners are afraid that the Riga team is not dedicated to the project, and therefore try to control them by constantly monitoring their performance. It took 10 years for the German partners to visit their Latvian team members. This first meeting uncovered that the German team did not know much about their partner and they were surprised to see the modern offices with high level security and technical equipment. Their perception of the remote team members changed and further collaboration with frequent meetings for some time improved overall project performance and especially team morale and psychological comfort.

However, diversity in process maturity has put the partners into a collision. Corporative culture doesn't allow the Riga team to act in a too agile way without any project management. And with respect to recent disputes between the partners considering these changes, the Riga team acts by competitive motives, and feels not trusted and insecure again.

Effect of lacking trust. The Riga team has continuously suffered from lack of trust and commitment, which dramatically decreased the ability to self-correct, which again initiated extensive monitoring from the contracting partner. This again affected the trust level negatively. Not satisfied with collaboration the contracting partner frequently required to change project leads. Searching for more beneficial collaboration partners puts the Riga team in competition. As a result the project atmosphere negatively influences team morale, productivity and causes conflicts in relationship.

5.2 Project B

Overview. Project B is a software product development and enhancement project run by a UK software house that outsourced software development to a Latvian partner from Riga. Programming activities in this project were performed in both countries. The outsourcing was a strategic choice from the management in the UK software house. This was however, according to the project manager from Riga, not appreciated by the UK team representatives directly involved in collaboration.

Reasons for lacking trust. The Riga project manager reported several problems related to lacking trust. The UK and Riga teams did not share a joint view on their collaboration due to diversity in their work practices. Such problems as poor cultural fit, dominant use of asynchronous tools, unwillingness or slowness of the UK team to act on partner's suggestions, led to poor, unpredictable communication. Due to a lack of joint problem handling, poor socialization and lack of face-to-face meetings process performance didn't take place.

Effect of lacking trust. The Riga project manager reported that sometimes his team seemed to lack motivation to give the customer value for money – manifesting itself in lower than reasonably expected productivity levels. Poor socialization and lack of face-to-face meetings resulted in a lack of team spirit, trust and commitment between the partners. Lacking trust and poor communication has also decreased information exchange and feedback. Lacking understanding of the context of decision making, the negative feedback from the continuously indifferent partner was doubted.

5.3 Project C

Overview. Project C was a pilot project in order to evaluate the investigated Riga software house as an external provider of coding for a software house in Sweden, which has recently switched to outsourcing mode. Their cooperation started by developing a small piece of software and was afterwards suspended.

Reasons for lacking trust. Both partners faced an increasing complexity of distributed multi-team management regarding the necessity of overcoming diversity and lack of joint procedures and tools. After joint risk management meetings with the customer, the project manager from Riga reported that the customer faced the

necessity to change and appeared not to be ready for that. Trust and belief in joint performance was affected by poor cultural fit, too little communication, lack of socialization and face-to-face meetings.

Effect of lacking trust. According to Riga project manager's opinion, the customer's employees felt insecure about their jobs, due to the corporative decision to switch to an outsourcing mode. Remote team members were put in competition instead of collaboration causing a productivity decrease. Consequently, the customer team's individual goals dominated over shared project goals. All task conflicts within the joint team were interpreted negatively. Lack of conflict handling finally led to collaboration suspension.

5.4 Project D

Overview. Project D is a complex project involving a customer from Norway, a direct supplier from Riga and a remote programmers team from a small Latvian town situated in the poorest region around 250km from the city. However, our attention in this case study was focused particularly on collaboration between two separate teams within one country and one organization not separated by country borders. Both supplier teams work for the same company and perform development by joint effort.

Reasons for lacking trust. Despite the fact that all the team members work for the same company, in comparison with the Riga team, the remote team works in a poorer environment and has significant problems with technology and communication lines. The remote team reported on lack of trust and belief in their performance by the Riga project manager, which he confirmed. Lacking trust in project D was caused by concerns of the project manager about successful remote team performance, the inability of direct control and communication problems due to distribution and poor technological infrastructure, lack of socialization and face-to-face meetings. Despite the fact that both teams are situated in the same country, they experienced socio-cultural diversity which also affected trust.

Effect of lacking trust. Lack of trust in this project decreased information exchange between the team members and increased suspicion and the desire to control by the Riga project manager. His behaviour led to self-protection and apprehension of the manager's feedback. This also resulted in low motivation for self-correction within the separated teams.

5.5 Key Factors Causing Lack of Trust and the Effects of Lacking Trust in the Projects

We have examined issues uncovered in related studies regarding trust in virtual environments within the investigated projects. A report of the occurrence of the identified key factors causing lack of trust is in Table 2 below.

Table 2. Key factors causing lack of trust in the project

Reason for lacking trust	Projects			
	A	B	C	D
Cognitive-based trust	✓	✓		✓
Poor socialization	✓	✓	✓	✓
Missing face-to-face meetings	✓	✓	✓	✓
No conflict handling		✓	✓	✓
Too little communication		✓	✓	✓
Unpredictability in communication		✓		✓
Increased monitoring	✓	✓		✓
Poor socio-cultural fit	✓	✓	✓	✓

Describing the main effects of lacking trust, all project managers reported that it to some level always influences customer satisfaction and supplier team morale.

A global environment puts new demands on trust achievement between the remote team members. An organization switching to outsourcing mode puts its own employees under threat of being fired. This leads to a competition instead of collaboration with the remote suppliers. Inability to achieve a shared understanding and compensation of diversity in work practices leads to remote team goal separation.

Table 3.The main effects of lacking trust

The main effects of lacking trust	Projects			
	A	B	C	D
Decreased information exchange and feedback	✓	✓		✓
Competition and not cooperation	✓		✓	
Self-protection	✓			✓
Doubt negative feedback from manager	✓	✓		✓
Relationship conflict	✓		✓	
Individual goals over group goals	✓		✓	
Team not self-correcting				✓
Not shifting workload among members				
Productivity and quality decrease	✓	✓	✓	✓

6 Discussion

In this paper we have used the literature to describe the key factors causing lacking trust, and the main effects of lacking trust while collaborating over geographic,

cultural and organizational boundaries. Then we have applied these key factors and effects to a multi-case study to understand the effect of mistrust on GSD team performance in a team situated in two countries. We have investigated projects that all have reported lacking trust; the data was only collected from the Latvian developers and managers.

6.1 Key Factors Causing Lacking Trust

From our study we found that poor socialization, lack of face-to-face meetings and poor socio-cultural fit were reported by all the projects. Lack of face-to-face meetings and poor socialization are probably related since it is difficult to socialize if you seldom or never meet. We think that poor socio-cultural fit may also be strengthened due to lack of face-to-face interaction and poor socialization. Other key factors for lacking trust were also reported frequently by the projects.

We also found additional factors leading to lack of trust. For instance, lack of language skills leads to poor socialization and communication problems, because employees with poor language skills tend to be afraid to speak over the phone. Inconsistency in work practices may lead to a lack of cognitive-based trust, misunderstandings and again cause increased monitoring. Involvement of unenthusiastic employees who lack previous experience in outsourcing projects can lead to a belief that the work cannot be done from a far off location. This negatively affects mutual socialization, communication and trust.

Finally we try to explain how the factors that characterize the GSD team are related to the key factors of lacking trust that we have found in this study:

- Multisourcing – increasing the number of collaboration partners involved in the project results in more complex communication, coordination and control. This again increases the number of sources of threat and complexity of trust achievement.
- Geographic distribution – leads to increased virtualness, communication problems, troubled socialization, and knowledge and awareness share.
- Contextual diversity – level of organizational fit, characterized by diversity in process maturity and inconsistency in work practices acts as a counterforce for shared environment development. Team members who do not share background and work habits seem to have less commitment to a joint team.

6.2 Effects of Lacking Trust

Like Dirks and Fern, and Badow [1, 9] we have found that lacking trust indeed may cause significant problems with performance and behaviour of the team members. The most frequently reported effect of lacking trust was productivity and quality decrease. This indeed proves the importance of trust for overall project performance. The next most frequently reported effect of lacking trust was decreased information exchange and feedback (3 projects). Another frequently reported effect (3 projects) was team members doubting negative feedback from their manager. Issues such as

team members not shifting their workload and not self-correcting were barely mentioned or not at all by the interviewed project members. This can be explained by the problem with information exchange and feedback, self-protection, competition and lack of cooperation. Because of these problems the team probably never had the chance to consider shifting their workload and self-correcting.

This and other comments point that although remote team members ought to form a joint team they consider distribution as team separator. After all, these projects demonstrate that there might be committed teams in each location and lack of team spirit between them.

The existence of committed and joint internal teams at every location that experience lack of trust may also explain why Project B and D, which have faced all of the mentioned key factors of lacking trust, have not reported as many effects of lacking trust as the other teams.

From our observations we would also like to add the following effects to the list of lacking trust outcomes:

- **Increased monitoring** (reported in Projects A, B and D) – in addition to a trust decline due to a pressing monitoring [15, 19], lacking trust in supplier performance and lack of direct control due to geographic and temporal distribution makes managers struggle with a desire to control instead of cooperating with the remote teams, resulting in increased monitoring and causing extra time for reporting. This also forms a locked loop.
- **Undermined morale of the employees** (reported in every project) – lacking trust creates a negative atmosphere that results in psychological discomfort of the members.
- **Threat of project cancellation** (reported in Project A and C) – we have also found that lacking trust may put the overall collaboration under threat.

6.3 Recommendations

The reason for failure of global projects is not the lack of capability, but lack of awareness of issues, problems, and barriers associated with global work [6]. From the multi-case study and the literature, the factors causing lack of trust can be linked with the global environmental characteristics – various diversities (organizational, socio-cultural, geographic, temporal, etc.) between the partners. We therefore emphasize the role of diversity and inconsistency awareness and the importance of flexibility and adaptability. Therefore, never start a distributed collaboration unprepared and without awareness of diversity. To face the key factors causing lack of trust we recommend:

- Go through the list of “key factors” and “main effects” of lacking trust, discuss this early with the team, and identify actions to meet these “threats“. E.g:
 - Invest in one or several face-to face meetings [1-3, 17, 19],
 - Invest in socialization activities for the whole team together [14, 16, 20],
 - Invest in groupware packages to provide remote team members with effective means of communication and compensate lack of personal contact during the project [[17]].

- Communicate expectations early and establish initial rules, in the form of a contract or trust structure, to spell out performance parameters for the team as a whole and for individual team members [1].
- Develop a 360° view by establishing a team intranet; facilitate publishing and updating individual, team, status and task information; encourage personal touches including personal pages [3].
- Create a common understanding of the work process, and how to cooperate in this process. This can be achieved by creating some common process elements. A common process workshop can be used to create this [8].
- Consider a software development method that provides both flexibility and adaptability, and that requires frequent communication. One solution to this is the use of agile methods [27].

7 Conclusion and Future Work

Trust is a recurring problem in GSD teams, because of geographical, temporal, organizational, cultural and political differences among the team members. Face-to-face meetings, active communication and socialization that are commonly used for building trust in software teams are a hard recipe for global software teams. Due to cost saving strategies, most of the GSD team members never meet.

In this paper we have conducted an empirical study that aimed to understand the reasons and effect of lacking trust on GSD team performance in four software projects in one company. All projects reported that lack of trust resulted in a decrease in quality and productivity. These and other findings leads to a conclusion that a company should consider the pros and cons of collaborating over borders and never start a distributed collaboration unprepared. Awareness of the importance of trust, the reasons for lacking trust and its effect, will help to avoid many problems of joint collaboration. However achievement of a high level of trust in GSD teams is not a simple question.

Accordingly further work should focus on investigating which methods for building and maintaining trust in GSD can be applied.

Acknowledgments

We appreciate the input received from project managers and other members of the investigated projects in LatSoftware, and thanks to Odd Nordland for proof reading and Torgeir Dingsøy for valuable feedback.

This research is partly supported by the Research Council of Norway under Grant 156701/220, European Social Fund under grant “Doctoral student research and post doctoral research support for university of Latvia” and the Latvian Council of Science within project Nr. 02.2002 “Latvian Informatics Production Unit Support Program in the Area of Engineering, Computer Networks and Signal Processing”.

References

1. Bandow, D.: Time to create sound teamwork. *The Journal for quality and participation* 24(2), 41 (2001)
2. Bhat, J.M., Gupta, M., Murthy, S.N.: Overcoming requirements engineering challenges: Lessons from offshore outsourcing. *IEEE Software* 23(5), 38+ (2006)
3. Carmel, E.: *Global software teams: collaborating across borders and time zones*. Prentice-Hall, Englewood Cliffs (1999)
4. Damian, D., Moitra, D.: Global software development: How far have we come? *Ieee Software* 23(5), 17–19 (2006)
5. Davidson, E.J., Tay, A.S.M.: *Studying teamwork in global IT support* (2003)
6. DeLone, W., et al.: Bridging global boundaries for IS project success. In: 38th Hawaii International Conference on System Scienc 2005. Big Island, HI, United States: Institute of Electrical and Electronics Engineers Computer Society, Piscataway, NJ 08855-1331, United States (2005)
7. Dingsøy, T.: Postmortem reviews: purpose and approaches in software engineering. *Information and Software Technology* 47(5), 293–303 (2005)
8. Dingsøy, T., et al.: A workshop-oriented approach for defining electronic process guides - A case study. In: Acuña, S.T., Juristo, N. (eds.) in *Software Process Modelling*, pp. 187–205. Kluwer Academic Publishers, Boston (2004)
9. Dirks, K.T., Ferrin, D.L.: The role of trust in organizational settings. *Organization Science* 12(4), 450–467 (2001)
10. Duarte, D.L., Snyder, L.T.: *Mastering Virtual Teams: Strategies, Tools, and Techniques that Succeed*, 2nd edn. A.W. Company, Jossey-Bass (2001)
11. Edwards, H.K., Sridhar, V.: Analysis of the effectiveness of global virtual teams in software engineering projects (2003)
12. Grabowski, M., Roberts, K.H.: Risk mitigation in virtual organizations. *Organization Science* 10(6), 704–721 (1999)
13. Jarvenpaa, S.L., Knoll, K., Leidner, D.E.: Is anybody out there? Antecedents of trust in global virtual teams. *Journal of Management Information Systems* 14(4), 29–64 (1998)
14. Jarvenpaa, S.L., Leidner, D.E.: Communication and trust in global virtual teams. *Organization Science* 10(6), 791–815 (1999)
15. Jarvenpaa, S.L., Shaw, T.R., Staples, D.S.: Toward contextualized theories of trust: The role of trust in global virtual teams. *Information Systems Research* 15(3), 250–267 (2004)
16. Kanawattanachai, P., Yoo, Y.: Dynamic nature of trust in virtual teams. *Journal of Strategic Information Systems* 11(3-4), 187–213 (2002)
17. Karolak, D.W.J.: *Global software development*. IEEE Computer Society Press, Los Alamitos (1998)
18. Martins, L.L., Gilson, L.L., Maynard, M.T.: Virtual teams: What do we know and where do we go from here? *Journal of Management* 30(6), 805–835 (2004)
19. Piccoli, G., Ives, B.: Trust and the unintended effects of behavior control in virtual teams. *Mis. Quarterly* 27(3), 365–395 (2003)
20. Rocco, E.: Trust breaks down in electronic contexts but can be repaired by some initial face-to-face contact, Los Angeles, CA, USA. ACM, New York, NY, USA (1998)
21. Sahay, S., Nicholson, B., Krishna, S.: *Global IT outsourcing: software development across borders*. Cambridge University Press, Cambridge (2003)
22. Salas, E., Sims, D.E., Burke, C.S.: Is there a big five in teamwork? *Small Group Research* 36(5), 555–599 (2005)

23. Smite, D., Borzovs, J.: A framework for overcoming supplier related threats in global projects. In: Richardson, I., Runeson, P., Messnarz, R. (eds.) EuroSPI 2006. LNCS, vol. 4257, pp. 50–61. Springer, Heidelberg (2006)
24. Vanzin, M. et al.: Global software processes definition in a distributed environment. Greenbelt, MD, United States: Institute of Electrical and Electronics Engineers Computer Society, Piscataway, NJ 08855-1331, United States (2005)
25. Webber, S.S.: Leadership and Trust Facilitating Cross-functional Team Success. *The Journal of management development* 21(3), 201 (2002)
26. Yin, R.K.: *Case Study Research: design and methods*, 3rd edn. vol. 5. Sage Publications, Thousand Oaks, CA (2003)
27. Ågerfalk, P.J.: Special Issue: Flexible and distributed software processes: old petunias in new bowls? table of contents. *Communications of the ACM* 49(10), 26 (2006)

Utilization of a Set of Software Engineering Roles for a Multinational Organization

Claude Y. Laporte¹, Mikel Doucet², Pierre Bourque¹, and Youssef Belkébir³

¹ Department of Software and IT Engineering
École de technologie supérieure
1100, rue Notre-Dame Ouest, Montréal, Québec, Canada, H3C 1K3.
{Claude.Y.Laporte,Pierre.Bourque}@etsmtl.ca

² Center of Competence Software Engineering
Bombardier Transportation
1101, rue Parent, St-Bruno, Québec, Canada, J3V 6E6
Mikel.Doucet@ca.transport.bombardier.com

³ ARINSO Africa
219, BD. Med Zerktouni
Angle BD. Brahim Roudani
CP 20100 El Maarif – Casablanca, Morocco
Youssef.Belkebir@arinso.com

Abstract. In this paper, we present the application of a set of software engineering roles. Role definitions were developed using internationally recognized software engineering reference documents for a major railway development organization: Bombardier Transportation. The description of the Software Architect role is explained. This paper will also illustrate how the role set could be used for project-specific needs during a typical project planning and launch session.

Keywords: Software Engineering, Roles definition, Process, standards.

1 Introduction

As stated by Humphrey [1]: “Without clearly identified responsibilities, it could take some time for a team to understand everything that it must do, to decide who should do each task...That is not so much because the engineers don’t want to take responsibility but rather because they don’t know what all the actions are or they are not sure whether anyone else is already doing them. They may also be reluctant to take on tasks that the team or team leader might plan to give to someone else.” This why we have conducted a project to improve the software engineering role definitions within the software engineering process definition of a large multinational organization. The project was also conducted because of the many conflicting and sometimes absent role definitions across the many Bombardier Transportation¹ sites

¹ www.bombardier.com

and projects. Moreover, many of the roles that had already been defined were defined only very briefly.

This paper presents the utilization of a set of roles in a project that includes software development. A detailed analysis and improvement initiative in regard to the role definitions within the Bombardier Engineering System Software Engineering (BES SWE) process definition has already been conducted [2, 3]. The BES SWE is the common software engineering process definition of Bombardier's Transportation division, in which each role definition specifies the purpose of the role, identifies the core responsibilities assumed by the role, and the hard and soft skills needed to perform the role.

Created in 1974 to provide subway wagons for the Montreal Transit Authority, Bombardier Transportation grew rapidly through many acquisitions to become the leading manufacturer of rail material for moving people. The company had 16,000 employees before Bombardier Transportation acquired, in 2001, ADtranZ. It is also interesting to note that ADtranZ was also the result of an upcoming merger between employees from sections of ABB and Daimler Chrysler. The acquisition of ADtranZ came with 20,000 employees with an engineering presence in 25 countries. The company also had to face many challenges of modern multinational organizations:

- Multidisciplinary system development,
- Multiple integrator-supplier relationships,
- Multi-country development,
- Multicultural teams,
- Downsizing/merger/turnover,
- Offshoring.

As an example of a complex software-intensive system, the company is working on the development of the European Rail Traffic Management System / European Train Control System (ERTMS / ETCS). This system will allow trains to cross borders without the need to change locomotive or driver. It also makes it possible for every train to be supervised individually, and for every train to be run according to its particular characteristics².

In order to facilitate the identification and deployment of technologies, the company established a number of corporate Centres of Competence in various engineering specialties. The Centre of Competence (CoC) in Software Engineering, located just outside Montreal (Canada), is where this project was coordinated. Some of the tasks of the Software CoC are:

- To reduce technical risks and quality deficiency costs;
- To support and monitor strategic initiatives;
- To assess, develop and deploy (e.g. training) software engineering technologies such as processes (BES SWE), methodologies and tools.

² Adapted from: *ERTMS/ETCS – for a competitive railway*, Bombardier Transportation, Rail Control Solution, Feb 2002.

In order to provide technologies to all divisions, and at a rapid pace, it has been decided to use a common vocabulary, common processes and common roles. The strategy that was developed is as follows:

- Adopt internationally recognized reference documents
 - Models
 - Standards
 - Body of Knowledge
- Develop common processes, work instructions and role definitions
 - Independent of the organizational structure and organizational changes.

The notion of the role is a core concept in the BES SWE, as it is in all software engineering process definitions, as shown in Figure 1 (excerpted from OMG00).

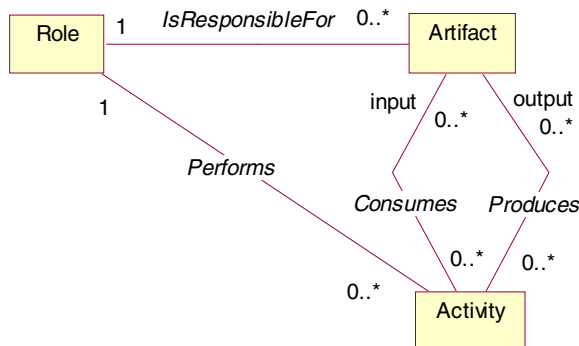


Fig. 1. Interaction of roles, activities and artifacts in a software engineering process definition

Table 1. BES SWE role definitions at the beginning of the project

<p>Management Category Senior Manager Project Manager Software Project Manager Software Quality Assurance Manager Product Manager Software Engineering Manager</p> <p>Software Engineering Category Software Team Leader Software Requirements Coordinator Software Architect Software Implementer Software Integrator Software Test Designer Software Tester</p>	<p>Software Engineering – Supporting Category Software Change Control Board Software Infrastructure Administrator Software Metrics Coordinator Software Process Engineer Software Project Coordinator Software Quality Assurance Engineer</p> <p>Other categories Customer Proposal Coordinator Safety representative Software Trainer Software Training Coordinator</p>
--	---

Roles perform activities that produce and consume artifacts. Roles are also responsible for artifacts. Of course, the same role may be performed in a given project by many people, and, conversely, one person may perform many roles. At the outset of the project, the BES SWE included 24 roles, as listed in Table 1, divided into four categories.

In order to facilitate the roll-out of the role definitions to all Bombardier Transportation software engineering sites, it was decided that the coverage analysis and subsequent improvements to the role definitions would be founded on internationally recognized reference documents. Notably, this was viewed as a way of adding credibility to the improved role definitions without giving the impression that one software engineering site was imposing its role definitions on the other sites. The selected reference documents were IEEE/EIA Standard 12207.0-1996, Standard for Information Technology–Software Life Cycle Processes [4], the IBM Rational Unified Process (RUP) [5]³ and the Guide to the Software Engineering Body of Knowledge (SWEBOK Guide) [6]. Each reference document is briefly described below.

1.1 IBM- Rational Unified Process

The IBM-Rational Unified Process (RUP) is a commercial object-oriented process framework for software development⁴. It includes a set of roles, activities, workflows and artifacts which describe the who, the how, the when and the what of a software development process. It can be tailored to company specifics and to various sectors of the industry.

1.2 IEEE/EIA Standard 12207

The IEEE/EIA 12207.0-1996 Standard for Information Technology–Software Life Cycle Processes is considered a key standard in terms of the definition of life cycle processes. It has notably been designated as the pivotal standard around which the Software Engineering Standards Committee (SESC) is harmonizing its entire collection of standards. This standard groups software processes into activities and tasks, and these are organized into three categories: Primary Processes, which are divided into Acquisition, Supply, Development, Operation and Maintenance; Supporting Processes, which are divided into Documentation, Configuration Management, Quality Assurance, Verification, Validation, Joint Review, Audit and Problem Resolution; Organizational Life Cycle Processes, which are divided into Management, Infrastructure, Improvement and Training.

1.3 The SWEBOK Guide

The objectives of the SWEBOK Guide are to characterize the content of the software engineering discipline, to promote a consistent view of software engineering worldwide, to clarify the place, and set the boundary, of software engineering with respect to other disciplines, and to provide a foundation for curriculum development and individual licensing material. The SWEBOK Guide is a project of the IEEE

³ Version 2001A.04.00 of IBM RUP was used in this project.

⁴ See <http://www-136.ibm.com/developerworks/rational/products/rup/>

Computer Society and has the support of numerous organizations⁵. The 2004 Version is also published as ISO Technical Report 19759 [7].

In the next section, an example of how one specific role definition was analyzed and improved based on the SWEBOK Guide is presented. The paper will also describe how the set of roles was used during project planning and launch activities in order to meet the specific needs of a particular project. A conclusion and ideas for further work are presented in the final section.

2 Comparing the Role Definitions and the Reference Documents: An Example

In the course of this project, every role definition was individually analyzed against each of the three reference documents. An example of such an analysis for the

Table 2. Analysis of the Software Architect Role using the SWEBOK Guide as the reference document

Role Name : Software Architect		Presence of the Role : Accept			
GAP :	RT :	P :	CR :	HS :	SS :
Minor	Accept	Modify	Accept	Modify	Modify
BES SWE		SWEBOK		Note	
The Software Architect establishes the overall software architectural framework. Thus, in contrast with the other Roles (ex. Software Implementer), the Software Architect's view is one of breadth, as opposed to depth.		Chapter 3 : Software Design <i>Software Structure and Architecture</i> In its strictest sense, "a software architecture is a description of the subsystems and components of a software system and the relationships between them" ⁶ . An architecture thus attempts to define the internal structure – "the way in which something is constructed or organized" ⁷ – of the resulting software.		The role of the Software Architect as and defined in the BES SWE assumes the activities stipulated in subsection III, Software Structure and Architecture, of the Guide SWEBOK The SWEBOK is very useful for improving the hard skills needed for this role.	

⁵ Available free of charge on www.swebok.org and can also be purchased in book format from the IEEE Computer Society Press.

⁶ Quotation from Chapter 6 of F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad and M. Stal. *Pattern-oriented Software Architecture – A System of Patterns*, J. Wiley and Sons.

Software Architect role using the SWEBOK Guide as the reference document is found in Table 2. Such a table was completed for each role definition against each of the three reference documents. The detailed improvements on the role definitions are based on these tables. Table 2 is composed of the following cells:

- Role name: Name of the role in the BES SWE;
- OR: Overall recommendation on the presence of the role resulting from the analysis with the reference document (Accept, Remove);
- GAP: Overall evaluation of the difference between the definition of the role in the BES SWE and the definition of the role explicitly stated or implied in the reference document. Possible values are: Major, Minor, No Gap.
- RT: Recommendation regarding the role title (Accept, Modify);
- P: in Recommendation regarding the role purpose section (Accept, Modify);
- CR: Recommendation regarding the core responsibilities section (Accept, Modify);
- HS: Recommendation regarding the hard skills section of the role definition (Accept, Modify);
- SS: Recommendation regarding the soft skills section of the role definition (Accept, Modify).
- BES SWE: Excerpted text from the definition of the role in the BES SWE prior to improvement.
- SWEBOK: Excerpted text from the SWEBOK Guide relevant to this role definition and potentially useful for improving the definition of the role in the BES SWE.
- Note: Indications on how to improve the definition of the role based on this comparison.

The improved definition of the Software Architect role after the comparison with the three reference documents is found in Table 3 (proposed improvements are in italics).

Table 3. Definition of the Software Architect Role (proposed improvements are in italics)

Purpose:

The Software Architect establishes the overall software architecture. Thus, in contrast with the other Roles (e.g. Software Implementer), the Software Architect's view is one of breadth, as opposed to one of depth.

The Software Architect is responsible for articulating the architectural vision, conceptualizing and experimenting with alternative architectural approaches, creating models and components, interface specification documents, and validating the architecture against requirements and assumptions.

Activities in this area include the creation of technology roadmaps, making assertions about technology directions and determining their consequences for the technical strategy, and hence architectural approach. This role involves not just these technical activities, but others that are more political and strategic.

⁷ Quotation from the Oxford Dictionary.

Table 3. (continued)**Core Responsibilities:**

- Derive the requirements for the system and software architecture.
- Identify the key design issues that must be resolved to support successful development of the software.
- Generate one or more alternatives and constraints for the architecture and select a solution.
- Allocate the software and derived requirements to the chosen architecture components and interfaces.
- Maintain requirement traceability for the software architecture's requirements.
- Describe the software architecture by capturing the design results and rationale.
- Identify appropriate derived requirements that address the effectiveness and cost of life-cycle phases following development, such as production and operation.

Hard Skills:

- Ability to identify technical project risks based on the software architecture model.
- Ability to perform software modeling and architecture conception/definition.
- Ability to perform conceptual product design, *and specify a software architecture and implement a software system embodying it.*
- Ability to apply modeling techniques *such as use case, and other techniques, using the UML notations.*
- *Ability to apply Architectural Styles, Reference models and Reference Architectures.*
- *Ability to specify structural descriptions with techniques and notations such as: Architecture Description Language, Class Responsibility Card, Entity Relation Diagram, Interface Description Language, Jackson structure Diagrams and Structure Charts.*
- *Ability to specify behavioral descriptions with techniques and notations such as: Program Design Language PDL, Data Flow Diagram DFD and Flowcharts.*
- *Ability to use computer-aided software engineering (CASE) tools in an architecture-driven design process.*
- *Knowledge in concepts of structural patterns such as layers and client/server, mechanisms such as brokers and bridges, and middleware such as CORBA , DCOM, RMI.*
- *Knowledge in operating system architectures, compiler and interpreter design, and Real-time and Embedded Systems.*
- *Ability to perform software engineering activities across the full development cycle, including analysis, design, implementation, testing and documenting.*
- *Understand the business context of Bombardier Transportation and its competitors, their products, strategies and product generation processes.*

Table 3. (continued)

Soft Skills:	
○	Flexibility: The ability to adapt and deal with situations and manage expectations during periods of change.
○	Sound Business Judgment: Know the business purpose of a project and make decisions within that context.
○	Exhibit several communication styles: Be able to recognize a person’s communication style and adapt to it.
○	Active listening skills.
○	Setting and managing expectations.
○	Conflict resolution.
○	<i>Have the ability to make critical decisions under pressure.</i>

Table 4 and table 5 illustrate examples of how the consolidation and final decision of a few roles were formalized. For the software architect role, there were only minor gaps in both 12207 and SWEBOK. It was decided by Bombardier Transportation to keep the role and add the information gathered when performing the gap analysis.

Table 4. Examples of consolidation of analyses regarding the presence of the role⁸

Role name	RUP		IEEE 12207		SWEBOK	
	GAP	OR ⁹	GAP	OR	GAP	OR
Project Manager	N	A	m	A	m	A
Safety Representative¹⁰	N/A	N/A	M	A	N/A	N/A
Software Architect	N	A	m	A	m	A
Software Engineering Manager	M	R	M	R	N	A
Software Implementer	N	A	m	A	M	A

Table 5. Examples of consolidation of decisions regarding the presence of the role

Role name	Global recommendation of the study	Decision regarding the presence of the role by Bombardier Transportation	Rationale for the decision (when relevant)
Project Manager	A	A	
Safety Representative	A	A	Important role in the context of Bombardier

⁸ Legend: M = Major ; m = minor ; A = Accept ; R = remove ; N = none ; N/A = Non Applicable.

⁹ Overall recommendation.

¹⁰ Safety functions are not covered in the IBM-RUP and are outside the scope of the SWEBOK Guide.

Table 5. (continued)

Software Architect	A	A	
Software Engineering Manager	A	A	This role is implied in the CMM [8].
Software Implementer	Ap	A	

3 Utilization of the Set of Roles During Project Planning and Launch Activities

Members of a new project usually have many concerns. In particular, they will have concerns regarding the organization of the team such as: Who will be my team members? Who will be the team leader? What will be my role and responsibilities? What will be the team members' roles and responsibilities? Will my team members have the skill and knowledge to do the project? Will we have all the skills to do the project?

To address these concerns, an organization may hold a project launch. A project launch is a workshop, usually led by a facilitator, in which identified project team members either define the project plan, including activities, deliverables and schedule, or walkthrough an already defined project plan. The project launch workshop could last between one and three days, but, for a typical Bombardier Transportation project, a one-day session is normally enough.

In order to illustrate the utilization of a defined set of roles, an example of a typical project planning and project launch session will be described. The purposes of a project launch session at Bombardier Transportation and are to:

- Define the project plan using an integrated team approach;
- Ensure a common understanding of objectives, process, deliverables, and role and responsibilities (R&R) of all project team members;
- Provide for an information exchange and offer just-in-time training to the project team members.

Before we present the example, an overview of the software engineering process will be provided to give the reader a better understanding of the environment at Bombardier Transportation.

3.1 Overview of the BES SWE

The BES SWE was inspired by and partially derived from the Rational Unified Process (RUP)¹¹. Illustrated in Figure 2, it provides a disciplined approach to assigning tasks and responsibilities within a software development organization. Its goal is to ensure the production of high-quality software that meets the needs of its end-users within a predictable timeframe and budget. The BES SWE has two dimensions:

¹¹ Version 2001A.04.00 of RUP was used.

- The vertical axis represents processes, which are group of activities based on the IEEE 12207 standard. This dimension represents the static aspect of the process, that is, how it is described in terms of process items: processes/sub-processes, activities and artifacts.
- The horizontal axis represents time and shows the Life Cycle aspects of the process as it unfolds. This dimension represents the dynamic aspect of the process as it unfolds, and it is expressed in terms of Phases, Iterations, Milestones and Formal Baselines.

To follow up on the set of roles described above, the Software Architect will be primarily responsible for the sub process titled Software Architecture Design. As illustrated in Figure 2, the Software Architect will have to participate in formal reviews such as the Preliminary Design Review (PDR) and the Critical Design Review (CDR). This role will also either lead or participate in reviews such as Walkthrough and Inspection [9, 10].

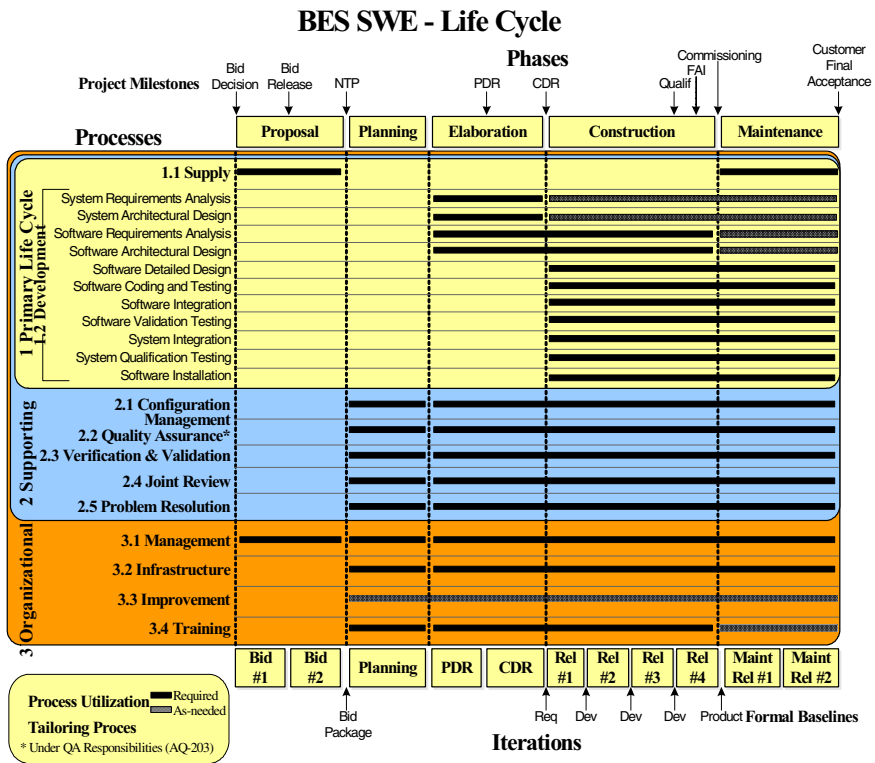


Fig. 2. BES SWE Life Cycle process¹²

¹² Legend: NTP: Notice to Proceed
 CDR: Critical Design Review
 FAI: First Article Inspection
 Dev: Development

PDR: Preliminary Design Review
 Qualif: System Qualification
 Req: Requirement
 Maint: Maintenance

3.2 Typical Agenda of a Project Launch Session

At Bombardier Transportation, a project launch session is typically performed at the beginning of a new project or at the beginning of an iteration. It can also be performed, for an iterative development project, to prepare the next iteration. In this case, it is called a project relaunch session. This intervention is also well suited in the

Table 6. Typical activities and timetable of a project planning and launch session (1 day)

TIME	AGENDA ITEMS
08h30	Welcome, agenda review and participants' expectations <ul style="list-style-type: none"> Logistics: time-keeper and recorder roles
09h00	BES Software Engineering Processes Overview
10h30	Software Project Management (SPM) process: <ol style="list-style-type: none"> Identify/review all project inputs documents/information Project scope, constraints and assumptions Project iterations and associated objectives (imposed Milestones) Project team structure and Role allocation Project architecture (high level – SCI List) Project tailoring and deliverables list (per iteration) [WBS] Project staffing needs Intergroup relationships and associated Roles and Responsibilities (context diagram + R&R table) Project risk identification and analysis
12h00	Lunch
13h00	Software Project Management (SPM) process (continue): <ol style="list-style-type: none"> Project tailoring and deliverables list (per iteration) Project risk identification and analysis
14h30	Break
14h45	Software Development (SD) process: <ol style="list-style-type: none"> Requirement definition: level, attributes Requirement traceability relationships
15h00	Software Configuration Management (SCM) process: <ol style="list-style-type: none"> Configuration Management (SCM) process: Project Software Configuration Identification (SCI) Project Baseline Plan (per iteration) Development of baseline approach (including tagging) SCM Audits, release management
15h45	Software Quality Assurance (SWQA) and Verification & Validation (SVV) processes: <ul style="list-style-type: none"> Identify SQA activities and associated R&R
16h00	Software Infrastructure and Training: <ol style="list-style-type: none"> Project Development environment Project Validation/testing environment Project System Qualification environment Project training needs
16h30	Session wrap-up
17h00	End

case where a project's performance and/or process needs to be improved, when a project needs recovery, for example.

Depending on the size, complexity and type of project (e.g. new or modified/reuse, safety critical, etc.), a typical project launch session 'meeting' will last for 1 or 2 days at the same location. During a project launch session, it is important to have the team members' time 100% dedicated to this activity. In order to reduce office disturbance (e.g. phone calls), the project launch session may be held outside the project team office or building. Table 6 illustrates a typical timetable for a one-day project launch session. As shown in the table, under the topic Software Project Management (SPM) process, roles and responsibilities (R&R) are first discussed in item 4 and then in item 8. R&R are also discussed under the topic Software Quality Assurance (SWQA) and Verification & Validation (SVV) processes.

In many locations in Bombardier Transportation, R&R is informally allocated, and, most of the time, there is no name associated with the set of activities performed by an individual. In a few locations, some roles are partially defined. But, the name of the role and its responsibilities varies from one location to another. It was essential to be able to deploy a common software engineering processes to embed an R&R set in the process. When a project launch was conducted, it was then only necessary to mention that the R&R had been developed using internationally recognized frameworks. Team members were then ready to proceed without arguments about the names and responsibilities of the roles in the defined R&R set. Additionally, common definitions of roles were essential when people from different sites had to work on the same project. It was very easy for the project manager to prepare his staffing plan, as we will show below.

3.3 Project Tailoring

During the project planning and launch session, one item for discussion is 'Tailor the Project'. The output from this tailoring activity will establish the project deliverables and the activities needed to develop the deliverables. Also, using the list of defined roles and responsibilities, roles for this project will be identified during the tailoring activity. As an example, if there are Safety regulations (e.g. CENELEC Standard EN50128¹³) imposed on a Project, 'Software Verifier', 'Software Validator' and 'Software Safety Assessor' roles will be identified. If, for the same project, software has to be acquired from a supplier, then a 'Software Acquisition Coordinator' role will also be needed.

3.4 Project Organization

Once the Project Manager has identified which roles are needed for his project, this person will then select, out of the pool of resources available, the people for whom the hard and soft skills have been identified. The project manager will again use the set

¹³ CENELEC EN50128 - Railway applications - Communications, signalling and processing systems - Software for railway control and protection systems.

of role definitions to select the individuals who have the required characteristics. Without a defined set of role definitions, it would be harder for project managers to perform this activity.

The project manager, having identified the individuals, can now fill in the Project Organization Table, as illustrated in Table 7. If a few roles of a local development center have already been defined, then, at this stage, the local role names will be used for the local project plan.

Table 7. Subset of a Project Organization Table

Name of team member	Local Title	Relationship*	SW Project Manager	SW Task Leader	SW Req. Coordinator	SW Architect	SW Implementer	SW Integrator	SW Tester	SCCB	SW Infrastructure Admin	SW Process Engineer
Greg	...	C							?			
Per	Software Engineer	I		X	X	X	X	X	X			
Michael	Software Engineer	I					X					
Annie	Software Tester	I										
Guy	Manager, QA and Test	E										
Pierre	Tech	E										

Notes about this table:

- * Relationship → C: Customer representative. E: External resource. I: Internal resource

This table contains only the list of Roles identified for a specific project after tailoring. The Project Manager has to verify that at least one person has been identified per Project Role. (i.e. each column needs to have at least one 'X'). The Project Organization Table (Table 7) is also used for the completion of the Project Staffing plan. As illustrated in Table 8, the numbers in the cells represent the required head count per Role for a given period within a Project Iteration.

Table 8. Example of a subset of a Project Staffing Plan

Roles	Iteration #1 Requirement		Iteration #2 Release 1		Iteration #3 Release 2	
	Months 1 -3	Months 4 -6	Months 7 -10	Months 11 -16	Months 17 -20	Months 21 -23
Software Project Manager	0.3	0.3	0.3	0.3	0.3	0.3
Software Task Leader	1	1	1	1	1	1
Software Architect	1	.8	.8	.5	.8	.5
Software Implementer – Database			1	1	1	1
Software Implementer –GUI			1	2	2	1

3.5 Training Plan

If individuals identified in the Project Staffing Plan do not have the required skills and knowledge, a training matrix is generated (see Table 9) using, again, the information obtained from the set of role definitions. The training matrix will be used to develop the training plan for the project. Once the topic of the course has been identified, the following information is entered in the table:

- In the “type” column, an indication as to whether the training session is a formal course, self-training, lectures, from an external organization, etc.
- In the “iteration” column, an indication as to when the training will take place.
- In the “duration” column, an indication as to the length, in hours or days, of the training session.

Table 9. Example of a subset of a Training Plan Matrix

Training Topics	Iteration	Duration [day]	Management	SW Project Manager	SW Task Leader	SW Req. Coordinator	SW Architect	SW Implementer	SW Integrator	SW Tester	SW Process Engineer	SW QA Engineer
UML Modeling	R2	2d				X	X		X	X		
Peer Review	R1	1d			X	X	X	X	X	X	X	X
Estimation	R1	2d	X	X	X							

Once the project is completed, a lessons-learned session is held to identify the project's strengths and weaknesses. One of the issues analyzed is human resources (i.e. role allocation, skill sets, training). The output of the lessons learned exercise is used to update the set of role and responsibility definitions and also to modify the software engineering processes.

4 Conclusion

A project was conducted to improve the software engineering role definitions within the software engineering process definition of a large multinational organization. Detailed improvements were proposed to all role definitions, and an illustration of these improvements was presented for the Software Architect role.

It was also demonstrated how, at Bombardier Transportation, software projects tailor the set of roles to meet specific project needs. The utilization of the list of roles was illustrated for a project planning and launch session. It was demonstrated how roles and responsibilities were selected and allocated. Also, it was shown how the list was used to prepare a staffing plan and a training plan. The set of roles has been used in six project launch sessions. This has helped Bombardier Transport show to their customers that a well defined set of software roles is implemented in their projects. Customers are also very satisfied because the role set were developed using documents such as ISO and IEEE Standards.

The use of standards has also reduced significantly discussions during the deployment of processes; since employees and managers know the value and the credibility of standards.

To deploy a common software engineering process in many sites around the world, it is essential to embed a common set of roles and responsibilities in the process. When a project launch is conducted, it is much easier to deploy the set of roles by demonstrating that these roles have been developed using three internationally recognized frameworks. Individuals are then ready to proceed without arguing about the names of the roles or responsibilities assigned to, and the skills required by, each role. Additionally, the common set of roles is essential when people from different sites have to work on the same project. It is easier for the project manager to prepare his project plan and his training plan.

As more remote Bombardier Transportation sites are putting their effort together to develop and integrate software components, it is critical to have a common set of roles and responsibilities. Bombardier Transportation is among a minority of organizations that have documented these roles, as stated in a recent book [11]. The author indicated that fewer than 65% of the organizations that he assessed had documents describing roles and responsibilities.

As a result of the utilization of the role set, we have responded to the typical team members' concerns, such as: Who will be my team members? Who will be the team leader? What will be my role and responsibilities? What will be the team members' roles and responsibilities? Will my team members have the skill and knowledge to carry out the project? Will we have all the skills to carry out the project?

References

1. Humphrey, W.: Introduction to the Team Software Process, p. 24. Addison Wesley, London (2000)
2. Belkebir, Y.: Analyse et amélioration des définitions de rôles du processus d'ingénierie logicielle du centre de compétence en génie logiciel de Bombardier Transport, Department of Software and IT Engineering, École de technologie supérieure, Montréal (2003)
3. Laporte, C.Y., Bourque, P., Belkebir, Y., Doucet, M.: Amélioration de la définition des rôles du processus de génie logiciel de la société Bombardier Transport. *Revue Génie Logiciel* 72, 43–52 (2005)
4. IEEE/EIA 12207.0-1996 IEEE/EIA Standard Industry Implementation of International Standard ISO/IEC 12207: 1995 (ISO/IEC 12207) Standard for Information Technology Software Life Cycle Processes, Institute of Electrical and Electronics Engineers (1998)
5. Kruchten, P.: The Rational Unified Process: An Introduction. Addison-Wesley, London (2003)
6. Abran, A., Moore, J.W., Bourque, P., Dupuis, R. (eds.): Guide to the Software Engineering Body of Knowledge. IEEE Computer Society Press, Los Alamitos (2004)
7. International Organization for Standardization. Software Engineering Body of Knowledge, Technical Report ISO/IEC TR 19759 (2005)
8. Paulk, M., Curtis, B., Chrissis, M.B., Weber, C.V.: Capability maturity model for software version 1.1. Software Engineering Institute, CMU-SEI-93-TR-24 (1993)
9. Gilb, T., Graham, D.: Software inspection. Addison-Wesley, Wokingham, U.K (1993)
10. IEEE 1028-2002 Software Reviews, Institute of Electrical and Electronics Engineers (2002)
11. Poulin, L.: Reducing Risk with Software Process Improvement, Auerbach Publications (2005)

Software Verification Process Improvement Proposal Using Six Sigma

Tihana Galinac¹ and Željka Car²

¹ Ericsson Nikola Tesla, Research and Development Center, Krapinska 45, HR-10000
Zagreb, Croatia

tihana.galinac@ericsson.com

² University of Zagreb, Faculty of Electrical Engineering and Computing, Unska 3,
HR-10000 Zagreb, Croatia

zeljka.car@fer.hr

Abstract. In the rapidly growing modern telecommunications industry, software quality and reliability requirements are in contrast to the shorter time to market and higher complexity requirements dictated by strong competition on the telecommunications market. In such a rapidly changing environment, software development organization must improve almost on a daily basis in order to achieve the operational excellence which leads to business success. In this paper, an approach to the continuous improvement of the software verification process based on the application of Six Sigma is given. More precisely, with the help of the Six Sigma methodology, change management, and statistical tools and techniques, the proposed approach solves the problem of fault slippage through verification phases, which is particularly important in overlapping project conditions. Success of the proposed process improvement, proved using Six Sigma methodologies for a case study from a real industrial project, encourages wide and general application to any software verification process.

1 Introduction

Significant research effort in software development during the last two decades has been devoted to Software Process Improvement (SPI) [15]. It is mostly triggered from industry [2]. The main problems faced by industry include rapid the software development and shorter time to market requirements caused by strong competition and new technologies. Consequently, final products are often of lower quality and development and verification expenses are greater. This effect is especially emphasized in telecommunications software development, due to its specific nature, as will be explained in Sect. 2.

Since software quality is measured with the number of faults, where fewer faults imply significant savings in rework time and cost, SPI is mostly concerned with the verification part of the software development process [14], [13], [1], [4], [8]. This includes not only better fault detection, but also better fault prevention.

One of the main goals of software verification improvement is reducing fault slippage through verification phases. It is also the main problem addressed in this paper.

There are several SPI approaches. The International Standards Organization (ISO) has established several standards for management of software quality [11], [12]. Their quality assurance procedure is based on reviews and audits measuring the ability of developers to fulfill the work assignments initially specified. The Capability Maturity Model (CMM) introduces continual improvement on the process management [6], [7]. The idea behind CMM is that independent assessments can be used to grade organizations based on how well they create software according to their definition and execution of their processes.

Nowadays, the importance of statistical methods and mechanisms is growing rapidly [5]. They are becoming very promising candidates for improving process control by reaching higher production efficiency and maturity levels, as well as early indicators of the need for changes in processes. However, due to the complexity of organization, Quality Management goes far beyond only statistics. Having that in mind, the Six Sigma approach [3] is a set of change management skills, and quality and statistical knowledge, based on the Define, Measure, Analyze, Improve, Control (DMAIC) methodology. One of the most important aspects of the Six Sigma approach is its statistical base and statistical process control which forms its integral part.

Originally one of the most successful approaches used in hardware production processes [15], Six Sigma is becoming more and more interesting to the software industry. However, as explained in [2], it is still unclear how to fully apply the concept of Six Sigma to software development. In this paper, we present an improvement proposal and show its successful deployment and further control, obtained as an outcome of the Six Sigma project. The precise research framework is given in Sect. 2.

The paper is organized as follows. In Sect. 2, we give an overview of the software verification process by analyzing the problem and establishing a research framework. Section 3 explains the problem in more detail and provides a cost analysis that was used before improvement project initiation to encourage investment. Furthermore, it deals with project execution, which includes defining, measuring and analyzing the problem, through logically elaborated steps leading to the final solution. Finally, Sect. 4 presents an improvement proposal, describes achieved benefits, provides a strategy for future process control and gives guidelines for implementing improvements for the general case.

2 Research Framework

The case study of this paper was performed for an industrial project within a development unit at Ericsson which aims to achieve business excellence through continuous improvement programs. It deals with developing large scale software for telephone exchanges forming network solutions for next generation networks. The development unit is a multinational organization consisting of four

dislocated design centers. A typical software development project, such as the one studied in this paper, usually lasts 1 – 1.5 years and involves, on average, 300 engineers.

Since telecommunications software mostly provides real-time services for end users one of the most important quality requirements when developing software for telecommunications equipment, is reliability. On the other hand, due to the rapidly evolving service functions offered to end users and the increasing number of end users, the requirements on functionality are becoming more complex requiring more processing capacity. Due to the complexity of these kinds of products, their lifecycle is treated as separate versions of the same software product line.

Moreover, in order to satisfy the contradictory requirements of high reliability and frequent function delivery, the process of software product development is an in-house developed model belonging to the incremental development model class with waterfall increments [17]. As presented in Fig. 1, waterfall increments consist of the following stages: network design, system design, function design, software unit design and test, function test, system test and network test. The process decomposition closely follows the hierarchy of the developed network. A network solution consists of a number of physically separated network nodes which communicate with each other via standardized network protocols. Network nodes represent telephone exchanges consisting of hardware and software parts, where the software part is a large-scale software package consisting of a number of functions. Functions are implemented within logically separated software structures and consist of several Software Units (SWU).

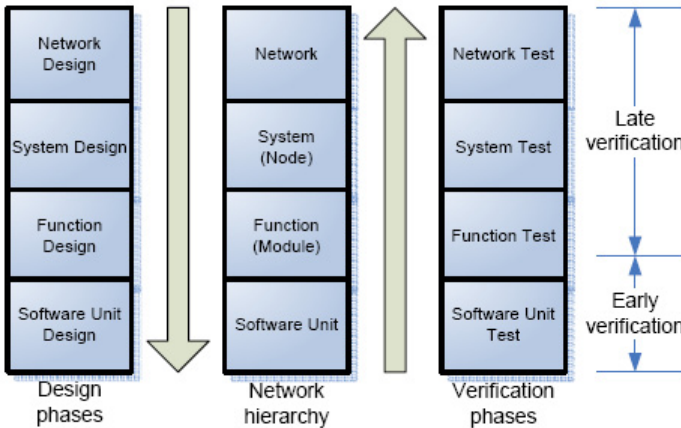


Fig. 1. Network hierarchy, design and verification phases

In the rest of the paper, we will refer to Early Verification (EV) and Late Verification (LV) as shown in Fig. 1. Early verification activities are all verification activities performed within a single design team. Thus, it consists of a

software unit test and part of a function test. All other verification activities are considered late verification, since multiple design teams are involved and a higher level of formalism is required. The reason for differentiating the two lies in the big jump in fault removal expenses between early and late verification.

The incremental model which divides of the project into a certain number of increments, allows different project phases, and even different projects on the same product line, to overlap. This means that, although one or more development increments/projects on the software product line may not be finished, new ones from the same software base need to commence in order to keep their position on the market and supply customers as expected.

The problem studied in this paper is the continuous increase in faults that slip through from EV phases into LV phases. One of the reasons for this lies in the fast development process applied, which uses a product line approach where software is increasingly complicated from revision to revision and inherits an increasing number of faults. However, besides the unavoidable increase in the number of faults detected in a single product, in overlapping conditions, tremendous work effort is needed to map these faults to its predecessor and successor products. In the following sections, we present a Six Sigma-based approach to the continuous adaptation and change of the verification process, thus reducing the consequences of project overlapping and higher product complexity in order to reach the customer expectations with minimal cost.

3 Implementation of the Six Sigma Project

In the rest of the paper, we follow the Six Sigma methodology described in [3], to obtain verification process improvements solving the fault slip through problem for our case study. The Six Sigma methodology used is DMAIC (define, measure, analyze, improve and control phase) and our description is structured accordingly.

3.1 Problem Definition

The Six Sigma improvement project is issued like every regular project with assignment specification. It starts with a lot of meetings and conferences aimed at gathering information needed to define the problem, identify customer needs, and set problem goals and success criteria. Using this information, a Six Sigma project charter document is prepared, approved and signed by the sponsor. This document contains the problem/opportunity statement, the goal of the project, the business case, the defined project steering group, the baseline and target projects and the project plan. The duration of the Six Sigma project in our case study was six months. The project team was composed of representative members from four design centers participating in the development process of the target project. Each member was responsible for locally driving and coordinating Six Sigma activities in his own organization, with the help of a team borrowed from the development target project if needed.

As explained in the research framework, the Six Sigma improvement project in this case study was established with the goal of reducing fault slip through, i.e. the number of faults detected during late verification. However, the Six Sigma methodology requires goal definition to include a precise statement indicating the expected benefit of the improvements. To verify goal fulfillment and improvement success, we defined two success indicators. The first was an early control indicator, referred to as early fault density *EFD*, which was used as control indicator within the early verification process and was defined as

$$EFD = \frac{\text{total number of faults in EV}}{\text{total modified volume}} .$$

The second control indicator, referred to as fault slip through *FST*, was used for goal verification at the end of the late verification process and was defined as

$$FST = \frac{\text{total number of faults in LV}}{\text{total number of faults in the project}} .$$

Based on our project goal, expressed as the percentage of faults which are detected late, i.e. should have been found earlier, we established the precise values of the success indicators that ensure goal fulfilment. When comparing the indicators for the baseline and target projects, we used a hypothesis t-test of equal means (Sect. 1.3.5.3 of [9]) with a null hypothesis

H_0 : Success indicators for sample of baseline and target project are equal.

Equality of means would imply failure of improvement. In other words, success of our improvement proposal is proved if the results imply rejection of the null hypothesis.

The Six Sigma project skeleton was adapted to the project management model while control of project progress was monitored through Project Steering Group (PSG) meetings. The sponsor of the project is also the chair of the PSG meetings and is responsible for the results achieved by the target project. The stakeholders of the project are mainly responsible for line and project management, as identified in the stakeholder map document, and basically form the PSG.

The define phase of our Six Sigma project started with several brainstorm sessions with experts participating in the design and test parts of the process. First, the Supplier, Input, Process, Output and Customer (SIPOC) diagram was used to document the process at a high level and visually show the process with respect to suppliers, inputs to the process, or services received by customers.

The next step was to identify in more detail the process map 'as is'. The flow of events in the process was identified, as well as inputs and outputs in every step. Critical process steps for final product quality were identified from the complicated and detailed process map. Summarized conclusions were put together in a Critical to Quality (CTQ) diagram, as depicted in Fig. 2. To determine which product and/or process requires the most attention, an evaluation of CTQ branches, based on their effect, feasibility and priority was performed. Using these evaluation criteria, the focus of the Six Sigma project was narrowed down to the planning and control processes within early verification.

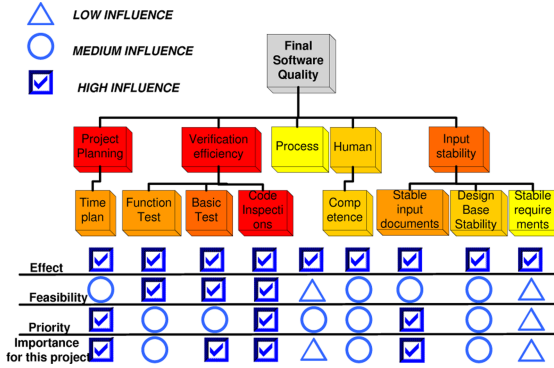


Fig. 2. Critical to quality tree

3.2 Data Collection and Analysis

The measure phase involved designing data collection plans, collecting data and then using data to describe the process. Based on the detailed process map 'as is' the following basic and derived variables were selected for statistical process measurement:

- modified volume V_{mod} ,
- number of faults reported in early verification F_{EV} ,
- number of faults reported in late verification F_{LV} ,
- effort spent on early verification E_{EV} ,
- fault density in early verification $FD_{\text{EV}} = \frac{F_{\text{EV}}}{V_{\text{mod}}}$,
- fault density in late verification $FD_{\text{LV}} = \frac{F_{\text{LV}}}{V_{\text{mod}}}$,
- effort density in early verification $ED_{\text{EV}} = \frac{E_{\text{EV}}}{V_{\text{mod}}}$.

All these variables were measured per software unit. The measurements were performed on the baseline project and collected from different design centers operating in four different countries. The measured sample of 200 SWU was large enough to draw conclusions with statistical significance. The measurements performed were the basis for quality analysis and solution generation. Once the data was collected, classified and summarized the analyze phase began which included graphical representations of all the measured variables per software unit. All statistical techniques used in this paper are well known. Thus, instead of recalling them here, we give precise and practical references from [9] or [10].

For all measured samples, an Anderson–Darling normality test (Sect. 7.2.1.3 of [9]) was performed to see whether the sample of data come from a population with a normal distribution. The results showed that none of the measured data samples were normally distributed. Since most statistical techniques require normally distributed data, we performed transformations on the measured data to obtain normal distributions. Using the Box–Cox transformation, for optimal transformation parameter λ (Sect. 6.5.2 of [9]), we found that all of the measured

samples could be transformed to follow normal distributions using $\lambda = 0$ with maximum correlation on the Box–Cox plot. In other words, in order to transform our data into normality, we simply needed to take the natural logarithm. After performing the Box–Cox transformation, i.e. taking the natural logarithm of the data, the Anderson–Darling normality test was rerun and showed that the transformed data indeed followed a normal distribution with a confidence level of 95%, while the residuals were normally distributed. Now, having all the data following normal distributions, we could continue to apply classical statistical techniques. All the graphs below show relations between the transformed data.

In order to verify the reliability of the data collected, we performed several hypothesis tests. We performed a Bartlett Test (Sects. 1.3.5.7 of [9]) and an Analysis of Variance (ANOVA, Sect. 7.4.3 of [9]) between the various samples collected from geographically distributed design centers developing parts of the same software. This was done in order to verify our assumption that data came from the same population. The Bartlett Test is a test for equality of variances across groups, while ANOVA is an additional test for the equality of means. The conclusion is that we can assume, with 95% confidence, that all the transformed samples have the same variance and mean. Therefore, they come from the same population and thus, the data is indeed reliable.

Next, we used correlation to determine the strength of linear relationships between pairs of process variables. We calculated the Pearson product moment correlation coefficient (Sect. 7.1 of [10]) for every possible pair of variables. In Table 1, only correlation coefficients for those pairs having significant correlation are displayed. From the correlation table, we drew three important conclusions for further analysis. Variables that highly correlate with the number of faults F_{LV} detected in late verification were the number of faults F_{EV} found in early verification and the modified volume V_{mod} . Nevertheless, correlation with the effort E_{EV} spent on early verification could not be neglected. Another important finding was that the number of faults detected in early verification was highly correlated to the effort spent on fault finding activities and the modified volume. This conclusion could be used to help plan early verification activities. Effort spent on early verification was correlated to the modified volume of code because, in the classical process, modified volume and expert judgment of complexity are the only main inputs for planning the verification effort.

Next, we used linear regression analysis (Sect. 7.2 of [10] or Sect. 4.1.4.1 of [9]) to find a linear model for the response variables, related to the number of faults detected in late verification, and the predictor variables, related to early verification. As shown in Fig. 3, the main result was that the highest

Table 1. Correlation table

	F_{EV}	E_{EV}	V_{mod}
E_{EV}	0.501		
V_{mod}	0.522	0.394	
F_{LV}	0.642	0.410	0.624

contribution (41.2%) to explanation of variation in response variable FD_{LV} was achieved by predictor FD_{EV} . The p -value in the ANOVA table (Sect. 7.4.3 of [9]) corresponding to that case indicates that the relation is statistically significant with 95% confidence. The R^2 value shows that the obtained linear model explains 41.2% of the variance of FD_{LV} .

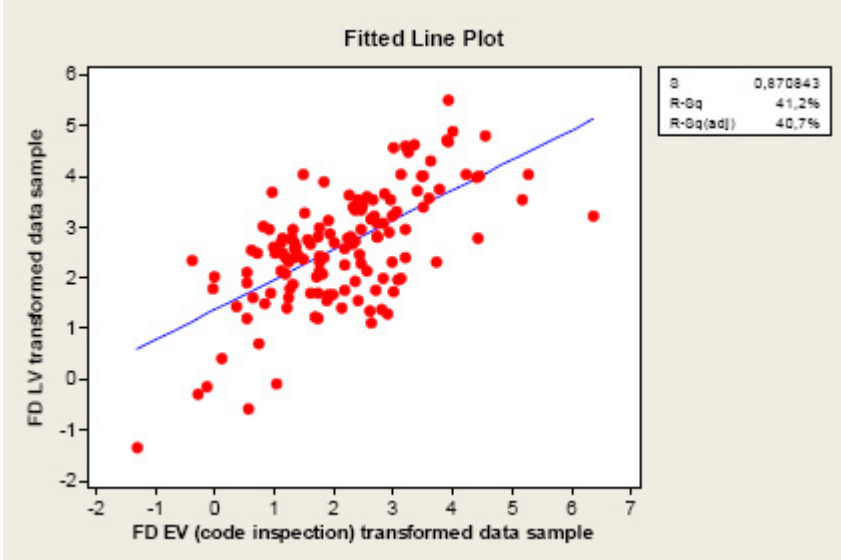


Fig. 3. Linear regression of fault density in EV and in LV

In perfect software production there would be no faults whatsoever and all the points in Fig. 3 would tend towards the left bottom corner. Since ideal software production is impossible to achieve, the most we can expect from our verification process is to move the points in that direction as much as possible, or at least make the slope of the fitted line as small as possible. That would mean that the FST indicator, i.e. the ratio between late and early detected faults, would change in our favor.

The next step of our case study analysis was to dig more deeply into the early verification process and find the reasons for the huge variation of effort density and fault density among different software units. We constructed a three dimensional scatter plot shown in Fig. 4. It reveals relationships and/or associations between three variables. The axes represent FD_{EV} , ED_{EV} and FD_{LV} . These three parameters were measured per software unit, where each point in Fig. 4 below represents one software unit.

In the scatter plot, we identified several groups of software units. For SWUs in group 1 in Fig. 4 huge effort was spent per modified volume but not many faults were found, in either early verification or in late verification. Group 2 show that no effort or minor effort, was spent per modified volume. As a result,

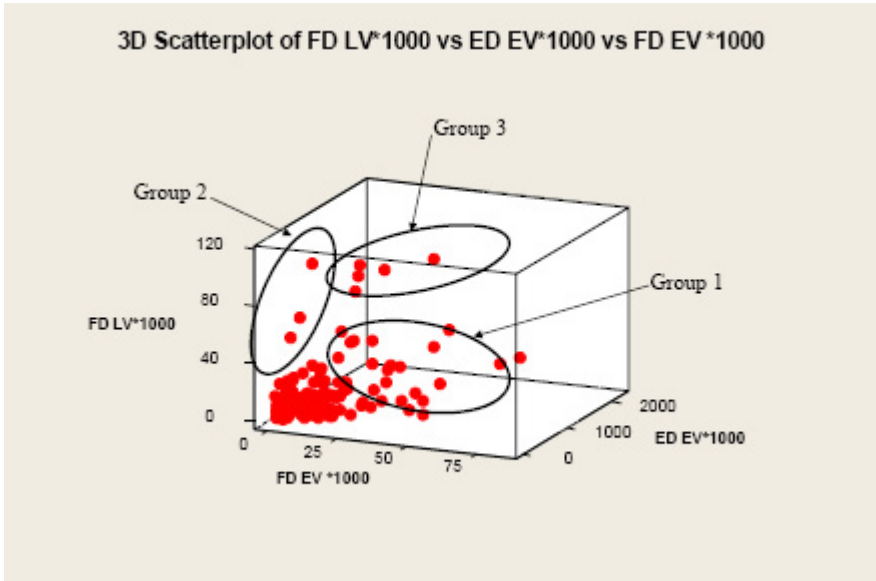


Fig. 4. Scatter plot of fault and effort density in EV and fault density in LV

not many faults were found in early verification, but many faults were found in late verification. For Group 3, a huge fault density was observed in the late verification phase, but with minor fault density in early verification. Scatter plots are very useful diagnostic tool for determining the associations of data. However, the plot rarely suggests and never proves an underlying cause and effect mechanism. It is only the researcher who can conclude that causality exists. In our case study, a team of engineers participating in the analyzing process drew the following conclusions based on the results of the analysis:

- inefficient planning of early verification activities,
- lack of control procedures in early verification,
- faults slipped through from the previous/overlapping project,
- lack of target verification concentrating on critical functions,
- lack of coordination of all verification activities (including early verification, such as code inspections and basic tests).

4 Software Verification Process Improvement Proposal

In this section, we propose improvements in the early verification process, give the results obtained for the case study target project, and provide guidelines for implementing improvements in similar processes.

4.1 Improvement Proposal

Based on the conclusions drawn from the analysis phase listed at the end of the previous section, we propose improvements for the improve phase of the Six Sigma project. The improvements are circled in Fig. 5.

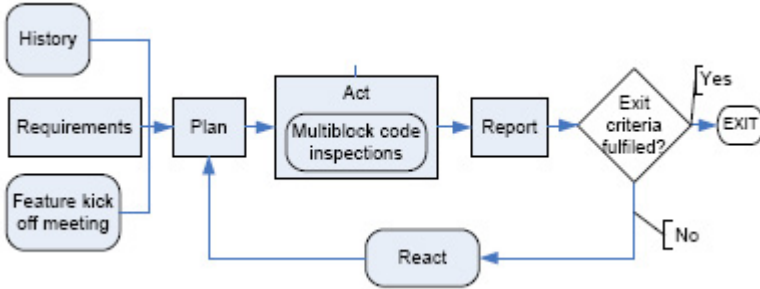


Fig. 5. Early verification process improvement proposal

First, we suggest a so-called feature kick-off meeting at the beginning of the verification process. Its purpose is to align the scope of all verification activities, since various parts of the verification process are performed by different verification teams. The need for this meeting was identified after several brainstorm sessions performed within verification teams in different verification phases. The main motivation was the fact that different verification phases focused on the same high priority checks, but often, skipped secondary checks due to lack of time, even though secondary checks often find several faults which can be very dangerous if not verified. The kick-off meeting investigates the requirement coverage matrix per planned test case, in the target project and analyzes the requirement coverage achieved in the baseline project. Furthermore, the best practices identified in previous projects are noted. The outcome of kick-off meeting is documented with a verification strategy that covers all the verification stages and sets the requirement coverage.

Continuously adding new and enhanced features to existing software tends to increase software unit complexity. Furthermore, software units already known to be unstable, due to the number of faults detected in previous and overlapping projects, can require more attention than others. Therefore, planning the required level of effort and competence needs has to be based on history records, measurements and knowledge regarding SWU behavior.

After analyzing the efficiency of different verification activities, we realized that early verification is more efficient in overlapping project conditions than non-overlapping ones. The fault removal cost increases with verification progress and the number of overlapping projects due to mapping activities. Consequently, early verification efficiency increases. The importance of early software (code) inspections was already emphasized in [13]. The classical approach to early verification focuses on just one specified SWU at a time, checking coding rules and

logic. Our suggestion is to introduce multiblock code inspection which means checking one feature at a time implemented within several SWUs and then reviewing their harmonic behavior over the corresponding interfaces. In order to ensure successful multiblock code inspection, a special check list was prepared which includes a list of the most critical issues identified in the past which are easily solved using this technique. Of course, a history database still exists containing all the problems reported in the past. However, the main benefit of the proposed check list is to select and group the most critical ones regarding the cost spent on their correction as function of the complexity of their repair.

Furthermore, a reaction mechanism in early verification is introduced. It is an algorithm based on early verification input variables F_{EV} , E_{EV} and V_{mod} . The output of the algorithm is the amount of additional effort, possibly zero, which should be put into early verification activities. Finally, all the measured data, during both classical and improved activities, is collected and stored in well-defined databases which serve as the perfect input for quality criteria and statistical process control.

4.2 Improvement Benefits in the Target Project

The same variables were measured in the target project as in the baseline project. Furthermore, the EFD and FST indicators, defined during problem definition, were calculated for both the baseline and target projects and are shown in Table 2. When comparing the indicators, it is necessary that the total modified volume of the whole target project be large enough to secure statistical significance of the conclusions. We used power and sample size techniques (Sect. 7.2.2.2 of [9]) to find the smallest percentage of increase of the EFD indicator and decrease of the FST indicator, detectable with 95% confidence for the size of the target project in our case study. These percentages are also given in Table 2. It is important to mention that our data was transformed to follow a normal distribution in order to apply power and sample size statistical technique. Consequently, the mean of the transformed data, when transformed back into its original scale, is not equal to the mean of the original data.

Table 2. Comparison of EFD and FST indicators for baseline and target projects

Indicator	Min. detectable change	Change	Saving/unit of mod. vol.
EFD	41%	214%	25.97 Eur
FST	3.3%	59%	10.24 Eur

Besides the increase of EFD and decrease of FST , in Table 2, also shows the savings per unit of modified volume in the target project. These savings figures were obtained after subtracting all the expenses of our Six Sigma improvement project and all the additional costs of applying the proposed improvements. Observe that the savings predicted during early verification based on the EFD

indicator, are greater than the actual savings of the target project. The reason for this lies in the improvements applied to early verification, making the EFD indicator extremely high. Nevertheless, we feel that 10.24 Euro savings per unit of modified volume in the target project of our case study is a significant improvement.

4.3 Guidelines for Implementing Improvements in General

The improvement proposal given in this paper, which proved successful for the case study, can be applied to any verification process in software development. More importantly, the Six Sigma approach to continuous control and improvement of processes in general, as explained in [3] can also be applied. Methods and tools of Six Sigma used during our case study are given in Fig. 6.

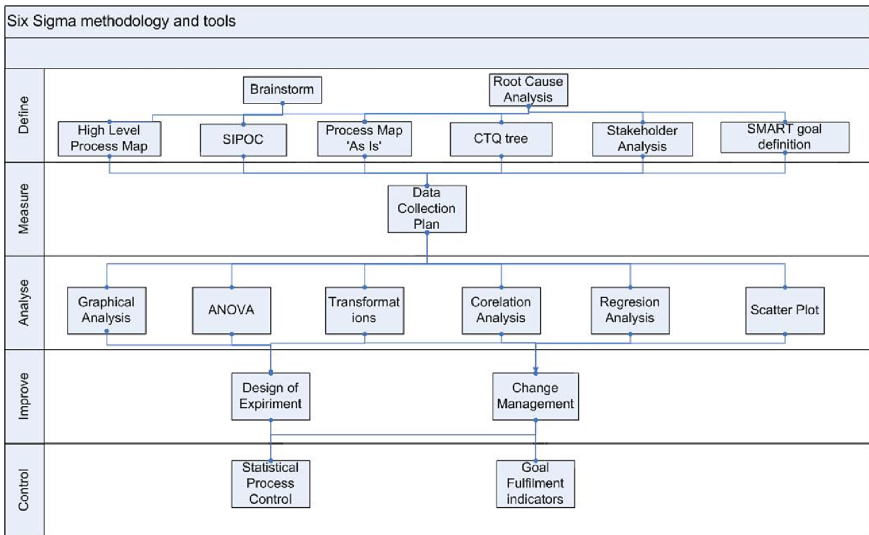


Fig. 6. Six Sigma methodology

The Six Sigma methodology is a powerful strategy for improving production processes in general. Historically, it was primarily applied to hardware production processes where process measures are naturally and uniquely defined. Here we present a case study applying Six Sigma to a large-scale software development process, where it was necessary to carefully choose variables measuring the process. The strength of Six Sigma lies in its strict sequence of steps, its variety of different statistical techniques, its change management and other tools (from brainstorming to regression analysis) and its data driven conclusions.

5 Conclusion

This paper proposes a Six Sigma based approach to improve the development process of large-scale software in rapidly changing environmental conditions with unavoidable project overlapping. Classical planning and management approaches based on traditional empirical data are insufficient. However, we found the Six Sigma approach, having internal continuous control, to be very appropriate for such dynamic conditions.

This paper deals with the verification process of incremental software development models with waterfall increments. More precisely, the main goal was to reduce fault slippage from early to late verification. This is an important issue since late verification fault removal expenses in overlapping project conditions are much higher than those in early verification.

In the paper, new improvements for the early verification part of the verification process are proposed. They proved to be very successful in a case study of a real industrial project executed in a big organization developing large-scale software. Since most big companies use a variant of the waterfall software development model, guidelines for implementing similar Six Sigma-based improvements for the general case are given. The improvement strategy, and the new improvements themselves, are widely applicable in the software industry. Therefore, they are a large step towards business success and operational excellence.

References

1. Arul, K.: Six Sigma for Software Application of Hypothesis Tests to Software Data. *Software Quality Journal* 12, 29–42 (2004)
2. Biehl, R.E.: Six Sigma for Software. *IEEE Software* 21, 68–70 (2004)
3. Breyfogle, F.W.: *Implementing Six Sigma*. John Wiley & Sons, Hoboken (2003)
4. Cangussu, J.W., DeCarlo, R.A., Mathur, A.P.: Monitoring the Software Test Process Using Statistical Process Control: A Logarithmic Approach. *ACM SIGSOFT Software Engineering Notes* 28, 158–167 (2003)
5. Card, D.N.: Statistical Techniques for Software Engineering Practice. In: *Proceedings of 26th International Conference on Software Engineering*, pp. 722–723 (2004)
6. CMMI Product Team: *Capability Maturity Model Integration (CMMI), v1.1 Continuous Representation*. CMU/SEI-2002-TR-003, Software Engineering Institute, Pittsburgh (December 2001)
7. CMMI Product Team: *Capability Maturity Model Integration (CMMI), v1.1, Staged Representation*. CMU/SEI-2002-TR-004, Software Engineering Institute, Pittsburgh (December 2001)
8. Damm, L.-O., Lundberg, L.: Using Fault Slippage Measurement for Monitoring Software Process Quality during Development. In: *ACM Proceedings of the 2006 International Workshop on Software Quality*, pp. 15–20 (2006)
9. *Engineering Statistics Handbook*, <http://www.itl.nist.gov/div898/handbook/>
10. Hoel, P.G.: *Introduction to Mathematical Statistics*, 3rd edn. John Wiley & Sons, New York (1962)
11. International Organization for Standardization: *Quality Management Systems – Guidelines for Performance Improvements*, ISO 9004:2000, ISO publication (December 2000)

12. International Organization for Standardization: Quality Management Systems – Requirements. ISO 9001:2000, ISO publication (December 2000)
13. Kollanus, S., Koskinen, J.: Software Inspections in Practice: Six Case Studies. In: Münch, J., Vierimaa, M. (eds.) PROFES 2006. LNCS, vol. 4034, pp. 377–382. Springer, Heidelberg (2006)
14. Leszak, M.: Software Defect Analysis of a Multi-release Telecommunications System. In: Bomarius, F., Komi-Sirviö, S. (eds.) PROFES 2005. LNCS, vol. 3547, pp. 98–114. Springer, Heidelberg (2005)
15. Serrano, M.A.: State of the Art and Future of Research in Software Process Improvement. In: Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC'04), vol. 01, p. 239 (2004)
16. Shao, J.: Mathematical Statistics. Springer Texts in Statistics. Springer, New York (1998)
17. Vliet, H.v.: Software Engineering, Principles and Practice, 2nd edn. John Wiley & Sons, Chichester (2000)

Software Development Improvement with SFIM

René Krikhaar^{1,2} and Martin Mermans³

¹ Vrije Universiteit Amsterdam, The Netherlands

² ICT NoviQ, The Netherlands

³ Philips Medical Systems, The Netherlands

Rene.Krikhaar@ict.nl, Martin.Mermans@philips.com

Abstract. Most industries are challenging to increase productivity of software development. Often many process improvement activities are started with enthusiasm, unfortunately most of these are less successful than forecasted or improvements do not sustain for long. This paper presents the Seven Forces Improvement Method, SFIM, which claims to overcome unexpected disappointment in improvement results. SFIM is built upon different aspects that influence the success of software process improvements, such as culture, skills and organization. The method has been applied to improvement activities in a large software department for a number of years. The success of SFIM is compared with the compliance with the SFIM method. The paper shows that application of SFIM increases the success rate of software improvement activities in industry.

Keywords: Software Process Improvement, 7S model, Force Field Analysis, CMMI.

1 Introduction

Healthy organizations are continuously looking for ways to better serve the customer and improve their business in a never-ending cycle. Good is never good enough and what is good today may become unacceptable tomorrow because of the ever changing environment in which each organization operates. New technologies, merges with other companies, changing customer demands, employees with fresh ideas, there are numerous triggers for changing.

During more than a decade, most organizations use the Capability Maturity Model (CMM) [Hum89] to control and measure software improvements. CMM provides a framework consisting of Key Process Areas (KPA's) with a kind of recipe in which order the KPA's have to be developed. For some reason this model does not work properly in each organization. CMM mainly focuses on process, while other aspects play a role as well, e.g. the factors that play a role during realization. When changing an organization to establish an improvement, many more factors are influencing the success. In general, it takes a lot of effort and energy of many people to achieve a change.

A feeling of discomfort is rising in the software world. Only with great effort and difficulty organizations move towards higher maturity levels. Most of them never

reach higher levels or when achieving a high level they have a large probability to fall back to a lower level. After 15 years of CMM only 18% of the organizations that report their CMM statuses to the SEI are at CMM level 4 or 5 [PMF05]. On the other hand, organizations at CMM level 5 still suffer problems that one might not expect at that maturity level such as projects that are still running late or providing unexpected results. The main objective of this paper is to provide means to realize sustainable software process improvements in an organization without restricting this to the domain of software process only. Here, we will answer the following questions:

- Why do software process improvements often not sustain in an organization?
- Why are these improvements slowly (or not at all) progressing in an organization?
- What are the influencing factors in software process improvements?

In section 2, we discuss various change management models to improve an organization. Two of the most influencing models, CMMI [CKS06] and 7S [PW82, PA81, WP80], are compared with each other in section 3. In section 4, we introduce the SFIM method, which encompasses good elements of multiple change management models. In section 5, we discuss SFIM in the context of an industrial case at Philips. In section 6, we discuss related work. Conclusions are drawn in section 7 including some suggestions for future work.

2 Change Management Models

In this section, we will discuss organizational models, which support a change in an organization. A lot of models have been published and still new models are developed in research and they are applied in industry [VBM06, SPI05, HHSE03]. Some models contain multiple viewpoints to address the organizational change. We will discuss a few of them: MOON [WW02] addresses cultural and human aspects, EFQM [HHH96] addresses quality in a full product lifecycle, CMMI [CKS06] identifies various process areas, the BAPO model [HKN+05] addresses four viewpoints, TOP is an integral development model [RHH06] and the 7S model [PW82] addresses seven points of view. The scope of operation of the above models ranges from culture to humans, from architecture to process and from skills to quality. We will shortly discuss these models.

Associates for corporate change developed the MOON-scan [WW02]. (MOON is a Dutch acronym, which means “Model Organizational Development Level”). MOON classifies the development of an organization into four increasing levels: ‘reactive’, ‘active’, ‘proactive’ and ‘top-performance’. The model provides actions based upon human and cultural elements to move to a higher level. This model typically addresses the softer aspects of organizational improvement. MOON is applied in some (Dutch) industry, however not widely known. Interesting is that culture is one of the most important views in this model.

At the end of the eighties, the European Foundation for Quality Management developed the EFQM model as a joint activity of 14 European organizations [HHH96]. The EFQM model explicitly covers the ‘soft’ aspects of improvements such as leadership, strategy, policy and people management and sees them as important enablers for quality management in the whole product lifecycle. EFQM is

applied in many European organizations. The model identifies a number of elements that have impact on quality. Metrics play an important role in EFQM and less attention is paid on causal analysis.

Software Engineering Institute developed the CMMI model [CKS06]. The Capability Maturity Model Integration covers the various capabilities of a development organization in 22 (CMMI version 1.2) process areas. CMMI has a staged and a continuous model. The staged model distributes the process areas over 5 maturity levels thereby indicating the order in which process area to improve first. The continuous model puts all process areas on the same level and lets the organization decide which one to address first. One of the success factors of the staged model is the ability to compare maturity levels of organizations in an objective way. Level 5 organizations have the (software) development process completely under control and are able to improve in any direction they like. The industrial success of CMM in the software community resulted in the CMMI model for system development. CMMI is used all over the world in many different types of industry.

From engineering disciplines, we also know some models that put their activities in a broader scope. BAPO is a model developed at Philips Research to address Architecture (A) that fits in the context of Business (B), Organization (O) and Process (P) [HKN+05]. BAPO is based on several years of experience in developing architectures for large intensive software systems. Applying architecture without having in mind the business, organization and development processes will not make sense.

The TOP model identifies Technology, Organization and Process as dimensions in which system engineering is active [RH06]. The key idea behind the TOP model is that there should be a balance between Technology, Organization and Process for any development activity, for example architecting or configuration management. In case of introducing a new technology it should fit in the organization and process, which may require adaptation in any of the three dimensions.

The 7S organizational model of Peters and Waterman distinguishes the hard and soft aspects of an organization and divide them over 7 views with an “S” name [PW82]. The hard S’s in the 7S model are:

- *Strategy*: the main objectives of an organization and the road to achieve them;
- *Structure*: the organization structure including the roles, hierarchy and coordination;
- *System*: the formal and informal rules

And the soft S’s in the 7S model

- *Style*: the way one behaves and the way of cooperating;
- *Staff*: human resource matters like payment structure, education, motivation and behavior;
- *Skills*: most important and distinguishing skills;
- *Shared Values*: the shared values or the culture of an organization.

From the above model, the 7S model puts attention to the widest range of issues. With respect to the 7S model, the EFQM model also addresses many aspects, but does not explicitly include the organization (structure in 7S). EFQM is focusing on steering on some performance indicators instead of a thorough cause analysis. MOON only

addresses culture (relates to *Shared Values* in 7S). CMMI mainly focuses on processes (relates to *System* in 7S) but also touches other aspects within the process areas, which are further, discussed in the next section. Software process improvement activities often mainly focus on process (as the term already indicates). We argue, as we have experienced during more than a decade of working on software improvements, that a single point of view will not result in success and/or will not sustain. In Fig. 1, we summarize the discussed models from a process perspective (gray areas related to more process focused elements).

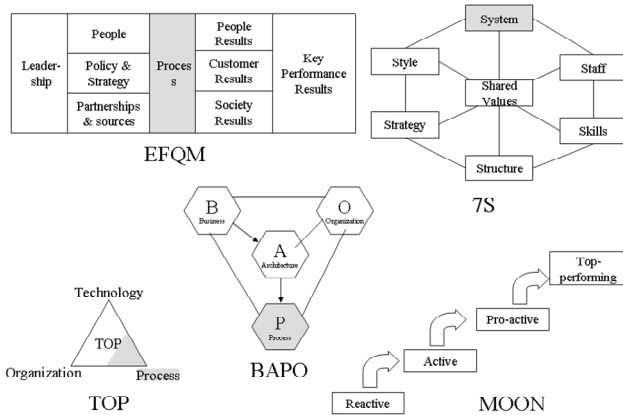


Fig. 1. Change Management Models (CMMI not included)

3 CMMI and 7S Model

In this section we compare CMMI with the 7S model. We show the communalities and differences that exist between the models.

CMMI distinguishes specific practices and generic practices. Specific practices are different for each of the process areas of CMMI. Generic practices however are the same for each process area and they determine the level of institutionalization that is achieved over each process. Common features organize the generic practices: *Commitment to Perform, Ability to Perform, Directing Implementation, and Verifying Implementation*. In Fig. 2, we show relations (dashed lines) between CMMI (ovals) and 7S (boxes).

The main focus of CMMI is on the *System* aspect and the prerequisites to make this work such as *skilled staffing* and organization. Aspects of the 7S model: *Strategy, Style* and *Shared Values* have to be in place but are not explicitly addressed in CMMI. Improvement actions guided by CMMI often fail despite apparent presence of all the prerequisites. Then the question arises: ‘why’ doesn’t it work.

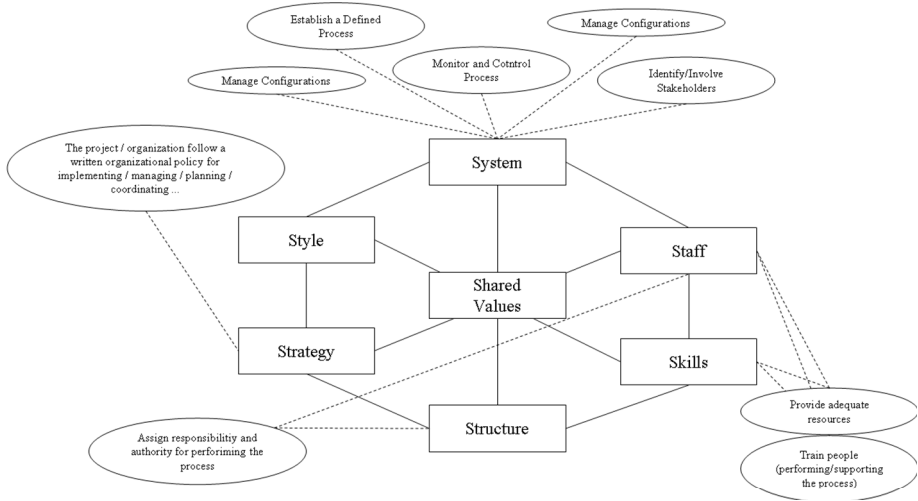


Fig. 2. Example of relations between CMMI and 7S model

The essence of the 7S model is that it specifically addresses the many different aspects of the organization and it emphasizes *harmonization*. Every aspect has to be aligned in the direction the organization wants to go. That might for instance be: move towards higher CMMI levels, improve quality of the product etc.

What we also notice is that CMMI provides Guidance at mainly the lower maturity levels. Even a Level 5 ‘continually improving process performance’ organization needs guidance. In the next section we introduce our method that assists at any level.

4 SFIM: Seven Forces Improvement Method

In this section the Seven Forces Improvement Method (SFIM) is introduced. The key principle of the SFIM method is to improve an organization’s performance in a sustainable way. This can be achieved by iteratively applying all SFIM steps for each Area of Attention. SFIM is heavily based upon existing successful methods and techniques in the field of organizational change methods: the 7S model and Process Improvement models like CMMI.

SFIM starts with the identification of an improvement area. In organizations, it is often outside discussion which areas should be improved. In general, an improvement with a lot of management attention is a good starting point. For an improvement, which we call Area of Attention (AoA), the SFIM method proposes the following steps (see Fig. 3), inspired by the Deming Cycle (*Plan-Do-Check-Act*) [WD86].

1. Define the precise AoA objective. Be sure that the objective is SMART (Specific Measurable Ambitious, Realistic, Time driven), meaning Specific, Measurable, Ambitious, Realistic and Time-Driven.

2. Refine the AoA objective in terms of 7S. This means that the AoA objective is defined per S in the 7S model. This should be complete in the sense that it completely covers the AoA objective.
3. Analyze the 7S objectives with the Force Field Analysis technique [Lew51]. Force Field Analysis is a technique to identify the driving forces and restraining forces for a solution for a certain problem. The forces are identified for example in a brainstorm session. These techniques provide insight in all forces and opens possibilities to especially resolve the negative forces in an organization. This results in a T-table with the driving and restraining forces below the left respectively right part of the T.
4. Improve the organization; first focus on restraining forces. Starting with the hardest issues will pay back in the end. We experience that in many improvements restraining forces are ignored which results in non-sustainable results.

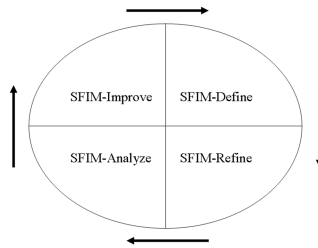


Fig. 3. Seven Forces Improvement Method

5 SFIM Case Study

We have applied the SFIM method in a development organization within Philips producing software intensive systems. In this section, we provide some characteristics, to be able to put the improvement activities as discussed in sections 6 in a better perspective.

The development organization operates worldwide at three sites. During the past ten years, the business has grown from about 100 systems to more than 500 per year. Time to market plays an important role so does Quality. The organization reached CMM level 2 in 1995, level 3 in 2002 (only software departments). Currently, we estimate that the organization is more or less at CMMI level 2 for all development departments.

About three hundred developers develop a highly innovative medical device. Developers have backgrounds in various disciplines from engineering to medicine. Innovative technologies are used to build from mechanical parts, hardware and software a proper working system. The organization is structured in a matrix, project management and line management. Line management is responsible for providing skilled resources and for the long-term quality of the system. Project management is taking care of running various projects in parallel. Each produces new functionality on time.

In this section, we describe the application of the SFIM method to the following Area's of Attention in the described organization: *Project Planning, Monitoring and Control* and *Software Quality*. Other Areas of Attention, with and without SFIM method, are briefly discussed in Section 5.3.

5.1 Project Planning, Monitoring and Control

Project Planning Monitoring and Control is an important process area of CMMI (level 2). It addresses issues such as estimations, risk management and progress tracking in order to control a project.

SFIM-Define: The organization wants to achieve mature Project Planning and Project Monitoring and Control processes at CMMI level 2 within 2 years. Of course, the main objective was to grow in process capability. Reaching CMMI levels is not a goal in itself.

SFIM-Refine: To achieve this the organization needs a strong and explicit project *structure*, with recognizable project management functions and roles and well-defined authorities and responsibilities. The *strategy* to use CMMI as guiding model has to be fully accepted and its consequences understood. Time was spent to convince leading people in the organization. This resulted in fewer discussions in the organization about the need for CMMI level 2. All people in a project management (operational) role or alike must be trained in estimating planning, tracking and managing projects. (*Skills*). The operational organizational axes should be sufficiently *staffed* with types of people that do belief in, and also intend to use a sound project planning and management approach. Management *style* should enforce starting projects or activities only with full commitment of those involved. The quality *Systems* of the organization should be adapted to support the project planning and project management activities. Procedures and manuals should provide the operational people with the right (level of) information to do their job well. The culture (*Shared Values*) of the organization must belief in the benefits of a sound project management approach and the added value of making plans.

SFIM-Analyze: The organization historically has a functional *structure* with many leading people on the functional axis of the organization and many hierarchical layers. There are several 'single' experts and they have to work on several projects in parallel therefore. The project structure also has a very hierarchical project structure with three or more layers: Project, segment and team layer. Project managers are organized in a Project Management Office. Segment and team leaders are located in the various departments of the development organization. Thus both the operational and functional axes are strong resulting in a struggle for power and people blaming each other for failures.

Within the organization the CMMI score is mentioned on the (one page) *strategy* as one of the results to achieve. The organization however tends to do many other improvement activities in parallel thereby not really following the essence of CMMI (Staged) which is to take one step at a time and begin at the beginning. Two forces: the one that tends to follow step by step approach of CMMI and the one that says do it

all at the same time. The risk of the latter approach is that the effort that is put on changes on non-institutionalized processes is lost because the change did not sustain or did not bring the expected benefits.

Skills: Because of the technical background of most people that fulfill a project management role the calculations that are needed for creating schedules and the usage of the applicable tooling is no issue. Sometimes the 'soft' skills need extra training that is also provided. Standard courses on project management are provided to project managers and segment leaders. Team leaders require skills in technical and project management areas. People however who really have both skills tend to move onwards to either a fully technical function or either an operational or line function and stop being a team leader.

Lately the operational organization is *staffed* with more project managers, segment leaders and team leaders. Typical of an organization that is moving from CMMI level 1 to 2 is the change in staff types that is needed. A CMMI level 1 organization depends on its heroes. Types that do well in crises situations, led by gut feeling and people that do not care too much about careful planning and data collection because there is no time for that. Towards CMMI level 2 more of the latter is needed but even at higher maturity levels sometimes projects still run out of control and need to be rescued by the hero types. Opponents of the more structural working method that is required might say: you see we need leadership not administration to run a project. If this force becomes too strong this might result in putting the wrong type in the lead of every project and not only on those that really require this (the ones that ran out of control) and thus pulling the organization back to the CMMI L1 behavior.

We recognize two *styles*: 'process' style versus the 'work hard; play hard' style of management: forget about the planning just do what you can and work as hard as you can which of course does not help getting a proper planning and tracking process to work. Both styles are still present in the organization'. A way of objectively determining which style is winning would be to count how many projects start with real commitment about the targeted end dates of the project. A process style of leadership will only start projects with full commitment of those involved. The organization still tends towards the 'work hard, play hard' style and thus does not benefit from reaching higher CMMI levels.

Systems: The organization had lots of Project Planning and Control procedures in place and reduced them to one small procedure fulfilling the needs of the CMMI and of the organization. The basic procedures and information are in place.

Shared Values: Within the organization it is really widely felt that plans should be based on thorough estimations and that they should be realistic before any commitment is given. On the opposite there are still some people who think that just working as hard as you can, will get the job done faster no matter what the outcome of the planning process is. Again these are two forces working in opposite directions.

The results of SFIM-analyze are visualized in Fig. 4. Dark gray means mainly restraining forces in this view; light gray means forces are neutral (or no issues) from this view and white boxes mean that there are mainly driving forces.

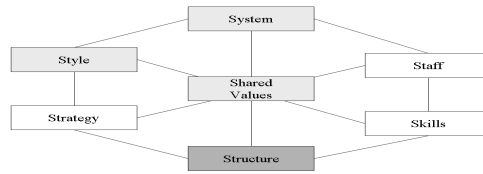


Fig. 4. SFIM-Analyze Project Planning, Monitoring and Control

SFIM-Improve: The organization has already made some changes to bring balance to the forces as mentioned above but still struggling with a few others. We considered the structure related issue the strongest force and there are changes happening already. *Structure*; the hierarchical line organization has been weakened already by merging two departments. Removing a hierarchical layer is considered but not done yet. *Strategy*; the organizations improvement is still focusing at many improvements at the same time. The awareness about the guidance that a model like CMMI can give is growing. *Skills*; training is now seen in the organization as an important driver for improving the output of people. *Staff*; ‘Pushers’ are lesser valued; what remains is the value for leadership in combination with good project administration keeping a discipline. *Style*; Management no longer pushes projects to the limit and accept that the impossible really cannot be done. Realism is the new key word. Projects do not start until it is clear that it can be done. *System* does not require any change. *Shared values*; Because of the change in management style people now dear to speak up in case they think a project or an assignment is not realistic.

5.2 Software Quality

This section Area of Attention that concerns improving the software quality in a sustainable way. In the mid 1990’s the software departments of the studied organization were using coding standards consisting of several rules for the C language and a proprietary C dialect. Compliance with the coding rules was checked with some tools (QAC [PR06] and dedicated house-made scripts) and by manually reviewing the software. This way of working was successful for many years. However, after a change of product’s operating system (from VMS to Windows) and a move to another programming language (from C to C++), during a few years less attention was paid to standards, so coding standards had to be introduced again. A lot of new people joined the organization that did not have the tradition of writing coding standard compliant code.

SFIM-Define. The main objective of this AoA is to introduce new coding standards in the software organization that have to be followed by the programmers. To be SMART in this sense, all the newly developed or modified software has to comply with the coding standards before software is checked in into to code archive. The rules only apply to newly developed code to have a realistic objective. Legacy code that does not comply with the rules is not taken into account. Only software that complies with the above objective may be considered in the daily build.

SFIM-Refine. The main Strategy behind the Software Quality objective is that it should be measurable. The software developers have to learn the coding rules (and rationale behind them) meaning that developers should have the right Skills for writing “error-free” software. For this it is of major importance to have coding rules that do not alter discussions (Style). In order to achieve this for a large group of developers, it also requires a Shared value of delivering a product with high quality software.

The organization, Structure, is built upon three dimensions: Line Management, Project Management and Technical Management. Technical Management address the technical aspects like what are the good rules to achieve high quality software. Line Management is responsible for quality in the long term, taking care of personnel with the appropriate skills and so on. Project Management is concerned to deliver a product on time having the specified functions with good quality. All supported by the right processes.

The above described organization structure requires to have the right people with the right responsibilities (Staff). The balance in the three concerns, Line, Project and Technique, has to be controlled by the right processes in the corresponding System. For example, legacy code should be treated in a different way than completely new developed components. In other words, coding rules should be applied with sense, sometimes rules have to be ignored and/or modified to serve the general objectives. Processes should be in place to change coding rules.

SFIM- Analyze: The organization is organized along three Structures: technical, line and project management. The leading technical people are willing to invest in software quality by means of coding standards; which is clearly supported by the line managers who are “enforcing” the employees to follow the coding rules. Project management is not always convinced about strict application of coding rules; however they share the overall goals of the organization.

In the overall business *Strategy*, the product’s quality is very important. High Quality Product is seen as one of the key selling points. Making software quality measurable by a coding standard and provide tools to automatically check these rules is supporting this business goal. Another positive impact, from strategic point of view, is that any good practice is upgraded to a consolidated (written) procedure. The organization is convinced that tools and processes walk hand-in-hand. Both tools and processes are embedded in the Integrated Development Environment (also illustrated in [BKP05]).

The organization is highly *skilled* in developing high tech products. A lot of software programmers have however less experience in programming languages. This becomes clear when certain coding rules are under discussion because not all programmers understand the rationale behind rules. Especially (at time of introduction) new software technology as C++, COM, C#, and .NET ask for more education. Highly experienced *staff* is hired to implement and support the coding rules and accompanying code-checking tools. The staff is supported by all kinds of monitoring tools to measure the current status of software quality, tuned for different “levels” of insight in coding standards. One software architect is championing the quality topic, supported by different engineers who support engineers at different levels (what is the rationale of this rule? Does this code comply with the rules? Can

this code be checked-in into the code base?). All software engineers have an attitude to deliver high quality results. In the area of software engineering the style of the organization can be characterized as *laissez-faire*. Any initiative that makes apparently sense is accepted, any bad practice will not be removed by any measures of line management, but there may be forces from other engineers to remove it.

A *System* is operational for many years in the organization by means of a Quality Manual, describing the procedures in development. New processes, based on best practices, have been defined and engineers are really involved and aware of these procedures. In case software is submitted to the code base it may only pass when it did not introduce new violations in the code. For this purpose a quality database has been implemented containing the whole history of coding violation in all code files. Status is reported to quality assurance officers, line managers, software architects and software engineers at the corresponding level of interest. An important *shared value* in the organization is that they build the best system of the world. This value is supported by the experience of an earlier usage of coding rules in engineering. In fact, most people in the organization are convinced that quality can be made explicit and measured. On the other hand, the organization has to deal with the culture of discussing all decisions at any place at any time.

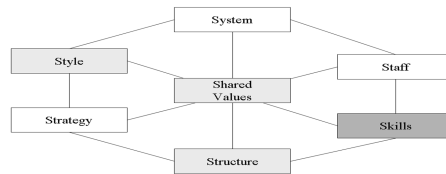


Fig. 5. SFIM-Analyze: Software Quality

SFIM-Improve: The above described S-in-7S describe the current status of the software quality AoA. In the past we have improved on a number of points which are briefly discussed below.

Structure: In various meetings and reports the value of coding standard compliant code is made clear to project management. Also the short term, meaning during the project's lifetime, value of coding rules is made clear by bad practices from the past and the ultimate impact on that project. This all resulted in some movement of project management to adhere to all coding rules in order to enlarge the final quality of the product. The restraining force at Structure level is diminishing.

Skills: Negative points of skills are partially solved by providing developer specialist courses (Microsoft Certified Sw engineering). Checking coding rules to signal the wrong programming attitude solves another part. Explaining the impact at the person's desk by expert helps to move into the right direction. We see that this costs a lot of time, but results are achieved. In the past all developers followed a coding standards course, which showed major acceptance. In the future we plan to organize such a course again.

Style: Management is involved in quality by providing quality performance indicators at a high level. A confidence factor is used to report results to managers [KJ03]. This resulted in improved attention (of managers) to software quality.

Shared Values: A very hard point is to stop discussions that do not contribute to the final result. Different organization wide workshops have been held to change this in the organization. We experienced that this requires a long breath.

In general, the introduction of coding standards was a success. We explicitly addressed the restraining forces at the different S's of the model: *Style*, *Structure* and *Shared Values*. Due to putting much attention to the *Style* aspect, management could be involved. This also motivated people to accept a change in *Sharing Values*. By embedding the processes in the integrated development environment, the organization was able to embed Software Quality in a *structure*. More details are described in [KJ03].

5.3 Other Areas of Attention

Many other improvement activities took place during the last decade in this organization. Many improvement activities were successful which resulted in a CMM level 3 in 2002. Nevertheless, the organization has many difficulties to keep this maturity level and is working on getting CMMI level 2 for all development departments (enlarging the scope). To illustrate SFIM, we discuss some major improvement activities taking place during the last 6 years. Note that this set is not representative.

System Testing (SFIM): the main objective of this AoA was to increase the performance of system testing, resulting in higher quality systems. The *Mean-Time-Between-Crashes* metric was defined to indicate the quality level of delivered systems. All 7 elements in SFIM were addressed to increase the level of success. Much attention was put on having a clear *strategy* for system testing, which was seen as the most restraining force in this AoA. Other elements like *Structure* (a separate group in the organization exists to address system testing), *Staff* (extra roles were defined, e.g. a project's test coordinator), *Skills* (people did follow test management courses, a dedicated test method was introduced) and *Systems* (processes for integration tests, alpha tests and beta tests are in place and well deployed) were in right shape and pointing to the right direction. During the period of improvement less attention was paid to *Strategy* change (although identified by SFIM). The main reason was that it is hard to change this and more attention was paid to the other six elements. The improvement activity was not successful in the sense that it did not result in a sustainable way of testing systems. A lesson learned was that one should stick to the most restraining forces to solve; otherwise the activity is doomed to fail.

Process Database (no SFIM): the main objective of this AoA was to introduce a process database containing all kind of figures of projects in order to achieve better results in future projects. For this AoA, the SFIM method was not applied. Much attention was paid on the technology (tool) and processes to fill the process database. At the end, the process database was filled for a number of projects and hardly anyone was using the content of this process data for new projects ("my project is completely

different from the previous ones”). After a few years, filling the process database was not performed anymore, of course due to lack of a need for it. We have analyzed the failure of this AoA. We may conclude that the *Systems* were addressed well, but there was no *Shared Value* on this topic. Each project manager had his own new ideas when a project started. Furthermore, *Staffing* failed, after a person left the organization, there was no drive for replacement. The *structure* of the organization hampered because the people who had to cooperate were distributed over different organizational units.

General Structural System View (no SFIM): the main objective of this AoA was to define a single structural view that is used by the different disciplines in development (software, mechanics and hardware). After a number of attempts, this improvement slowly progressed. The main reason for slow progress was the absence of a *Shared Value*, but also the right *Structure*, people were not aware of responsibilities in defining and using this structure.

Peer reviews (no SFIM): the main objective of this AoA was to structurally introduce peer reviews of source code and documents. A very enthusiastic person was introducing peer reviews, but there was no (technical) champion in the organization to get the job done (*Staff*). A lot of bureaucracy was introduced which resulted in resistance from engineers. For example, after each review, minutes had to be written that did not introduce new facts. So the process (*Systems*) was not well thought off.

Use Case Introduction (no SFIM): the main objective of this AoA was to introduce a method to close the gap between the user (represented by the marketing department) and development. For this the Use Case approach from the object-oriented community was chosen and adapted. The Use Case Introduction failed after a few years of partial application. The main reason was that the marketing department was not *skilled* to understand the Use Cases (which stem from software community). Furthermore, the processes were not well defined and staffing was only properly addressed by development (and not at marketing).

5.4 SFIM Conclusions

We have applied SFIM on three improvement activities, two of them more successful than the third one. Non-SFIM improvement activities took place with different success rate. In this chapter we discussed some improvement activities that could not sustain in the organization. Unfortunately, we are not able to present exact figures of success and failures of improvement activities.

We conclude from this chapter:

- SFIM should be applied without excluding any of the seven elements. The System Testing AoA showed that relaxing the *Strategy* element resulted in a failure.
- In retrospect, we have analyzed a number of improvement activities in which we could easily identify, in terms of SFIM, which elements were underestimated during the organizational change. This provides us evidence for the SFIM method.

6 Related Work

In [WR94] the authors describe a software improvement program as performed at Schlumberger in the nineties. They demonstrate various elements that have impact on this program. The People-Process-Technology triangle is followed to describe the various points of concern when improving. The P-P-T triangle maps more or less on the earlier described TOP model. The paper states "*Software process improvement must involve the other parts of business with which software interacts, namely marketing, hardware development, sales, manufacturing, etc.*". It shows the importance to broaden the scope of software improvements. The authors indicate that cultural change may be required however hard to achieve. *Shared Values* in the SFIM method addresses the culture aspect. Looking at the discussed topics in this paper, we see that in this industrial improvement program 5 or 6 aspects of the 7S model are addressed although not explicitly referred to by the authors.

Siemens poses that a focus on quality lead to reductions in cycle-time, effort, and costs and thus to business benefit [AP03]. Various activities took place to increase the industrial strength of the company. Process, Quality and Test, Organization and People, Architecture and Agility are discussed in this paper. All these improvement activities lead to more focus on innovation (more time available for) and the long-term success of the company. The paper describes some AoA's that were addressed. In each of them a number of the 7S elements were discussed but not explicitly identified.

Dyba [Dyba05] investigated the effect of factors that influence SPI success. The impact of the factors is determined by means questionnaires send to about 120 managers. The paper concludes that organizational culture is important rather than entirely SW engineering tools and techniques. [Dyba05] did not address issues such as organizational structure nor does it provide means to determine on forehand which aspect to focus on primarily.

In [BMPZ02] 13 lessons learned are discussed during the 3 phases of their existence: emphasis is put on data collection, a learning organization and change. The lessons are grouped in: 1) Need for collecting project data, 2) Need for management buy-in on the process, 3) Need for a focused research agenda and 4) Need for continued staff support (on data collection). They provide some prerequisites for process improvement as also SFIM proposes. The four lessons address *Style*, *System* and *Staff* in the 7S model.

7 Conclusions

In this paper we have introduced the Seven Forces Improvement Method that is based on the 7S model, Force Field Analysis techniques and best practices in software development improvement projects in industry. We have applied SFIM in a large software department for three major improvement activities. Furthermore, we have analyzed some less successful improvement activities from the past having SFIM in mind to learn from it.

The major contribution of this paper is that we show that a wide range of factors play a role in improving software development in a complex organization. Furthermore, we proposed a technique to balance these various factors and how to prevent you, as improver, from spoiling time on the wrong actions. We experienced that a restraining force (e.g. the culture or structure of an organization is not in line with the desired change) is more influencing than pushing activities in other areas (e.g. only defining process guidelines in a quality system). This means that “easiest things first” is not working, but more “hardest things first”.

From this we conclude, that software development improvements will not sustain in an organization when forces, related to one or more of the 7S-es, are pushing in the wrong direction (a restraining force). Meaning that, restraining forces have to be tackled first to be successful. Smoldering restraining forces in the organization may result in a fall back of an initially successful process improvement. In SFIM we identified that *Strategy, Structure, System, Style, Staff, Skills* and *Shared Values* are influencing the performance of software process improvements.

Future work: To increase the confidence in SFIM, the method should be applied in other industries as well (we are challenging readers of this paper and we are willing to discuss issues). Thorough analysis of more improvement stories (success or failure) from a SFIM perspective may help to improve the method itself. Elaboration of the SFIM method could also be achieved by more detailing the four SFIM steps.

Acknowledgement. We want to thank Hans van Vliet for his input on an earlier version of this paper. This work was supported by ITEA SERIOUS (if04032).

References

- [Hum89] Humphrey, W.S.: *Managing the Software Process*. Addison-Wesley Publishing, London (1989)
- [PMF05] Carnegie Mellon Software Engineering Institute, *Process Maturity Profile, Software CMM, 2005 Mid-Year Update* (2005)
- [CKS06] Chrissis, M.B., Konrad, M., Shrum, S.: *CMMI 2 edn.(R): Guidelines for Process Integration and Product Improvement*, The SEI Series in Software Engineering (2006)
- [PW82] Peters, T., Waterman, R.: *In Search of Excellence*. Harper & Row, New York, London (1982)
- [PA81] Pascale, R., Athos, A.: *The Art of Japanese Management*. Penguin Books, London (1981)
- [WP80] Waterman Jr., R., Peters, T., Phillips, J.R.: *Structure Is Not Organisation in Business Horizons* (1980)
- [VBM06] <http://www.valuebasedmanagement.net/> visited (December 31, 2006)
- [SPI05] SPIder working group *integral SPI strategies, Modellen: wanneer wat* (Dutch) (2005)
- [HHSE03] ten Have, S.W., Stevens, F., van der Elst, M.: *Key Management Models*. Prentice Hall, Englewood Cliffs (2003)
- [WW02] Wanrooij, W.: *Corporate Change: de weg naar topprestaties* (Dutch), Scriptum (2002)

- [HHH96] Hardjono, T.W., ten Have, S.W.: The European Way to Excellence: How 35 European Manufacturing, Public and Service Organizations Make Use of Waulit Management, DGIII Industry European Commission (1996)
- [HKN+05] Hofmeister, C., Kruchten, P., Nord, R., Obbink, H., Ran, A., America, P.: Generalizing a Model of Software Architecture Design from Five Industrial Approaches, Succeedings of WICSA (2005)
- [RH06] Reek, E., Hulshout, A.: TOP: Technology Organization and Process (company report) (2006)
- [WD86] Walton, M., Deming, W.E.: The Deming Management Method, New York, Dodd (1986)
- [Lew51] Lewin, K.: Field Theory in Social Science. Harper Row, New York (1951)
- [PR06] <http://www.programmingresearch.com> visited (December 30, 2006)
- [BKP05] Bril, R.J., Krikhaar, R.L., Postma, A.: Architectural Support in Industry: a Reflection using C-POSH, Journal of Software Maintenance: Research and Practice, 17(1) (2005)
- [KJ03] en Krikhaar, R., Jansen, P. In zeven stappen naar een code standaard (Dutch). Informatie (2003)
- [WR94] Wohlwend, H., Rosenbaum, S.: Schlumberger's Software Improvement Program, IEEE Transactions on Software Engineering, 20(11) (1994)
- [AP03] Achatz, R., Paulisch, F.: Industrial Strength Software and Quality: Software and Engineering at Siemens. In: Proc. International Conference on Quality Software (2003)
- [Dyba05] Dyba, T.: An empirical Investigation of the Key Factors for Success in Software Process Improvement, IEEE Transactions on Software Engineering, 31(5) (2005)
- [BMPZ02] Basili, V.R., McGarry, F.E., Pajerski, R., Zelkowitz, M.V.: Lessons learned from 25 years of process improvement: The rise and fall of the NASA Software Engineering Laboratory, ICSE (2002)

SPI-KM - Lessons Learned from Applying a Software Process Improvement Strategy Supported by Knowledge Management

Gleison Santos, Mariano Montoni, Sávio Figueiredo, and Ana Regina Rocha

COPPE/UFRJ - Federal University of Rio de Janeiro
POBOX 68511 – ZIP 21945-970 – Rio de Janeiro, Brazil
{gleison,mmontoni,savio,darocha}@cos.ufrj.br

Abstract. Software development organizations recognize the importance of improving software processes to enhance their competitive advantages. COPPE/UFRJ software process research group has been providing SPI consultancy services to the Brazilian software industry for more than two decades. In order to support the SPI activities of the group, a SPI deployment strategy named SPI-KM that is supported by Knowledge Management and has been developed based on international and national reference models and standards. This paper presents the SPI-KM strategy and the results of an empirical study executed aiming to characterize the SPI initiatives that employed it. The study findings are presented as lessons learned and their applications are discussed in different organizations. We consider the adoption of the SPI-KM strategy and the lessons learned as important knowledge to be appreciated during SPI initiatives aiming to facilitate SPI deployment and to assure their success.

1 Introduction

Its necessary to drive the improvement initiatives based on organizational business goals, i.e., a SPI deployment plan must be elaborated, executed and monitored aiming to achieve these goals, to minimize impact on resources and to maximize return on investments [1]. The significant time to fully implement an SPI initiative is often considered too expensive for many organizations as they need to commit significant resources over an extensive period of time. Moreover, SPI initiatives exhibit low levels of adoption and limited success (the failure rate of SPI initiatives is estimated as 70%) [11]. Therefore, there is a need of developing effective strategies to increase processes' maturity and capacity in software organizations.

COPPE/UFRJ software process research group comprises 2 PhD and 12 PhD and master students of the Federal University of Rio de Janeiro. This group also has been providing Software Process Improvement (SPI) consultancy services to the Brazilian software industry for more than two decades. In order to support the SPI activities of the group, a SPI deployment strategy named SPI-KM was developed based on international and national reference models and standards. The SPI-KM strategy is supported by Knowledge Management and has been applied in more than 30 organizations. The results of SPI-KM application are promising since all of the

organizations that applied the strategy have been successfully assessed by official appraisals and demonstrate positive impacts and benefits of their SPI initiatives.

This paper presents the main characteristics of the SPI-KM strategy and the results of an empirical study aiming to characterize the SPI initiatives that adopted the strategy. Through this study we were able to identify common problems that could jeopardize SPI deployment supported by the SPI-KM strategy. The study findings are presented as five lessons learned. To adduce how the problems pointed out have been identified and overcome we discuss their occurrences during SPI initiatives in five different Brazilian organizations.

The next section discusses the deployment of SPI initiatives and the maturity models CMMI and MPS.BR. Section 3 presents how Knowledge Management can support SPI. Section 4 describes the main characteristics of the SPI-KM strategy. The findings of the empirical study executed are described in section 5. Finally, section 6 presents final considerations and points out future directions.

2 Software Process Improvement

Over the last years a consensus has emerged that an iterative process of assessment and improvement of the software process is essential to increase understanding and manageability of software development process, to ensure quality of the product, to reduce costs and maximize productivity [6]. In order to reduce the time to assess and introduce process changes, an adequate SPI infrastructure must be defined and implemented. According to Krasner [19] a successful systematic SPI program requires: (i) well defined objectives, (ii) a method for catalyzing and institutionalizing the SPI program in an organizational setting, (iii) one or more goal/maturity model for guidance, (iv) best practice examples and benchmarks to draw from, (v) an organizational commitment to action in the form of an improvement roadmap that is defined, resourced and followed, (vi) expertise in process diagnosis, culture change tactics, process problem solving, etc, and (vii) a set of champions/change agents that can sponsor, commit to and effectively implement a planned SPI program.

International standards like ISO 12207 [15], ISO 15504 [7] and software process quality models like CMMI (Capability Maturity Model Integration) [8] were developed aiming to define the requirements of an ideal organization, i.e., a reference model to be used in order to assess the maturity of the organization and their capability to develop software.

Based on these standards and models, Brazilian industry and research institutions have worked together during the last two years defining a Reference Model for Software Process Improvement in Brazil (MR-MPS.BR) in order to enhance the maturity of the processes of Brazilian software organizations and to improve the quality of its products [9, 10]. Among the main goals of the MR-MPS.BR, the Brazilian Reference Model was meant to be affordable for small and medium-sized companies. Therefore, instead of having five maturity levels like CMMI, the model comprises seven maturity levels in order to make possible for the company employees, managers and partners to see the results soon. Moreover, the company can enhance the maturity of its processes gradually and with less effort to go from a lower maturity level to the next one. For each of these maturity levels, processes are assigned based on the ISO/IEC

12207 standard and on the process areas of CMMI staged representation. Table 1 presents the mapping between the MR-MPS.BR and CMMI maturity levels.

Table 1. Mapping between CMMI and MR-MPS.BR Maturity Levels

CMMI Maturity Level	MR-MPS.BR Maturity Level	MR-MPS.BR Processes
2 – Managed	G - Partially Managed F – Managed	Project Management, Requirement Management Measurement, Acquisition, Configuration Management, Quality Assurance
3 – Defined	E - Partially Defined D - Largely Defined C - Defined	Training, Process Establishment, Process Assessment and Improvement, Tailoring Process for Project Management Requirements Development, Technical Solution, Software Integration, Verification, Validation Decision Analysis and Resolution, Risk Management
4 – Quantita- tively Managed	B - Quantitatively Managed	Organizational Process Performance, Quantitative Project Management
5 - Optimizing	A - Optimization	Organizational Innovation and Deployment, Causal Analysis and Resolution

Recent research focuses on identifying factors that have a positive impact on the deployment of SPI programs. For instance, Niazi et al. [12] identify that organizations should pay particular attention to: awareness of SPI benefits, defined SPI deployment methodology, experienced staff, higher management support, staff involvement, training, and others. Dybå [13] also executed an empirical investigation to identify success factors of SPI deployment, for instance, business orientation and employee participation. In the context of Brazilian organizations, other factors have also been identified [14] to have positive impact in the deployment of SPI initiatives, for instance, organizational culture change, use of supporting tools, motivation, follow-up of deployed process, and process alignment to organizational business strategies. Among other facts, these studies have been considered and pondered while we defined the SPI-KM strategy to be presented in the next sections.

3 Supporting SPI Through Knowledge Management

SPI deployment is a knowledge-intensive task since it requires different types of knowledge regarding software processes (e.g. software processes models, best practices and lessons learned). Software processes models, for instance, explicitly represent knowledge about software development activities, but also the software products, necessary resources and tools, and best practices related to software processes execution [23]. Therefore, efficient management of such knowledge supports organizational learning and SPI deployment initiatives [24]. Another aspect to be considered is the experimental, evolutionary and non-repetitive characteristics of the software engineering area [20], which means that there are approaches that work best in certain situations and it is necessary to tailor them in order to deal with new situations. Moreover, unforeseen events may occur despite careful software project planning. This implies

making constant choices among the many feasible options throughout the deployment of SPI [21]. As a result, many software companies have recognized the importance of administrating knowledge effectively, productively and creatively at both the individual and organizational levels [22]. The fact that most software development organizations are process-centered provides many benefits (e.g. process-centered knowledge management systems can be designed to explicitly associate software process activities with knowledge necessary to execute it) [24]. Moreover, tacit and explicit member’s knowledge regarding software processes are valuable individual assets that must be captured and converted into the organizational level. This important knowledge can be used to learn about the software process and to provide the means for implementing organizational changes aimed to enhance business performance [25].

4 SPI-KM: A Software Process Improvement Approach Supported by Knowledge Management

In order to support SPI deployment initiatives, we developed a strategy that has evidenced its feasibility and benefits over past well-succeeded SPI appraisals [16, 2]. The strategy consists of a set of defined phases that focus on specific issues related to SPI initiatives’ deployment; it has the support of Knowledge Management and takes advantage of the use of a Process-centered Software Engineering Environment (PSEE). The strategy is depicted in the Figure 1.

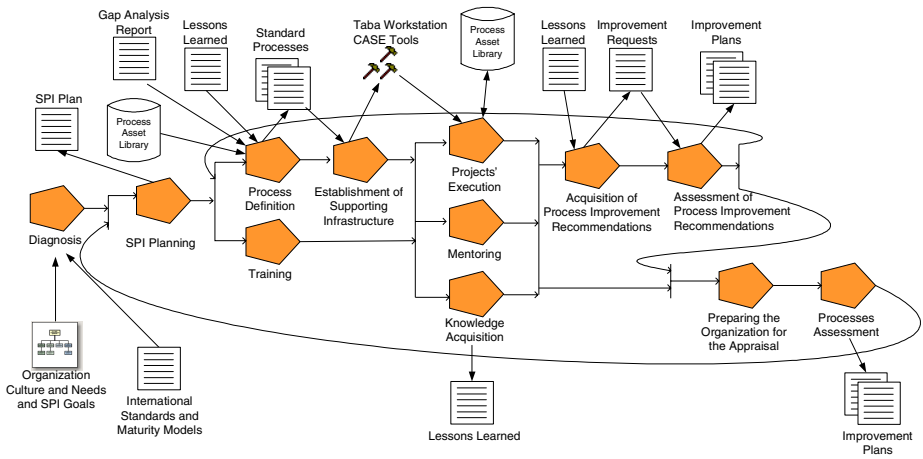


Fig. 1. SPI-KM phases

4.1 SPI-KM Phases

Diagnosis: The strategy begins when the software organization aiming to enhance its processes gets in touch with COPPE/UFRJ research group. At first, the organization business needs and goals, the organizational culture, the SPI goals, the software process reference model to be used and the level desired are identified with the

high-level management. The organizational unit that is going to take part of the SPI initiative is also identified.

SPI Planning: During this phase, a plan for the SPI initiative is developed. This plan comprises, among other things: (i) the consultant team that is going to be allocated during the initiative, (ii) the organizational members to be trained, (iii) the schedule for the trainings, (iv) the processes definition prioritization regarding organizational goals and strategic needs, (v) creation of groups of work with designated responsibilities, (vi) definition of supporting tools, infrastructure and operation responsibilities, and (vi) the expected appraisal date. The SPI plan is reviewed at predefined milestones (e. g., after a review of processes improvement recommendations or after processes assessment).

Process Definition: It involves a series of meetings aiming to assess the organization processes in order to identify their current state of practice. At this moment, a process is defined trying to regard the activities that software developers in the organization already deploy and trying to be adherent to the practices of the capability maturity model level selected on the prior phase. If the company already uses a software process, a gap analysis is performed to identify practices needed to accomplish de SPI goal (e.g., required practices of maturity models). If the company does not have defined software process yet, a new one is defined based on the consultancy experience and lessons learned. Regardless the maturity model and the level selected, a standard software process is always defined and institutionalized in the organizational level. We are confident that the adoption of an institutionalized process to guide projects execution since the initial phases of SPI initiatives is essential to catalyze improvement changes and to make the SPI cycles faster. The Software Engineering Knowledge Base available in Taba Workstation through the use of Acknowledge CASE tool [17] provides valuable lessons learned and best practices to improve the efficiency and correctness of the processes to be defined.

Training: In this phase training in Software Engineering methods and techniques are provided to organization members. The training program is tailored to the characteristics and needs of the organization and its SPI initiative; for example, comprises the process areas of CMMI Level 3 or MPS.BR Level G processes (Project Management and Requirements Management). Oftentimes, it includes training in the software process defined, practices required by the capability maturity model and tools to be used. Some training activities are also carried out along with mentoring sessions during projects execution.

Mentoring: It takes place during projects execution and involves direct knowledge transference to the organizational members. COPPE/UFRJ software engineering consultants are present while the software developers carry out any particular process activity for the first time, explaining how to execute that activity and the benefits expected. This close contact between the organization members and the consultants accelerates the learning process, increases the awareness of SPI benefits and minimizes resistances to changes. The knowledge items of the Software Engineering Knowledge Base help the consultant to support the activities of the organizational member during the mentoring. Nevertheless, all the knowledge is always available to any user of Taba Workstation.

Knowledge Acquisition: It involves the acquisition of knowledge, from consultants and organizational members, regarding software process activities and the SPI initiative in order to allow the organizational knowledge preservation and dissemination. After collecting the knowledge, it is filtered, packaged, stored in an Organizational Knowledge Repository and made available to guide process executions and SPI plans reviews. The support to knowledge management in Taba Workstation is provided by Acknowledge CASE tool [17] which is integrated to all others CASE tools in the environment.

Acquisition and Assessment of Process Improvement Recommendations: The acquisition of processes improvement recommendations occurs in parallel with the project execution. Process improvement ideas appear while developers get a better grasp about the process. These improvement suggestions are collected and assessed by the organizational process group and, if approved, it is incorporated into the standard software processes and can influence future reviews of the SPI plans. People affected by the changes are trained again and, new projects can use the new process.

Preparing the Organization for the Appraisal: The high management defines the expected appraisal date and commits on all necessary resources in order to achieve the SPI goals. To increase the success of the appraisal, two activities are executed during the Appraisal Preparation Phase: the fulfillment of the evidence worksheet that is going to be assessed by the appraisal team, and training the project members for the appraisal interviews that will be carried out during the appraisal. Basically, the worksheet contains the practices that an organization has to execute aiming to be adherent to the selected level of the software process reference model, and under these good practices the organization has to link artifacts that provide evidence of these practices deployment in the organization. During this phase the consultants also explain to the project members the different questions that are going to be made during the interviews and how they are going to be conducted.

Processes Assessment: The improvements deployed assessment is important to make evident the impact and benefits of the SPI initiative. Therefore, one of the characteristic of the strategy is that an official appraisal constitutes the final milestone of the SPI initiative.

4.2 Taba Workstation: Supporting the SPI-KM Strategy

The use of Taba Workstation, a Process-centered Software Engineering Environment (PSEE) that supports software processes definition, deployment and enactment, is a key factor of this strategy whose goal is to increase the processes' capability of organizations through the suitable use of Software Engineering techniques in their software processes aiming to enhance the software products quality and, thus, increase organizational competitiveness. This environment has been developed by the COPPE/UFRJ software process engineering research group in the context of an academic project and it is granted to software development organizations with no costs.

Taba Workstation provides support to a wide range of activities related to standard processes definition, enacting and evolution, software projects planning and monitoring (covering most Project Planning and Project Monitoring and Control process areas

of CMMI and equivalent MPS.BR process), measurement and analysis, design rationale related to technical solutions [18], requirements management and verification and validation planning etc. It also integrates knowledge management activities within software processes aiming to preserve organizational knowledge, and to foster the institutionalization of a learning software organization [17].

5 Results of an Empirical Study of SPI Experiences that Adopted the SPI-KM Strategy

The empirical study presented in this section aimed to identify common problems that had to be coped with during the application of the SPI-KM strategy. The study was conducted through a survey with SPI practitioners that participated in SPI projects that adopted the SPI-KM strategy. Unstructured interviews were also used to better understand the findings and also to characterise these findings in the organizations where the SPI-KM approach were applied. To foster discussion, 5 cases that exemplify the findings are presented. We consider the adoption of the SPI-KM strategy and the lessons learned as important knowledge to be appreciated during SPI initiatives aiming to facilitate SPI deployment and to guarantee their success.

5.1 Methodology

At first we identified the COPPE/UFRJ members with experience in deploying software process that were going to participate in the survey. 15 members were identified to take part of the survey. The practical experiences of this group include SPI deployment in more than 20 organizations, 3 of which have gone successfully through official CMMI appraisals, 2 have gone successfully through official MR-MPS.BR appraisals and 2 have gone successfully through both official appraisals. All organizations have used the same strategy as described in the previous section to plan and execute their SPI initiatives. Minor adaptations have been made while tailoring the strategy to the specific needs and cultural aspects of the organizations. More than 7,000 staff-hours of consultancy effort were spent during those SPI deployments. A questionnaire was distributed among the participants; all questionnaires were returned with findings related to difficulties identified during the deployment of software process. In order to obtain objectivity in the survey, the questionnaire did not contain pre-identified items and the participants answered the questionnaires with no contact to each other. After the return of the questionnaires, the difficulties and success factors were categorized according to the similarities of the findings. These findings and categories are described in [26].

Later, interviews were conducted with 4 of the most experienced consultants aiming to identify the findings that had the most significant impact in the organizations which have gone through official CMMI or MR-MPS.BR appraisals; 5 findings were identified. At least 2 of these 4 consultants have played key roles in the deployment of SPI-KM approach in the organizations. New unstructured interviews were conducted to most of the consultants that implemented the SPI-KM approach in organizations and also with key members of the organizations involved in the SPI deployment. The interviewees were asked to talk about the SPI initiative and the organizational culture on SPI. Last but not least, they were asked to identify how the occurrence of the 5 most significant findings was perceived and how their absence was overcome.

Finally, the results of all interviews and questionnaires were analysed and we selected the 5 most significant findings based on their impact to the SPI-KM approach deployment. These findings were written as lessons learned to foster their dissemination. We also selected 5 organizations to exemplify how the lessons learned were realized and how the problems were overcome. It does not mean, however, that these lessons were not useful neither had not the problems been identified in other organizations. We have selected these organizations based on the impact of the lessons on the SPI-KM approach deployment according to the interviews.

Organization A is the Brazilian unit of a large-sized international corporation that develops and maintains web based, client-server based, data warehouse, business intelligence and helpdesk systems. The initial SPI goal of the organization was to achieve CMMI Level 2 and to be certified on ISO 9001:2000, and then, gradually improve the processes aiming to reach higher levels of MPS.BR and CMMI models.

Organization B is a medium-sized Brazilian company concerned with software development, maintenance, deployment and integration. Its SPI main goals are to improve products quality and increase their productivity. Four cycles were defined: achievement of the ISO 9001:2000 certification, achievement of MPS.BR Level F, achievement of CMMI Level 3, and achievement of CMMI Level 5.

Organization C is a medium-sized Brazilian organization that develops e-commerce systems. The organizational structure is defined based on similar types of projects and common resources are shared by all teams, for instance, infrastructure, support services, marketing and sales. Since 2004, this organization has invested in SPI initiatives for increasing competitiveness and reaching higher maturity levels.

Organization D is a governmental agency responsible for executing electoral services activities for the Brazilian Federal Government. Most of these activities are supported by electoral systems developed by three organizational units. The SPI main goal was to standardize the software process in its three electoral systems development units.

Organization E is a large-sized Brazilian organization that provides services to companies and institutions in different economic sectors (electrical, water and gas companies and public sector). The main goals of the SPI initiative were to enhance organization members' competences, to increase their satisfaction and productivity, to increase organizational competitive advantages and to acquire international customers.

5.2 Study Findings: The Lessons Learned

The following lessons learned were identified after the analysis of data from the survey and the interviews.

Lesson 1: SPI initiatives should effectively improve software processes - The SPI must be established on a process improvement program which defines the strategies, policies, goals, responsibilities and activities concerned with the achievement of specific improvement goals. This program can span more than one complete cycle of SPI. On each of these cycles actions should be taken to change processes so that they are more effective and efficient to meet organization's business goals.

Lesson 2: You will not succeed without a leader - SPI deployment implies the adoption of new practices in the organization. Therefore, many barriers are encountered during

SPI endeavor, for instance, organizational politics, lack of support and resources, lack of knowledge and schedule pressure [4]. It is, therefore, very important to conduct SPI initiatives appropriately aiming to overcome the inherent difficulties.

Lesson 3: Commitment is crucial - SPI literature [1] recognizes that without commitment from all organizational levels to SPI the improvement goals will be difficult to achieve. So, people involved with the SPI initiative must perceive the benefits deriving from its deployment, and not only its costs.

Lesson 4: No brain, no gain - Once this difficulty is found in an organization, most of the methods and techniques used to support software development and management must be taught increasing the cost, difficulty and time to achieve the SPI goals. Therefore, a capacity program for enhancing members' knowledge eases the employment of new practices in both project and organizational levels. This particular type of training catalyzes knowledge transference and is considered to be one of the pillars for creating a learning software organization.

Lesson 5: SPI is facilitated by software process infrastructure - Most organizations with low maturity software processes do not have suitable infrastructure for SPI deployment. Zaharan [5] defines two types of infrastructure to support process-related activities and to sustain SPI actions: (i) organization and management infrastructure, and (ii) technical infrastructure.

5.3 Discussion on the Lessons Learned

This section exemplifies how the lessons learned were realized in 5 organizations that adopted the SPI-KM strategy and how the problems were overcome. Table 2 summarizes the SPI-KM adoption by each organization. It also presents the lessons learned identified as relevant to the SPI initiatives goals.

Table 2. SPI-KM strategy adoption, lessons learned relevance and SPI cycles

Organization	Relevant Lessons	Cycle	SPI Initiative Goal	SPI-KM Use	Result
A	1, 4, 5	1	ISO 9001:2000 and CMMI Maturity Level 2	Yes	Achieved
		2	MPS.BR Maturity Level E	Yes	Achieved
		3	CMMI Maturity Level 3	Yes	Not yet
B	1, 5	1	ISO 9001:2000	No	Achieved
		2	MPS.BR Maturity Level F	Yes	Achieved
		3	CMMI Maturity Level 3	Yes	Achieved
		4	CMMI Maturity Level 5	Yes	Not yet
C	1, 2, 3, 4, 5	1	MPS.BR Maturity Level G	Yes	Not achieved
		2	MPS.BR Maturity Level D	Yes	Achieved
		3	CMMI Maturity Level 3	Yes	Not yet
D	4, 5	1	CMMI Maturity Level 2	Yes	Achieved
E	2, 3, 5	1	CMMI Maturity Level 2	Yes	Achieved
		2	CMMI Maturity Level 3	Yes	Not yet

Lesson 1: SPI initiatives should effectively improve software processes - One important factor we have observed is the SPI results monitoring to guarantee that the initiatives are effectively improving software processes. SPI initiatives can be monitored by defining performance indicators in order to ensure that process performance is on track. Moreover, process monitoring and feedback mechanisms must be established to support the use of feedback data to evaluate the payoff for doing improvements [5, 3]. If SPI costs are viewed as an investment, then the payoff is expressed in a temporally-shifted, return-on-investment (ROI) model [3]. Management indicators within ROI models of SPI include, for instance, measures of: product quality, process quality, project predictability and customer satisfaction. Nonetheless, some of the biggest payoffs of SPI are expressible in human terms, not monetary units. They might involve: better job satisfaction, pride in work, an increased ability to attract, retain grow experts that will innovate, company reputation for excellence, etc [3].

Organization B used to gather quantitative data related to the execution of its software projects even before the beginning of its SPI initiative. Analyzing this data we could realize the relation between the adoption of specific software quality activities and the time expended on rework along the projects [2]. Due to the non existence of quality activities before the first SPI cycle, 44% of projects total schedule was spent in activities related to rework. The adoption of quality assurance activities in the first SPI cycle (9.2% of projects total time) reflected in 26.7% of time expended in rework activities. In the third cycle, 10.8% of the time was expended in quality assurance related activities and the rework ratio dropped to merely 7.3%. As a direct effect of these achievements we can point out high management support to all SPI activities and commitment to new SPI cycles, impressive collaborators' satisfaction and significant decrease of people turnover.

On the context of Brazilian industry, the focus of the MR-MPS.BR model is to enable a more gradual and suitable software process deployment in small and medium size companies. We observed that SPI initiatives based on MPS.BR can be defined to address more immediate organization's business and improvement goals. In fact, the Organizations A, B and C used this strategy of alternating MPS.BR and CMMI evaluations while planning their SPI goals and so far, all of them accomplished good results. Moreover, rating companies' maturity considering more levels, not only decreases the cost and effort of achieving a certain maturity level, but also allows the visibility of the results of the SPI initiative within the company and across the company's boundaries in a shorter time when compared against other models, such as CMMI.

Lessons 2 and 3: You will not succeed without a leader and Commitment is crucial -

These lessons are somehow related. One factor that was perceived to have influenced the success of our SPI experiences regards to organization commitment (from lower level to the higher one): it is very difficult to obtain organization members commitment to SPI in some organizations. Another difficulty was to maintain the organization commitment during all SPI cycles. In order to cope with this problem, SPI quantitative data related to time, cost, quality and customer satisfaction were continuously communicated to high level managers. So, managers could perceive the benefits deriving from the SPI deployment, and not only its costs.

According to our experience in several organizations, a leading group responsible for promoting awareness of SPI and to support knowledge sharing among different practitioners is crucial to the success of SPI initiatives. This group is sometimes a full time resource with responsibility to manage the deployment and coordination of SPI activities [5]. The group is also responsible for obtaining and sustaining high level commitment with different management levels and project members during all SPI deployments. In some organizations they are considered as real heroes, i.e., the SPI leaders behaviors are highly prized, serving as role models for others in the organization.

On the first SPI cycle, *Organization C* was not ready to endeavor in a SPI initiative. The investment in training and mentoring was insufficient. Their customers were not informed about the necessary involvement during project development required by the SPI initiative. Moreover, higher management didn't assign clear responsibilities and didn't consider the improvement program as high priority. Therefore, most of improvement actions were ineffective due to lack of organization members' commitment. However, this first experience provided important lessons to planning the second SPI cycle. On the second SPI cycle, the mistakes of the first cycle didn't occur. A new strategy was defined cooperatively with sponsors considering resource availability and business goals. The SPI plan was elaborated in joint with consulting team which pointed essential activities and established milestones to evaluate the progress of the project. Organization members were also mobilized to participate proactively and contribute with process improvement opportunities. All stakeholders were aware of the importance and benefits of their involvement and of the SPI initiative. The continuous training program was launched to improve knowledge level about how to increase process quality and create awareness about continuous improvement. As a result of this approach the organization successfully achieved its goal (to be evaluated MPS.BR maturity Level D).

At the beginning of *Organization E* first SPI cycle, there was a great resistance to perform the process activities, because the organization members were used to execute a process without many complex and important tasks, for instance, product and process quality assurance activities. To deal with this resistance, all the activities were executed with the support of an experienced consultant. This approach was effective to train organization members in the process, tools and Software Engineering best practices. This constant contact with organization members made them acknowledge the benefits of the SPI benefits. Moreover, the mentoring during project execution minimized the risk of giving low priority to other activities having priority over the project activities. High level management support was also very important in this aspect.

Lesson 4: No brain, no gain - The most relevant deficiency we have detected in many organizations was the lack of knowledge in Software Engineering and Project Management. Mentoring activities performed by specialists is part of our SPI strategy as consultants and are carried out constantly during the whole SPI cycle, sometimes on a daily basis. Mentoring activities, besides teaching engineering methods and techniques, how to use CASE tools and how to execute the software process, also

help consultants to enforce the benefits of the SPI program and the necessity of being committed to the improvement goals.

During the second SPI cycle in Organization C, the project managers worked together with consultants that acted like mentors aiming: (i) to avoid knowledge gap about process tools and activities, (ii) to obtain continuous project management commitment, and (iii) to stimulate contributions to process improvement. The mentoring also offered support to execution of organizational process-related activities like configuration management, product and process quality assurance. Some issues that offered risks to the SPI initiative, including resistance to changes, were identified and carefully treated by mentors. The project managers used to have a schedule-oriented reactive behavior for solving most of the problems. During mentoring, the constant presence of mentors was very important to enforce organizational policies and to support project managers during process execution. Other direct related result from mentoring besides huge knowledge transfer was the involvement of project management regarding improving the process. When they recognized that something could be improved, an improvement recommendation request was formalized and submitted to the process group.

In Organization A, in parallel to the processes definition activity, training in Software Engineering methods and techniques were provided to its members as lectures on topics such as Software Engineering, Software Process, Knowledge Management, Software Products Quality, Project Management, Configuration Management, Measurement and Analysis, Requirements Engineering, Peer-review, Tests, Knowledge Management and Function Point Analysis. All project managers, system analysts, developers and SEPG (Software Expert Process Group) members were trained. After the definition of the process more training session as lectures on the processes were conducted. There was no strong resistance to use the new standard process, since people in the organization was used to execute a defined process. The consultancy efforts were concentrated on mentoring activities to coach the organization's members in the Taba Workstation and Software Engineering best practices. The mentoring allowed concentrating efforts to guarantee the fast execution of the projects and to guarantee that the activities were adherent to the maturity model practices.

In Organization D, training in CMMI level 2 process areas, in software engineering concepts and in the process were important steps to minimize resistance and to obtain general satisfaction during the SPI initiative in this company. Mentoring activities were also performed along the execution of projects to support the accomplishment of process activities and to support the use of Taba Workstation tools. This mentoring was essential to motivate organization members, instead of stopping in the first difficulty during process execution. In order to guarantee SPI institutionalization, to minimize variation on the activities execution and to promote best practices dissemination, we provided support to the acquisition of process directives during the execution of mentoring activities throughout the projects. Once the directives were acquired, a Knowledge Manager assessed each one of the directives and indexed it in an Organizational Knowledge Repository. Dissemination of the assessed directives to all the employees was achieved through regular trainings.

Lesson 5: SPI is facilitated by software process infrastructure - In order to provide an adequate software process infrastructure to software organizations, our SPI strategy is supported by Taba Workstation that supports individual and group activities and project management activities [14, 2]. All the 5 organizations presented used extensively the Taba Workstation CASE tools. In several CMMI and MPS.BR official appraisals, the Taba Workstation PSEE was reported as a significant strength for the SPI deployments.

At the beginning of Organization B second SPI cycle the organization decided not to use any CASE tool to support the process deployment, including Taba Workstation. This was supposed to decrease the impact during the initial stages of its process deployment. But difficulties to manage the project pointed out that a CASE tool was necessary to support the process utilization and, moreover, to support planning, control and execution of the project. Due to this fact, Taba Workstation utilization was reconsidered and from this moment on it began to be used. In the beginning the environment was used only to control software process workflow. Eventually all Taba Workstation CASE tools started to be used to support each step of the process enactment. Nowadays besides using the Taba Workstation, the Organization B developed internal CASE tools to deal with specific characteristics and needs of its development unit: a process management system, document management system and a workflow system. The tight integration of these tools was reported as a significant strength for its SPI program during the CMMI Level 3 appraisal.

6 Conclusions and Future Work

This paper described lessons learned from the application of the SPI-KM strategy by COPPE/UF RJ software process research and SPI consultancy group. Nonetheless, many problems may affect SPI institutionalization, e.g., high people turnover and loosening of organization politics. Therefore, efficient mechanisms must be implemented to assure that problems that can impact software processes institutionalization will be addressed in a suitable manner. In order to efficiently maintain SPI deployments, it is important to manage software processes knowledge. By doing so, SPI practices can be easily disseminated and evolved. Moreover, knowledge necessary to sustain SPI programs is preserved, reducing the risks of losing important knowledge due to people turnover. Our group has also observed that organizations face difficulties to sustain SPI deployment results as time goes by. For instance, after succeeding in a specific SPI initiative, difficulties emerged in some organizations due to stagnation of its processes and infrastructure. Therefore, software organizations must continuously promote SPI changes aiming to enhance their competitive advantages and guarantee survival.

The set of lessons learned presented in this paper will provide the means to refine the SPI-KM strategy in order to ease the training of new members of the SPI practitioners and to enhance the results of SPI programs executed by Brazilian software industry. Also a new empirical study is being planned and will be executed in 2007 aiming to characterize success factors that have influence on SPI initiatives.

References

1. Abrahamsson, P.: Commitment Development in Software Process Improvement: Critical Misconceptions. In: Proceedings of the 23rd Int. Conf. on Sof. Eng, pp. 71–80 (2001)
2. Ferreira, A.I.F., Santos, G., Cerqueira, R., Montoni, M., Barreto, A., Rocha, A.R., Figueiredo, S., Barreto, A., Silva Filho, R.C., Lupo, P., Cerdeiral, C.: Taba Workstation: Supporting Software Process Improvement Initiatives based on Software Standards and Maturity Models. In: Richardson, I., Runeson, P., Messnarz, R. (eds.) EuroSPI 2006. LNCS, vol. 4257, pp. 207–218. Springer, Heidelberg (2006)
3. Krasner, H.: Accumulating the Body of Evidence for The Payoff of Software Process Improvement. In: Software Process Improvement, pp. 519–539. IEEE, NJ, New York (2001)
4. Niazi, M., Wilson, D., Zowghi, D.: A framework for assisting the design of effective software process improvement implementation strategies. *J. of Systems and Software* 78(2), 204–222 (2005)
5. Zaharan, S.: Software Process Improvement – Practical Guidelines for Business Success. Addison-Wesley, London (1998)
6. Allen, P., Ramachandran, M., Abushama, H.: PRISMS: an Approach to Software Process Improvement for Small to Medium Enterprises. In: Proc. Of the Third International Conference On Quality Software, pp. 211–214 (2003)
7. ISO/IEC 15504 – 1 Information Technology – Process Assessment, - Part 1: Concepts and Vocabulary (2003)
8. Chrissis, M.B., Konrad, M., Shrum, S.: CMMI: Guidelines for Process Integration and Product Improvement. Addison-Wesley, London (2003)
9. MPS.BR - Melhoria de Processo do Software Brasileiro, Guia Geral (v. 1.1) (in portuguese) (2006), Available at <http://www.softex.br/mpsbr>
10. Rocha, A.R., Montoni, M., Santos, S., Mafra, S., Figueiredo, S., Albuquerque, A., Mian, P.: Reference Model for Software Process Improvement: A Brazilian Experience. In: Richardson, I., Abrahamsson, P., Messnarz, R. (eds.) EuroSPI 2005. LNCS, vol. 3792, pp. 130–141. Springer, Heidelberg (2005)
11. Niazi, M.: Software Process Improvement: A Road to Success. In: Münch, J., Vierimaa, M. (eds.) PROFES 2006. LNCS, vol. 4034, pp. 395–401. Springer, Heidelberg (2006)
12. Niazi, M., Wilson, D., Zowghi, D.: Critical Success Factors for Software Process Improvement Implementation: An Empirical Study. In: Software Process Improvement and Practice 11(2), 193–211 (2006)
13. Dybå, T.: An Empirical Investigation of the Key Factors for Success in Software Process Improvement. *IEEE Trans. Software Eng.* 31(5), 410–424 (2005)
14. Montoni, M., Santos, G., Rocha, A.R., Figueiredo, S., Cabral, R., Barcellos, R., Barreto, A., Soares, A., Cerdeiral, C., Lupo, P.: Taba Workstation: Supporting Software Process Deployment based on CMMI and MR-MPS.BR. In: Lecture Notes of Computer Science (LNCS), presented at the 7th Int. Conference on Product Focused Software Process Improvement, Amsterdam, The Netherlands, pp. 249–262 (June 2006)
15. ISO/IEC 12207:2000 - Information technology – software process life cycle (2000)
16. Santos, G., Montoni, M., Rocha, A.R., Figueiredo, S., Mafra, S., Albuquerque, A., Paret, B.D., Amaral, M.: Using a Software Development Environment with Knowledge Management to Support Deploying Software Processes in Small and Medium Size Companies. In: 3rd Conf. Prof. Know. Manag. Exp. and Visions, Kaiserslautern, Germany, vol. 10. pp. 72–76 (April 10-13, 2005)

17. Montoni, M., Santos, G., Villela, K., Miranda, R., Rocha, A.R., Travassos, G.H., Figueiredo, S., Mafra, S.: Knowledge Management in an Enterprise-Oriented Software Development Environment. In: Karagiannis, D., Reimer, U. (eds.) PAKM 2004. LNCS (LNAI), vol. 3336, pp. 117–128. Springer, Heidelberg (2004)
18. Figueiredo, S., Santos, M., Montoni, R., Rocha, A.R., Barreto, A., Barreto, A., Ferreira, A.: Taba Workstation: Supporting Technical Solution Through Knowledge Management of Design Rationale. In: Reimer, U., Karagiannis, D. (eds.) PAKM 2006. LNCS (LNAI), vol. 4333, pp. 61–71. Springer, Heidelberg (2006)
19. Krasner, H.: Accumulating the Body of Evidence for The Payoff of Software Process Improvement, Software Process Improvement, IEEE, pp. 519–539 (2001)
20. Lindvall, M., Frey, M., Costa, P., et al.: Lessons Learned about Structuring and Describing Experience for Three Experience Bases. In: Althoff, K.-D., Feldmann, R.L., Müller, W. (eds.) LSO 2001. LNCS, vol. 2176, pp. 106–119. Springer, Heidelberg (2001)
21. Oh, E., Hoek, A.: Adapting Game Technology to Support Individual and Organizational Learning. In: Proceedings of SEKE'2001, Buenos Aires, pp. 347–362 (June 2001)
22. Kucza, T., Nattinen, M., Parviainen, P.: Improving Knowledge Management in Software Reuse Process. In: Bomarius, F., Komi-Sirviö, S. (eds.) PROFES 2001. LNCS, vol. 2188, pp. 141–152. Springer, Heidelberg (2001)
23. Holz, H., Konnecker, A., Maurer, F.: Task Specific Knowledge Management in a Process Centered SEE. In: Althoff, K.-D., Feldmann, R.L., Müller, W. (eds.) LSO 2001. LNCS, vol. 2176, pp. 163–177. Springer, Heidelberg (2001)
24. Maurer, F., Holz, H.: Process-Centered Knowledge Organization for Software Engineering. In: Papers of the AAAI-99 Workshop on Exploring Synergies of Knowledge Management and Case-Based-Reasoning, Orlando, Florida, AAAI Press, Stanford (1999)
25. Decker, B., Althoff, K.-D., Nick, M., Tautz, C.: Integrating Business Process Descriptions and Lessons Learned with an Experience Factory. In: Professionelles Wissensmanagement – Erfahrungen und Visionen (Beiträge der 1. Konferenz für Professionelles Wissensmanagement), Schnurr, H.-P., Staab, S., Studer, R., Stumme, G., Sure, Y. (eds.) Baden-Baden, Germany, Shaker Verlag, Aachen (March 2001)
26. Rocha, A.R., Montoni, M., Santos, G., Oliveira, K., Natali, A.C., Mian, P., Conte, T., Mafra, S., Barreto, A., Albuquerque, A., Figueiredo, S., Soares, A., Bianchi, F., Cabral, R., Dias Neto, A.: Success Factors and Difficulties in Software Process Deployment Experiences based on CMMI and MR-MPS. In: Proceedings of 8th Workshop on Learning Software Organizations LSO'2006, Rio de Janeiro, Set pp. 77–87 (September 2006)

Organisational Readiness and Software Process Improvement

Mahmood Niazi¹, David Wilson², and Didar Zowghi²

¹ School of Computing and Mathematics, Keele University, ST5 5BG, UK
mkniazi@cs.keele.ac.uk

² Faculty of Information Technology, University of Technology Sydney,
NSW 2007, Australia
{davidw,didar}@it.uts.edu.au

Abstract. The Capability Maturity Model Integration (CMMI) is a structured representation of software development processes that can support an organisation's software process improvement (SPI) strategies. However, CMMI and SPI initiatives generally exhibit low levels of adoption and limited success. One of the major reasons for these shortcomings is that many organisations undertake SPI initiatives without knowing whether or not they are ready to undertake them. Our previous research has enabled us to develop a software process improvement readiness model/framework to address this problem.

This paper reports on the implementation of the SPI readiness model in three large-scale case studies. We have found that organisations with higher CMMI levels are more ready for SPI initiatives than organisations with low CMMI levels. We suggest that organisations at higher CMMI levels have developed capabilities that enable them to further leverage SPI than organisations at lower CMMI levels.

Keywords: Software Process Improvement, Case Study, Organisational Readiness.

1 Introduction

Software Process Improvement (SPI) has been a long-standing approach promoted by software engineering researchers, intended to help organisations develop higher-quality software more efficiently. Process capability maturity models such as CMM, CMMI [1] and ISO/IEC 15504 (SPICE) are SPI frameworks for defining and measuring processes and practices that can be used by software developing organisations. However, only a small number of software organisations have successfully adopted SPI. SPI initiatives exhibit low levels of adoption and limited success [2]. Deployment is often not only multi-project, but multi-site and multi-customer and the whole SPI initiative typically requires a long-term approach. It takes significant time to fully implement an SPI initiative [3]. A recent report of the Software Engineering Institute shows the number of months needed in order to move from one maturity level of CMM to the next one [3]:

- Maturity level 1 to 2 is 22 months
- Maturity level 2 to 3 is 19 months
- Maturity level 3 to 4 is 25 months
- Maturity level 4 to 5 is 13 months

Such time frames mean that the SPI approach is often considered an expensive challenge for many organizations [2] as they need to commit significant resources over an extensive period of time. Even organisations who are willing to commit the resources and time do not always achieve their desired results. The failure rate of SPI initiatives is very high, estimated as 70% [4; 5]. The significant investment and limited success are reasons for many organisations being reluctant to embark on a long path of systematic process improvement.

In order to improve the SPI implementation process, in our previous research, we have developed a SPI implementation readiness model [6]. The objective of the SPI readiness model is to assist organisations in assessing and improving their SPI implementation readiness. In this paper we report on our evaluation of the readiness model in three large scale case studies. The objective of this evaluation is to further improve the readiness model and to observe the correlation between organisation readiness and SPI maturity.

In this paper we have addressed the following research question:

RQ: Are organisations in higher CMM(I) levels more ready for SPI implementation than organisations in lower CMM(I) levels?

This paper is organised as follows. Section 2 describes the background. Section 3 describes the research design. In Section 4 findings are presented and analysed. Discussion is provided in Section 5. In Section 6 case study validity is discussed. Section 7 provides the conclusion.

2 Background

Despite the importance of the SPI implementation process, little empirical research has been carried out on developing ways in which to effectively implement SPI programmes [2; 7]. Much attention has been paid to developing standards and models for SPI. Also, organisations typically adopt ad hoc methods instead of standard, systematic and rigorous methods in order to implement SPI initiatives [8]. This risk can lead organisations to a chaotic situation with no standard for SPI implementation practices. In the appraisal of SPI models, e.g. CMMI, the software process maturity of the organisations is assessed. Little attention, however, has been paid to assess the SPI implementation maturity/ readiness of the organisations. The assessment of SPI implementation maturity/ readiness can help organisations in successfully implementing SPI initiatives. This is because the readiness of the organisations for successfully implementing SPI initiatives could be judged through this SPI implementation maturity. We have focused on these issues and developed a SPI readiness model (as shown in Figure 1) in order to assess the SPI implementation maturity/ readiness of the organisations [6]. The CMMI perspective [1] and the findings from our previous empirical study [9; 10] were used in the design of the SPI readiness model. The SPI

readiness model has four SPI implementation maturity/ readiness levels abstracted from CMMI. These maturity levels contain different critical success factors (CSFs) [11] and critical barriers (CBs) [10] identified through the literature and interviews. Under each factor, different practices have been designed that guide how to assess and implement each factor.

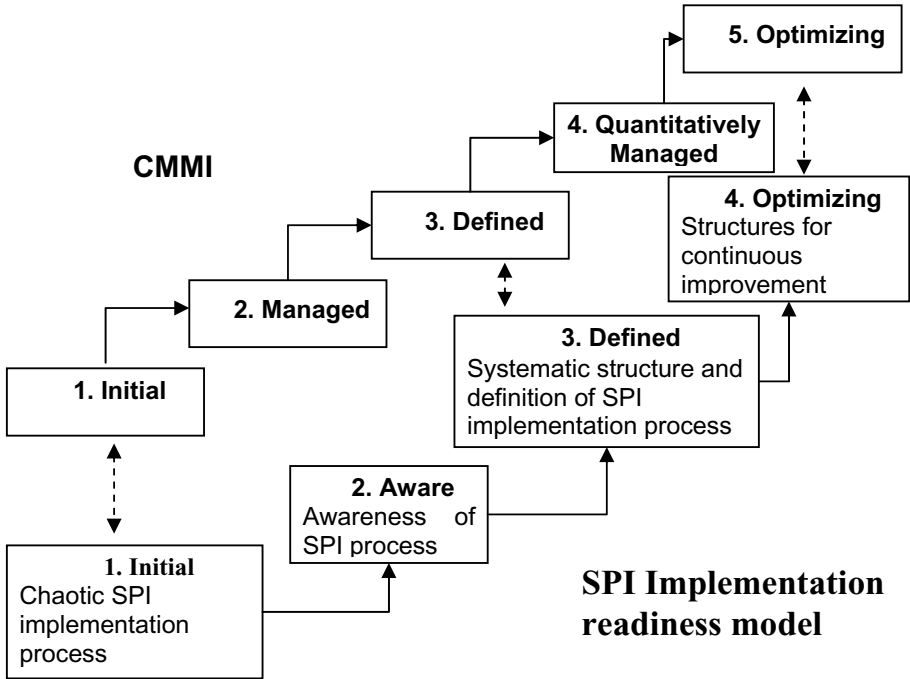


Fig. 1. SPI implementation readiness model [6]

3 Study Design

The case study method was used because this method is said to be powerful for evaluation and can provide sufficient information in the real software industry environment [12]. The case study also provides valuable insights for problem solving, evaluation and strategy [13]. Since the SPI readiness model is more applicable to a real software industry environment, the case study research method is believed to be a more appropriate method for this situation.

Real life case studies were necessary because they:

- Showed that the SPI readiness model is suitable or will fit in the real world environment.
- Highlighted areas where the SPI readiness model needs improvement.
- Showed the practicality and usability of the SPI readiness model use.

To provide more confidence in this study, three separate case studies were conducted at three different organisations. Organisations were selected for case studies because they provided especially rich descriptions of their SPI efforts and because they agreed to release the case studies results.

Initially, we talked to each participant face-to-face, explained what the case study was about and handed out a hard copy of the SPI readiness model. The participants also contacted us through emails to solicit more information about the use of the SPI readiness model. One participant from each organisation, who was the key member of SPI team, was involved in each case study. The key participant communicated with us through email and face-to-face discussion for one month in order to get a thorough understanding of the SPI readiness model. Different components of the SPI readiness model were explained and participants were encouraged to use this model independently.

In each case study, a participant used the SPI readiness model and assessed the SPI implementation readiness of his/her organisation independently without any suggestion or help from the researchers. At the end of each case study, an interview was conducted with the participant in order to provide feedback about the SPI readiness model. A questionnaire (available from the authors) was used as a means to structure this feedback session. This questionnaire is divided into four parts: demo-graphic, ease of learning, user satisfaction and structure of the SPI readiness model. Each feedback session was an informal discussion and the questionnaire was filled out by each participant. Each questionnaire was analysed qualitatively.

4 Findings

The three organisations in our case study are called “Organisation A”, “Organisation B” and “Organisation C”. The assessment process for SPI implementation readiness is described in [6].

4.1 SPI Implementation Readiness of Organisation A

Organisation A is an international organisation that provides consultancy and information technology services to both the private and public sector, employing 10,000 professionals in Asia Pacific, Canada, Europe and United States. The main purpose of the organisation is to enhance the efficiency and effectiveness of the Information Systems prevailing in the public and private sectors by applying relevant state-of-the-art technologies related to computer software, hardware and data communication.

The following are some of the major areas in which Organisation A can provide services to its clients:

- E-Business
- Enterprise Consulting
- Technology Consulting

- Solution Delivery
- Application Portfolio Management/ Outsourcing
- Project Management

An SPI initiative was initiated seven years ago in Organisation A. The reasons for initiating the SPI programme were:

- To reduce software development cost
- To improve management visibility in software development
- To increase productivity
- To improve the quality of the software developed
- To meet customer requirements

The SPI programme was initiated by the research division of Organisation A. The research division has developed a standard methodology for software development. During the development of this methodology special attention was given to the requirements of the ISO 9001 standard and the CMM model. Organisation A is ISO 9001 certified and is currently assessed at CMM level 3.

The assessment results of Organisation A are summarised in Table 1. The key points of this assessment are as follows:

- It is clear that Organisation A stands at Level-1 'Initial' of the SPI readiness model because two factors of Level-2 'Aware' are not fully implemented in Organisation A. In order to achieve any maturity level it is important that all the CSFs and CBs that belong to that maturity level should have been fully implemented. Table 1 shows that in order to achieve Level-2 'Aware' the Organisation A needs to improve two factors, i.e. senior management commitment and staff involvement. Similarly, in order to achieve Level-3 'Defined' the Organisation A needs to improve 3 factors, i.e. Creating process action teams, Staff time and resources and Time pressure.
- It shows that Organisation A has well defined training and SPI awareness programmes.
- SPI activities have been assigned to experienced staff members.
- It is clear that a defined SPI implementation methodology is in use and Organisation A managed to avoid organizational politics. The defined SPI implementation methodology could be the reason that this organisation was assessed in CMM Level-3.
- Organisation A has established some processes in order to review the implementation processes

Organisation A is a relatively high maturity organisation with CMM Level-3. It is surprising to see that Organisation A has not successfully implemented these factors such as 'senior management commitment', 'staff involvement', 'creating process action teams', 'staff time and resources' and 'time pressure'. As these factors are weak in organisation A therefore this organisation stands in Level-1 'Initial' of SPI readiness model.

Table 1. SPI implementation readiness of organisation A

Readiness Level	Critical success factors and barriers	Status
Level-2 Aware	Senior management commitment	weak
	Training and mentoring	strong
	Staff involvement	weak
	Awareness of SPI	strong
	Lack of support	strong
Level-3 Defined	Creating process action teams	weak
	Experienced staff	strong
	Staff time and resources	weak
	Defined SPI implementation methodology	strong
	Time pressure	weak
	Organizational politics	strong
Level-4 Optimising	Reviews	strong

4.2 SPI Implementation Readiness of Organisation B

Organisation B is an international organisation that provides consultancy and information technology services to both the private and public sector, employing more than 2000 professionals in Australia and worldwide. The core business of the organisation is to provide services in software development, system integration, business innovation and business process improvement.

The following are some of the major areas in which Organisation B can provide services to its clients:

- Business and IT services
- Business consulting services
- Infrastructure services
- Financing
- E-Business
- Project Management

The organisation delivers complex software systems to a number of clients. The SPI programme was initiated five years ago in Organisation B. The main reasons for initiating the SPI programmes were to:

- Reduce development cost and time to market
- Increase productivity and quality of the product

Organisation B adopted a CMM model for its SPI programme. According to self assessment results, the organization's process maturity was found to be in CMM

Level 1. The process teams undertook different SPI actions in order to achieve level 2, i.e. working on requirements management, software project planning and software quality assurance etc. Using CMM-based assessment in 2001, the process maturity was found to be in CMM level 2 with traces of Level 3. Now organisation B is working to achieve level 3.

The SPI implementation readiness assessment results of Organisation B are summarised in Table 2:

- It is clear that Organisation B stands at Level-1 ‘Initial’ because four factors of Level-2 ‘Aware’ are not fully implemented in the Organisation B. Table 2 shows that in order to achieve Level-2 ‘Aware’ the Organisation B needs to improve four factors, i.e. training and mentoring, staff involvement, awareness of SPI and lack of support.
- In order to achieve Level-3 ‘Defined’ and Level-4 ‘Optimising’ the Organisation B needs to improve five factors, i.e. creating process action teams, experienced staff, staff time and resources, time pressure and reviews.
- It also shows that the Organisation B has adequate senior management support for SPI programmes.
- Table 2 shows that a defined SPI implementation methodology is in use and Organisation B managed to avoid organisational politics.

Table 2. SPI implementation readiness of organisation B

Readiness Level	Critical success factors and barriers	Status
Level-2 Aware	Senior management commitment	strong
	Training and mentoring	weak
	Staff involvement	weak
	Awareness of SPI	weak
	Lack of support	weak
Level-3 Defined	Creating process action teams	weak
	Experienced staff	weak
	Staff time and resources	weak
	Defined SPI implementation methodology	strong
	Time pressure	Weak
Level-4 Optimising	Organizational politics	Strong
Level-4 Optimising	Reviews	Weak

4.3 SPI Implementation Readiness of Organisation C

Organisation C provides telecommunication services and employs more than 2000 professionals in Australia and worldwide. The core business of the organisation is to

provide cutting-edge communications, information and entertainment services. The organisation provides a broad range of communications services including mobile, national and long distance services, local telephony, international telephony, business network services, Internet and satellite services and subscription television.

The SPI programme was initiated in Organisation C three years ago. The reasons for initiating the SPI programmes were:

- To reduce software development cost
- To reduce time-to-market
- To increase productivity
- To improve the quality of the software developed
- To automate the production of relevant development documentation

In 2002 this Organisation C was assessed at CMM level 2. The SPI implementation readiness assessment results of Organisation B are summarised in Table 3:

Table 3. SPI implementation readiness of organisation C

Readiness Level	Critical success factors and barriers	Status
Level-2 Aware	Senior management commitment	Weak
	Training and mentoring	Weak
	Staff involvement	Strong
	Awareness of SPI	Weak
	Lack of support	Weak
Level-3 Defined	Creating process action teams	Strong
	Experienced staff	Weak
	Staff time and resources	Weak
	Defined SPI implementation methodology	Strong
	Time pressure	Weak
	Organizational politics	Weak
Level-4 Optimising	Reviews	Weak

- It is clear that Organisation C stands in Level-1 'Initial' because four factors of Level-2 'aware' are not fully implemented in the Organisation C.
- In order to achieve Level-3 'Defined' and Level-4 'Optimising' the Organisation C needs to improve five factors, i.e. experienced staff, staff time and resources, time pressure, organizational politics and reviews.
- It also shows that the Organisation C has experienced staff for SPI programmes.
- A defined SPI implementation methodology is in use and Organisation C has created teams for SPI activities.

5 Discussion

The CMMI framework is structured into five maturity levels ranging from level 1 to 5. Each maturity level expresses a different state of software development maturity in an organisation. Level-1 corresponds to the lowest state of software development maturity while level-5 corresponds to the highest state of software development maturity. We argue that higher levels of CMMI (level 3 and above) indicate that the organisation has well defined processes for the implementation of SPI initiatives. This is because the organisation has successfully implemented CMMI. While lower levels of CMMI (level 2 and below) indicate that the organisation does not have well defined processes for the implementation of SPI initiatives. This is because the organisation is struggling to successfully implement CMMI. Keeping in view these points, the organisations in higher CMMI levels should have less implementation issues than organisations in lower CMMI levels.

In order to address above points our research question was:

RQ: Are organisations in higher CMM(I) levels more ready for SPI implementation than organisations in lower CMM(I) levels?

In order to address this research question, it is important to compare the results of the three case studies. As discussed earlier Organisation A is at CMM Level-3 and Organisations B and C are at CMM Level-2 respectively. The results of the three case studies are summarised into Table 4.

As discussed in Section 4 all organisation were assessed at Level-1 Aware of SPI implementation readiness model. However, by looking at weak implementation factors we have noticed the following differences:

- Organisation A has only two weak factors in level-2, while Organisations B and C have four weak factors in Level-2.
- For Level-3, Organisation A has three weak factors while Organisations B and C have four weak factors.
- Table 4 shows that Organisation A has successfully implemented more implementation factors than Organisations B and C. This also shows that Organisation A has less weak implementation factors (i.e. five) than Organisations B and C (i.e. nine).
- It shows that 78% (i.e. seven factors) of the weak factors are common between CMM Level-2 organisations B and C.

These findings have confirmed our research question that organisations with higher CMMI levels are more ready than organisations with low CMMI levels.

Comparison of weak factors of the three organisations provides evidence that there are some clear similarities and differences between the findings of the three data sets. The factors 'time pressure' and 'staff time and resources' are common among three organisations. This shows that organisations both at lower and higher levels of CMMI need to improve these two common factors. In the literature different studies have discussed 'time pressure' and 'staff time and resources' as barriers for SPI implementation. For example, Baddoo and Hall [14] present empirical findings analysing what de-motivates UK practitioners in SPI. The authors have separated senior managers, project managers and developers into separate focus groups. The

Table 4. Summary of results of organisations A, B and C

Assessment issue	Organisation A (CMM Level-3)	Organisation B (CMM Level-2)	Organisation C (CMM Level-2)
Weak implementation factors in SPI readiness model Level-2 'Aware'	<ul style="list-style-type: none"> • Senior management commitment • Staff involvement 	<ul style="list-style-type: none"> • Awareness of SPI • Lack of support • Staff involvement • Training and mentoring 	<ul style="list-style-type: none"> • Awareness of SPI • Lack of support • Senior management commitment • Training and mentoring
Weak implementation factors in SPI readiness model Level-3 'Defined'	<ul style="list-style-type: none"> • Creating process action teams • Staff time and resources • Time pressure 	<ul style="list-style-type: none"> • Creating process action teams • Experienced staff • Staff time and resources • Time pressure 	<ul style="list-style-type: none"> • Experienced staff • Staff time and resources • Time pressure • Organizational politics
Weak implementation factors in SPI readiness model Level-4 'Optimising'	<ul style="list-style-type: none"> • Nil 	<ul style="list-style-type: none"> • Reviews 	<ul style="list-style-type: none"> • Reviews
Total Weak implementation factors	5	9	9

authors state that all the groups of practitioners have cited time pressure as a demotivator for SPI, i.e. 62% of developers cited, 44% of project managers cited and 58% of senior managers cited. In the study of Goldenson and Herbsleb [7] "almost three-quarters (72%) report that process improvement has often suffered due to time and resource limitations". Paulish and Carleton [15] also describe case studies for SPI measurement and illustrate time restriction as one of the SPI implementation problem.

Table 4 also shows factors that are common between organisations of lower CMMI level (i.e. CMM level 2). For example, the organisations at lower CMM level are having problems of 'awareness of SPI', 'experienced staff', 'lack of support', 'training and mentoring' and 'reviews'. These factors need to be addressed in order to successfully implement SPI initiatives.

6 Case Study Validity

Two types of threats to case study validity are relevant to this study: construct validity and external validity [16]. Construct validity is concerned with whether or not the measurement scales represent the attributes being measured. The attributes are taken from a substantial body of previous research [17; 18] and further studies conducted by one of the authors [6]. The responses from the post case study questionnaire show that all the attributes considered were relevant to their workspace. Also, all participants agreed with the assessment results.

External validity is concerned with the generalisation of the results to other environments than the one in which the initial study was conducted [19]. Since three case studies were conducted, it is hard to justify the external validity at this stage. However, since SPI readiness model's evaluation is currently limited to only the three organisations reported in this study, generalisation to whole software industry should be considered with extreme caution.

7 Conclusion

In this research a case study method was chosen because the SPI implementation readiness model is more applicable to the real software industry environment. Three separate case studies were conducted at three different companies. The results of the case studies show that the SPI implementation readiness model is not only significant in the theoretical work but also significant in the real world environment as each of the three companies was able to successfully use the SPI implementation readiness model to assess their SPI implementation readiness. The participants have noticed the SPI implementation issues that the SPI implementation readiness model has identified for their companies and they were agreed with those issues.

We have found that organisations with higher CMMI levels were more ready for SPI initiatives than organisations with low CMMI levels. We suggest that readiness for SPI is directly associated with organisations' software development maturity. We have also found some clear similarities and differences between the findings of three case studies. For successful SPI initiatives, the organisations both at lower and higher levels of CMMI need to facilitate their staff members from time pressure and need to allocate required resources for SPI activities. We found that the organisations at lower CMM level are having problems of 'awareness of SPI', 'experienced staff', 'lack of support', 'training and mentoring' and 'reviews'. These factors need to be addressed in order to successfully implement SPI initiatives.

References

1. SEI: Capability Maturity Model® Integration (CMMISM), Version 1.1. SEI, CMU/SEI-2002-TR-029, Software Engineering Institute, USA (2002)
2. Leung, H.: Slow change of information system development practice. *Software quality journal* 8(3), 197–210 (1999)
3. SEI: Process Maturity Profile. Software Engineering Institute Carnegie Mellon University (2004)

4. SEI: Process maturity profile of the software community. Software Engineering Institute (2002)
5. Ngwenyama, O., Nielsen, P.: A Competing values in software process improvement: An assumption analysis of CMM from an organizational culture perspective. *IEEE Transactions on Software Engineering* 50, 100–112 (2003)
6. Niazi, M., Wilson, D., Zowghi, D.: A Maturity Model for the Implementation of Software Process Improvement: An empirical study. *Journal of Systems and Software* 74(2), 155–172 (2005)
7. Goldenson, D.R., Herbsleb, J.D.: After the appraisal: A systematic survey of Process Improvement, Its benefits, And Factors That Influence Success. SEI, CMU/SEI-95-TR-009, Software Engineering Institute, USA (1995)
8. Zahran, S.: Software process improvement - practical guidelines for business success. Addison-Wesley, London (1998)
9. Niazi, M., Wilson, D., Zowghi, D.: Critical success factors and critical barriers for software process improvement: An analysis of literature. In: the proceedings of Australasian Conference on Information Systems (ACIS03), Perth, Australia (2003)
10. Niazi, M., Wilson, D., Zowghi, D.: Critical Barriers for SPI Implementation: An empirical study. In: IASTED International Conference on Software Engineering (SE, 2004) Austria pp. 389–395 (2004)
11. Niazi, M., Wilson, D., Zowghi, D.: Critical Success Factors for Software Process Improvement: An Empirical Study. *Software Process Improvement and Practice Journal* 11(2), 193–211 (2006)
12. Yin, R.K.: Applications of Case Study Research. Sage Publications, Thousand Oaks (1993)
13. Cooper, D., Schindler, P.: Business research methods, 7th edn. McGraw-Hill, New York (2001)
14. Baddoo, N., Hall, T.: De-Motivators of software process improvement: An analysis of practitioner's views. *Journal of Systems and Software* 66(1), 23–33 (2003)
15. Paulish, D., Carleton, A.: Case studies of software process improvement measurement. *IEEE Computer* 27(9), 50–59 (1994)
16. Briand, L., Wust, J., Lounis, H.: Replicated Case Studies for Investigating Quality Factors in Object Oriented Designs. *Empirical Software Engineering* 6(1), 11–58 (2001)
17. Daskalantonakis, M.K.: Achieving higher SEI levels. *IEEE Software* 11(4), 17–24 (1994)
18. Beecham, S., Hall, T., Rainer, A.: Building a requirements process improvement model. Department of Computer Science, University of Hertfordshire, Technical report No: 378 (2003)
19. Regnell, B., Runeson, P., Thelin, T.: Are the Perspectives Really Different-Further Experimentation on Scenario-Based Reading of Requirements. *Empirical Software Engineering* 5(4), 331–356 (2000)

Software Process Improvement Through Teamwork Management

Esperança Amengual and Antònia Mas

Department of Computer Science. University of the Balearic Islands
Ctra. De Valldemossa, Km. 7.5. 07122 - Palma de Mallorca, Spain
{eamengual, antonia.mas}@uib.es

Abstract. In modern organizations teamwork is considered a key factor for succeeding in business. A growing emphasis on team culture culminates with a great number of articles analyzing different aspects to improve teamwork practises. Since software development projects are normally team efforts, teamwork improvement in software organizations should also be considered essential. In these companies, software process improvement programs based on international maturity standards are current issues in software engineering investigation. In this article, we firstly establish the teamwork key factors for succeeding in software development projects. Secondly, these key factors are analysed taking the ISO/IEC 15504 as a reference improvement framework.

1 Introduction

Nowadays, teamwork has become popular as a solution to a great number of companies main goal: producing to the lowest cost. Considering employees as the most important resource of an organization, teamwork is revealed as the most efficient way to achieve this objective. Modern organizations have expectations over their employees that go beyond work realisation to contribute to business success. As numerous articles demonstrate, there is a growing interest on team culture which refers to the ability of working successfully in a work team [1]. However, although the majority of companies consider teamwork ability as an important skill to measure to select their employees, a lot of work is still necessary to create a real teamwork culture.

Mother Nature provides us teamwork models, like ants and bees communities, where the final goal is achieved joining individual efforts. These natural organizations are good examples to follow that show that interdependence between team members is a key characteristic of successful teams [2]. However, this natural predisposition towards teamwork seems not to be as evident in the case of human beings. Teamwork is a work style that not all people are prepared to accept. Sometimes, individualistic work spirit can be an important obstacle to remove. One of the biggest problems in all companies is bringing together a group of people to accomplish a business goal, since all of them have different needs, interests, knowledge, experiences, expectations and motivations.

Although, we should not consider teamwork as the panacea, investigation on work groups formation and performance has been the centre of attention of different

specialists during the last two decades. This fact can be justified considering the important role that teams can play by performing effective tasks in an organization. Accordingly to Dr. Charles J. Margerison who states that "it is on the competency and effectiveness of teams that we depend", a lot has been said and written about individual competencies at work in contrast to the issue of team competency that has received little attention. Individual competencies are important, but they need to be seen in the context of what a team requires to perform well [3].

2 Teams in Software Projects

In software companies, the big demand on new systems together with the increment in their complexity make software development process to be considered a team activity. Thus, in these organizations in particular, interest on teamwork should not be an exception.

Some works demonstrate that this subject has not gone unnoticed. In [4], it is stated that coordination and communication in a software team are key aspects to be considered. In the same way, social interaction is also considered an important point for succeeding in software projects. In [5], its author, being in line with other articles also considered in this work, highlights that "it appears the human aspects of software development are more important than the technological aspects for better performance". To analyse this statement, the before mentioned article presents an investigation to explore the effects of personality on team productivity. In particular, the study seeks to determine the effect of the project leader's personality and the effect of team members' personalities on team performance.

Considering the human aspects as a key factor to control in a software development team as well, it is possible to find other publications. In [6] it is demonstrated that team roles described by R. Meredith Belbin [7] are useful to improve the effectiveness of software development teams. In this article, three software development teams working in different environment are analysed using the Belbin's questionnaire as instrument to gather data from individuals to analyze the teams. In [8] their authors present an experiment to demonstrate the utility of forming teams based on Belbin's team roles. The overall research focuses on the utility of Belbin's roles for team performance improvement. This experiment explores Belbin's Plant, who adds innovation and new ideas to teams. The specific conclusion is that Belbin's questionnaire is useful to identify characteristics of team members that can be used to make teams perform better.

In [9] its author points out that the majority of problems in software projects "are due to people problems, not technical ones". Although producing quality software is a technical activity, software is produced by people. Different maturity models and process models have been proposed, but problems still continue.

The different investigations mentioned in this section state of one or another way that it is necessary to consider specific teamwork aspects to be successful in a software development project.

3 Teamwork in Maturity Models

The Software Engineering Institute (SEISM), after developing the Capability Maturity Model as a descriptive model of the characteristics of an organization at a particular level of software process maturity [10], has developed the Team Software ProcessSM (TSPSM), a prescriptive model for software development teams. As it is defined in the SEI technical report which relates the TSP to the CMM [11], "TSP is a high-maturity process for project teams. It contains an adaptable set of processes, procedures, guidelines, and tools for project teams to use in the production of high-quality software on time and on budget". In [12] some results from projects that have adopted the TSP are provided. The results show that TSP teams are delivering essentially defect-free software on schedule, while improving productivity

In [13] the relationship between the Capability Maturity Model and the Team Software Process as complementary technologies is examined and the degree to which the CMM is addressed by the TSP is analysed. Other published articles [14, 15] show the usefulness of the TSP to reach a specific maturity level of the CMM.

With regard to our experience of software process improvement in eight small and medium enterprises of our environment [16, 17], we agree that teamwork improvement could be a key factor to better software development processes. Following our investigation into the applicability of the ISO/IEC 15504 on software small and medium enterprises [18], in this article the way in which teamwork aspects are considered by this standard is analysed. In order to do that, firstly the key teamwork factors for succeeding in a software project are established. Secondly, from an exhaustive analysis of the standard, the degree to which these factors are considered explicitly or implicitly by the standard is determined.

4 Teamwork Key Factors

Although the majority of software projects are performed by professional teams, technological aspects usually receive more attention than team dynamics in a process improvement initiative. From our research work in teamwork and, more concretely, in software teamwork, and from our own experience, in this section we establish the characteristics we consider every software development team should possess.

4.1 Team Management

As it is exposed in [19], when people work in groups, there are two quite separate issues to consider. The first is the task involved in getting the job done. Frequently this is the only issue which the group considers. The second is the process of the group work itself: the mechanisms by which the group acts as a unit and not as a loose rabble.

In accordance with this author, we think that teams must be considered an important resource that needs to be managed like it is done with all resources in a project. Then, for the good performance of the team, we consider important:

- Planning. Definition of objectives and tasks
- Monitoring. Control that goals are met accordingly with the defined schedule.

These two aspects, basic in all management, must be considered in particular for each one of the members of the team, as well as for the team as an entity.

4.2 Coordination

Accordingly with [4] coordination and efficiency translate to successful teamwork. These two aspects of teamwork can be specifically noted in the definition of teamwork as "the work of a team with reference to coordination of effort and to collective efficiency". Presumably, efficiency is an expected outcome of coordination effort. So it is relevant to consider how to achieve coordination.

Successful coordination requires that each team member understands:

- Who is responsible for what parts of the project.
- The relations between the tasks assigned to different team members.
- How the work is progressing with respects to the set schedule.

4.3 Effective Communication

Communication is probably the most essential component of teamwork [4]. Accordingly with [2], where ineffective communication is identified as one of the six factors that do not support teamwork, we think that it is necessary to explicitly define a communication mechanism. This can be performed in different manners:

- Project meetings (weekly, biweekly or other appropriately scheduled project meeting).
- Team members can provide written progress reports summarizing project status as it relates to timelines, expectations and other related criteria.
- Informal communication to keep team members connected.

4.4 Team Composition

Several studies about the improvement of work team effectiveness are based on team composition as a key factor that can affect project performance. One of the most important contributions to the analysis of team performance in organizations is the identification of the different roles in a team. A relevant work in this area is Belbin's Role theory where eight team roles are identified. In [20] a team model designed to improve performance in a software development team is described. This model describes an approach to structure people and their activities to enable project success. It is based on six key quality goals that drive the team and the associated roles.

Then, to build a good team it is desirable:

- Identify the roles to perform the different tasks.
- Determine the most suitable people for each role.
- Assign responsibilities.

4.5 Motivation

In software small and medium enterprises work teams are usually small groups in which human factor is crucial for succeeding. In [21] motivation is considered

essential for the effectiveness of the team. Although it is not easy, it is necessary to maintain enthusiasm and commitment from the team.

Motivation in a work team can be achieved by considering the following people motivators for each member of the team:

- Responsibility
- Interest in the assigned tasks
- Achievement of the targets
- Advancement
- Recognition of the work done

5 ISO/IEC 15504. Teamwork Aspects

ISO/IEC 15504 is an international standard that is appropriate across all application domains and sizes of organization and provides a structured approach for the assessment of processes with the objective of understanding the state of these processes for process improvement.

Part 5 of the standard, *An exemplar process assessment model based upon ISO/IEC 12207 Amd 1 & 2* [22], published on March, 2006, provides an example of a Process Assessment Model for use in performing a conformant assessment in accordance with the requirements of ISO/IEC 15504-2, *Performing an assessment* [23].

Process capability measure, defined in Part 2 of the standard, is based on nine process attributes. These attributes are used to determine if a process has reached a particular capability level. Each attribute measures a particular aspect of the capability of the process. To measure the degree of achievement of these attributes the standard considers different base practises for each one of them.

After an accurate revision of all the generic practises in each one of the six levels defined by the standard, we want to analyse if each teamwork key factor is considered among these generic practises. Next, results of this analysis are introduced.

5.1 Team Management

Planning aspects, more concretely, human resources identification and responsibilities definition are considered in the generic practises proposed by the standard to measure level 2 which refers to a process implemented in a managed fashion.

Level 3, *Established process*, ensures that the managed process is implemented using a defined process. Allocation of human resources to support the performance of the defined process is explicitly considered in this level.

In level 5, *Optimizing process*, commitment aspects, both at organizational level and process owner level, are considered.

Moreover, the standard highlights the human factors that impact the effectiveness and full deployment of the process agreed changes. In particular, it considers commitment, organizational culture and risks as important management factors.

Table 1 shows a summary of team management aspects considered by the standard in the different capability levels.

Table 1. Team management factor

ISO/IEC 15504 Generic Practices	Team Management
Level 2 assessment	
Define responsibilities and authorities for performing the process	Define responsibilities
Identify and make available resources to perform de process according to plan	Identify human and infrastructure resources
Level 3 assessment	
Provide resources and information to support the performance of the defined process	Make available, allocate and use required human resources
Level 5 assessment	
Define an implementation strategy based on long-term improvement vision and objectives	Organizational management and process owner(s) demonstrate commitment to improvement
Manage the implementation of agreed changes to selected areas of the defined and standard process according to the implementation strategy	Identify and manage: commitment, organizational culture and risks

5.2 Coordination

Although coordination aspects can not be directly observed from the practises proposed by the standard, it is possible to deduce that some aspects are specifically considered in some levels, as it is shown in Table 2.

Table 2. Coordination factor

ISO/IEC 15504 Generic Practices	Team coordination
Level 2 assessment	
Manage the interfaces between involved parties	Manage interfaces between the involved parties
Level 4 assessment	
Establish quantitative objectives for the performance of the defined process, according to the alignment of the process with the business goals	Verify process performance objectives with process owner(s)
Collect product and process measurement results through performing the define process	Report measurement results to those responsible for monitoring that objectives are met
Level 5 assessment	
Manage the implementation of agreed changes to selected areas of the defined and standard process according to the implementation strategy	Identify and manage: conflict / cohesion goal consensus participation

5.3 Effective Communication

Accordingly with the ISO/IEC 15504 it is necessary to establish formal communication mechanisms to ensure that all participants in a process can obtain the necessary information to perform it. Table 3 demonstrates that communication is an essential aspect fully considered in the different capability levels.

Table 3. Effective communication factor

ISO/IEC 15504 Generic Practices	Team Communication
Level 2 assessment	
Define responsibilities and authorities for performing the process	Communicate responsibilities
Identify and make available resources to perform the process according to plan	Make information to perform the process available
Manage the interfaces between involved parties	Assure communication between the involved parties
Level 3 assessment	
Provide resources and information to support the performance of the defined process	Make available, allocate and use required information to perform the process
Level 4 assessment	
Analyse process and product measurement results to identify variations in process performance	Provide results to those responsible for tacking action
Level 5 assessment	
Manage the implementation of agreed changes to selected areas of the defined and standard process according to the implementation strategy	Communicate process changes to all affected parties
Evaluate the effectiveness of process change on the basis of actual performance against process performance and capability objectives and business goals	Make available a mechanism for documenting and reporting analysis results

5.4 Team Composition

Different team composition aspects, like role identification, participant determination and responsibility assignation, are present through all capability levels in the standard (Table 4).

Table 4. Team composition factor

ISO/IEC 15504 Generic Practices	Team Composition
Level 2 assessment	
Define responsibilities and authorities for performing the process	Assign responsibilities
Manage the interfaces between involved parties	Assign responsibilities. Determine individuals and groups involved in the process
Level 3 assessment	
Identify the roles and competencies for performing the standard process	Identify process performance roles. Identify competencies for performing the process
Assign and communicate roles, responsibilities and authorities for performing the defined process	Assign and communicate roles and authorities for performing the defined process
Ensure necessary competencies for performing the defined process	Identify appropriate competencies for assigned personnel
Identify process information needs in relation with business goals	Identify process stakeholders
Define responsibilities and establish infrastructure to collect product and process measures	Define responsibilities for data collection
Level 4 assessment	
Identify process information needs in relation with business goals	Identify process stakeholders
Define responsibilities and establish infrastructure to collect product and process measures	Define responsibilities for data collection
Level 5 assessment	
Manage the implementation of agreed changes to selected areas of the defined and standard process according to the implementation strategy	Identify and manage: skills, leadership, knowledge, ability

5.5 Motivation

This key factor is an aspect that the standard only considers at capability level 5, where job satisfaction, motivation and morale of the employees are measured as indicators of a process at the highest level in the dimension capability (Table 5).

Table 5. Motivation factor

ISO/IEC 15504 Generic Practices	Team Motivation
Level 5 assessment	
Manage the implementation of agreed changes to selected areas of the defined and standard process according to the implementation strategy	Identify and manage: job satisfaction, motivation and morale

6 Conclusions and Further Work

In this work, we have presented the first results of a research effort aimed to consider teamwork aspects as essential for succeeding in software process improvement programs.

Firstly, we have established the key teamwork factors we consider essential for a software development team. Then, considering these teamwork key factors in the ISO/IEC 15504 framework, we have observed and deduced that Team Composition and Effective Communication factors are considered by the Standard in all maturity levels. On the other hand, Team Management and Coordination, we consider as important factors as the before mentioned ones to be successful in a software project improvement initiative, are not being considered with the same detail.

Moreover, although we consider the Motivation factor an essential characteristic in a work team to reach any software process capability level, ISO/IEC 15504 only considers it explicitly in generic practices to obtain capability level 5.

However, to validate these preliminary results it is still necessary to demonstrate that improving teamwork can result in a real software process improvement. To do so, our future work would be oriented to the real application of a software process improvement program accordingly with the International Standard in small and medium enterprises, and focused on teamwork particular aspects.

Acknowledgements. The authors wish to thank the Comisión Interministerial de Ciencia y Tecnología, Spain, (under grant IN2GESOFT TIN2004-06689-C03-00) for supporting this research effort.

References

- [1] Tarricone, P., Luca, J.: Employees, teamwork and social interdependence - a formula for successful business? *Team Performance Management: An International Journal* 8(3/4), 54–59 (2002)
- [2] Scarnati, J.T.: On becoming a team player. *Team Performance Management: An International Journal* 7(1/2), 5–10 (2001)
- [3] Margerison, C.: Team Competencies. *Team Performance Management: An International Journal* 7(7/8), 122–177 (2001)
- [4] Miller, E.: Teamwork on the Job - An Essential Ingredient to Success. *IEEE-USA Today's Engineer Online* (available 07/12/2006) (2001), <http://www.todaysengineer.org/Careerfocus/sept01te/sept01features/teamwork.html>
- [5] Gorla, N., Wah Lam, Y.: Who Should Work with Whom? Building Effective Software Project Teams. *Communications of the ACM* 47(6), 79–82 (2004)
- [6] Mazhil, R.: Analysis of team effectiveness in software development teams working on hardware and software environment using Belbin Self-perception Inventory. *Journal of Management Development* 24(8), 738–753 (2005)
- [7] Belbin, R.: *Management Teams: Why they succeed or fail*. Elsevier Butterworth-Heinemann, Oxford (2004)
- [8] Stevens, K., Henry, S.: *Analysing Software Teams Using Belbin's Innovative Plant Role* (available 18/12/2006), <http://www.radford.edu/~kstevens2/ISTall.pdf>

- [9] Evans, I.: *Achieving Software Quality through Teamwork*. Artech House, Inc. Norwood (2004)
- [10] Paulk, M., et al.: *The Capability Maturity Model: Guidelines for Improving the Software Process*. Addison-Wesley, London (1995)
- [11] CMU/SEI-2002-TR-008. Relating the Team Software ProcessSM (TSPSM) to the Capability Maturity Model[®] for Software (SW-CMM[®]). Software Engineering Institute (2002)
- [12] CMU/SEI-2003-TR-014. The Team Software ProcessSM (TSPSM) in Practice: A Summary of Recent Results. Software Engineering Institute (2003)
- [13] Noopur, D.: Using the TSP to Implement the CMM. *CrossTalk. The Journal of Defense Software Engineering*, (September 2002 Issue) (available 10/07/2006), <http://www.stsc.hill.af.mil/crosstalk/2002/09/davis.html>
- [14] Hefley, B., Schwalb, J., Pracchia, L.: AV-8B's Experience Using the TSP to Accelerate SW-CMM Adoption. *CrossTalk. The Journal of Defense Software Engineering*, (September 2002 Issue) (available 27/12/2006), <http://www.stsc.hill.af.mil/crosstalk/2002/09/hefley.html>
- [15] Pracchia, L.: The AV-8B Team Learns Synergy of EVM and TSP Accelerates Software Improvement. *CrossTalk. The Journal of Defense Software Engineering*, (January 2004 Issue) (available 27/12/2006), <http://www.stsc.hill.af.mil/crosstalk/2004/01/0401pracchia.htm>
- [16] Amengual, E., Mas, A.: A New Method of ISO/IEC TR 15504 and ISO 9001:2000 Simultaneous Application on Software SMEs. In: *Proceedings of the Joint ESA - 3rd International SPICE Conference on Process Assessment and Improvement*, pp. 87–92 (March 2003)
- [17] Mas, A., Amengual, E.: A Method for the Implementation of a Quality Management System in Software SMEs. In: *Software Quality Management XII. New Approaches to Software Quality*, The British Computer Society, Great Britain (2004)
- [18] Mas, A., Amengual, E.: ISO/IEC 15504 Adaptation for Software Process Assessment in SMEs. In: *Proceedings of the International Conference on Software Engineering Research and Practice*, pp. 693–697 (June 2003)
- [19] Blair, G.M.: Groups that Work. *Engineering Management Journal* 1(5), 219–223 (1991)
- [20] MSF Team Model v. 3.1. Microsoft Corporation (2002)
- [21] Blair, G.M.: The Human Factor. *Engineering Management Journal* 2(5), 219–223 (1992)
- [22] ISO/IEC 15504-5:2006 Information technology – Process Assessment – Part 5: An exemplar Process Assessment Model. International Organization for Standardization (2006)
- [23] ISO/IEC 15504-2:2003. Information technology – Process assessment – Part 2: Performing an assessment. International Organization for Standardization (2003)

De-motivators of Software Process Improvement: An Analysis of Vietnamese Practitioners' Views

Mahmood Niazi¹ and Muhammad Ali Babar²

¹ School of Computing and Mathematics, Keele University, ST5 5BG, UK
mkniazi@cs.keele.ac.uk

² Lero, University of Limerick, Ireland
muhammad.alibabar@ul.ie

Abstract. We have conducted face-to-face questionnaire based survey sessions with twenty-three Vietnamese software practitioners in order to determine software process improvement (SPI) de-motivators. The main objective of this study is to provide SPI practitioners with some insight into designing appropriate SPI implementation strategies and to maximize practitioners support for SPI.

We asked practitioners to choose and rank various SPI de-motivator against the five types of assessments (high, medium, low, zero or do not know). From this, we propose the notion of 'perceived value' associated with each SPI de-motivator. We have identified 'high' and 'medium' perceived values de-motivators that can undermine SPI initiatives. We have identified what de-motivates developers and managers to be actively involved in SPI initiatives. We have also identified SPI de-motivators of small-medium and large sized organisations.

1 Introduction

Software quality problems are widely acknowledged to affect the development cost and time [1; 2]. In order to reduce these problem, much attention has been paid to develop standards and models for SPI [3; 4]. However, the population of organisations that have adopted process capability maturity models is only a part of the entire population of software-developing organisations [5]. SPI initiatives exhibit low levels of adoption and limited success [6]. The recent report of Software Engineering Institute shows that on average organisations need 79 months to achieve CMMI Level-5 [7].

In order to successfully implement SPI standards and models, as researchers, we need to be constantly aware of what really de-motivates practitioners in real life. This will enable us to position our research within an appropriate context [8]. It is important to discover which de-motivators will undermine SPI implementation, as research shows that the SPI approach is often considered an expensive approach for many organisations [6], as they need to commit significant resources over an extensive period of time. Even the organisations, which are willing to commit the resources and time do not always achieve their desired results [9; 10]. The failure rate

of SPI initiatives is very high, estimated as 70% [11; 12]. The knowledge of SPI de-motivation may help us to develop new or improved SPI approaches, whose adoption will better match organisations' objectives, and also may help to communicate a compelling case to organisations making decisions about adopting SPI.

We have conducted a study with twenty-three software development practitioners; an understanding of the perceived value of each SPI de-motivator across different practitioners may help with more effective SPI implementation strategies. We believe that where respondents from different organisations identify a de-motivator as having a high perceived value then that de-motivator should be seriously considered for its importance in SPI initiatives. If different software development practitioners cite the same de-motivator, it is obviously important to the practitioners involved.

Previously, other researchers [13; 14] have also conducted studies to identify de-motivators of software development practitioners. Our research is aimed at not only extending the findings of those studies by conducting a similar study in a different culture; but also intending to expand this type of research by understanding the relative value of each identified de-motivator perceived by practitioners. We believe that software practitioners may associate different values to different SPI de-motivators. Moreover, it is also possible that SPI de-motivators may vary from one geographical region to another. As part of a large project about the SPI motivation, Keele University and National ICT Australia has been carrying out a research project to investigate the SPI motivators and de-motivators in the Asia-pacific region. The results of this project are expected not only to help software practitioners understand the usage of SPI de-motivators in the Asia-pacific region, but also help them compare SPI de-motivators identified in other regions [13; 14].

The contribution of this paper is to report the findings of one part of our research project aimed at identifying the factors that are perceived by Vietnamese software practitioners as SPI de-motivators. The findings of this research combined with the findings of the previous similar studies can shed some light on the de-motivators that should be considered critical when designing SPI implementation strategies.

There are three research questions that have motivated the work reported in this paper:

- RQ1. What SPI de-motivators have high and medium perceived values?
- RQ2. What de-motivates practitioners in order to implement SPI initiatives?
- RQ3. How are these de-motivators related to the size of organisations?

This paper is organised as follows. In Section 2 background is described. Section 3 describes the concept of perceived value. Section 4 describes the research design. In Section 5 findings are presented and analysed. Section 6 provides the summary and conclusion.

2 Background

McDermid and Bennet [15] have argued that the human factors to SPI have been ignored and this has impacted on effectiveness of SPI programmes. Hall and Wilson [16; 17] have also suggested that the experiences, opinions and perceptions of software practitioners impact indirectly on the quality of software produced. This also

implies that such attributes influence the attitudes of software practitioners towards SPI implementation approaches. These views, experiences and perceptions collectively will provide practitioners with sufficient knowledge about the nature of issues that play a positive or negative role in the implementation of SPI programmes and will assist them in effectively planning SPI implementation strategies.

A number of empirical studies have investigated factors that positively or negatively impact SPI, e.g. [14; 18-22]. To highlight few of these: in the survey of 138 individuals in 56 software organisations, Goldenson and Herbsleb [18], identified the factors necessary for implementing a successful SPI programme. Stelzer and Werner [19] determined ten factors that affect organisational change in SPI. Rainer and Hall [21] have conducted a questionnaire survey of UK companies and identified the key success factors that can impact on SPI implementation. Baddo and Hall reported SPI de-motivators in UK [14].

The work we report in this paper complements work previously done in above studies. However, our research also identifies the perceived value of each identified SPI de-motivator. In addition, creating solutions that are based on previous work may help to progress software improvement [23]. Moreover, our study has been conducted in a country that is increasing becoming an attractive destination for software development outsourcing from Western and Asian countries alike [24]. That is why we believe that there is a need of shedding some light on the factors that de-motivate Vietnamese practitioners to support SPI initiatives.

3 Perceived Value

In this particular study, we define ‘perceived value’ to mean the extent to which a SPI de-motivator is used, because it is perceived by practitioners to bring benefit either to the project or to the organisation. This may be considered to be a subjective view as it is provided by the respondents of this study. However, our respondents are considered to be SPI experts within their organisations. As such, we can assume that their opinion is grounded in significant experience of real world SPI initiatives.

In order to describe the notion of perceived value within SPI de-motivators, it is important to decide the “criticality” of a perceived value. For this purpose, we have used the following criterion:

- If the majority of respondents ($\geq 50\%$) thought that a de-motivator had high value then we treat that de-motivator as critical.

A similar approach has been used in the literature [21; 25]. Rainer and Hall [21] identified important factors in software process improvement with the criterion that if the 50% or more participants perceive that a factor has a major role in software process improvement efforts then that factor should be treated as having a major impact on software process improvement.

The perceived values of SPI de-motivators can act as a guide for SPI practitioners when implementing SPI initiatives because it will be easier to avoid a limited numbers of de-motivators that can undermine SPI implementation.

4 Study Design

We used face-to-face questionnaire based survey sessions as our main approach to collect data from twenty-three software development practitioners of eight Vietnamese software development organisations, which had initiated SPI programs. Although we do not claim this is a statistically representative sample, Appendix A does show that the participants of this study were working for organisations of varying sizes. It is further worth mentioning that the data was collected from practitioners who were involved in tackling real SPI implementation issues on a daily basis in their respective organisations. Therefore we have high confidence in the accuracy of their responses about their personal de-motivators of SPI.

It is further important to acknowledge that the practitioners sampled within organisations are representative of practitioners in organisations as a whole. A truly representative sample is impossible to attain and the researcher should try to remove as much of the sample bias as possible [26]. In order to make the sample fairly representative of SPI practitioners in particular organisation, different groups of practitioners from each organisation were selected to participate in this research. The sample of practitioners involved in this research includes developers, quality analysts, SQA team leaders, SQA managers, project managers, and senior management. Thus the sample is not random but a convenience sample, because we sought a response from a person with a specific role within a software development organisation. The practitioners who participated in this study fall into two main categories:

- “Developers” consisting of programmer/ analyst/ SQA coordinator.
- “Managers” consisting of team leader/ project manager, and senior managers.

We used a closed ended questionnaire as an instrument to collect self-reported data. Our questionnaire was based on the SPI de-motivators reported by Baddoo et al. [14] and Khalil et al. [13]. The questionnaire was also designed to elicit the importance that each respondent places on each identified de-motivator (perceived value). In order to describe the importance of de-motivators, the respondents were supposed to mention each identified de-motivator’s relative value (i.e., High value, Medium value, Low value, Zero value, or Not sure).

In order to analyse the perceived value of each identified SPI de-motivator, the occurrence of a perceived value (high, medium, low, zero) in each questionnaire was counted. By comparing the occurrences of one SPI de-motivator’s perceived values obtained against the occurrences of other SPI de-motivators’ perceived values, the relative importance of each de-motivator has been identified. We have also used this approach to identifying highly and lowly valued requirements engineering practices reported in [27].

The responses to the questionnaire were gathered during September 2005. We managed to administer the survey instrument to more than two software practitioners from some organisations. Although all the participants were well-versed in English and the questionnaire was in English. Moreover, the research team had a Vietnamese speaking researcher, who could have provided necessary explanation if required. The researchers also explained to the participants the meanings of SPI de-motivators. It was also explained to them that they were supposed to identify their personal motivators for SPI efforts.

5 Findings

5.1 Demographics

Table 1 shows the profile of the participants. Twenty-three surveys were conducted in eight organisations. We also wanted to analyse the responses based on a respondent's organisation size. To achieve this objective, we decided to cluster the participating organisations into different groups based on their sizes in terms of number of software development staff. Using the organisation size definition provided by the Australian Bureau of Statistics [28], we divided these organisations into three categories: SMALL (0 to => 19 employees), MEDIUM (20 to => 199 employees), and LARGE (200+ employees). According to this categorisation of the organisations, six are small-medium sized and two are large sized organisations.

Table 1. Demographics

ID	Number of employees	Number of participants	Titles of participants
1	80	2	Project manager, Team leader
2	70	6	Developer, Test leader, Programmer, Divisional head, Developer, QA manager
3	150	2	Chief Technology Officer, QA manager
4	150	3	Design team leader, R&D team leader, QA team leader
5	700	2	Project Manager, Process quality manager
6	150	2	QA Manager, Operation manager
7	50	4	QA manager, Project engineer, Project leader, Project leader
8	200	2	QA coordinator, QA manager

5.2 SPI De-motivators Identified by All Practitioners

Table 2 presents the list of de-motivators cited by all practitioners. The most common 'high' value de-motivator (13 out of 23) is 'lack of resources'. Lack of resources was also identified as one of the major de-motivators by Baddoo and Hall [14]. 'Workload' and 'lack of management commitment' are also most frequently cited 'high' value SPI de-motivators. Our results have confirmed the previous findings of several accounts that describe the importance of 'higher management support', 'resources' and 'time pressure for SPI initiatives' [18; 29-32]. More than 40% of practitioners consider 'lack of SPI management skills' as a 'high' value de-motivator. We have also found that 'low process priority' is a common 'high' value de-motivators for SPI. Research shows that normally SPI is not considered real work and software practitioners are expected to do SPI implementation in addition to their day-to-day software development activities [33].

Table 2. SPI de-motivators identified by all practitioners

SPI de-motivators	Occurrence in surveys (n=23) Perceived value					
	High	Medium	Low	Zero	Not sure	No response
Budget constraints	7	8	5	0	3	0
Cumbersome processes	7	11	2	1	1	1
Commercial pressures	5	12	3	0	2	1
Customers	3	10	6	2	2	0
Fire fighting	2	6	7	2	5	1
Imposition	5	7	6	2	3	0
Inadequate communication	5	11	4	1	2	0
Inadequate metrics	5	9	2	2	3	2
Inertia	7	5	8	0	3	0
Inexperienced staff	4	12	5	2	0	0
Irrelevant objectives/deliverables	4	11	4	3	1	0
Isolated best practices	7	6	7	2	1	0
Lack of evidence of direct benefits	6	11	3	1	2	0
Lack of feedback	7	11	3	1	1	0
Lack of management direction/commitment	11	9	1	1	1	0
Lack of resources	13	8	2	0	0	0
Lack of SPI management skills	10	9	3	0	1	0
Lack of standards	7	13	3	0	0	0
Lack of overall support	8	11	4	0	0	0
Large-scale programmes	1	13	6	1	2	0
Low process priority	9	7	4	0	3	0
Negative/bad experience	5	11	6	1	0	0
Organisational changes	5	10	6	0	2	0
Personality clashes	6	8	5	1	2	1
Project manager's lack of technical knowledge	3	12	4	2	1	1
Reduced creativity	5	7	6	3	2	0
Staff turnover	3	8	6	2	4	0
Time pressure/constraints	6	9	4	0	4	0
Workload	11	4	4	2	2	0

'Lack of standards' and 'large-scale programmes' are considered as the most common 'medium' value SPI de-motivator (13 out of 23). Research has shown that despite the importance of SPI implementation process, little empirical research has been carried out on developing ways in which to effectively implement SPI programmes [6; 18]. This suggests that the current problems with SPI are lack of SPI implementation standards. Different organisations adopted different approaches, based on their own individual experiences, in order to implement SPI initiatives rather than following a standard SPI implementation approach. This has led organisations to a chaotic situation with no standard for SPI implementation practices [34]. Because of this lack of SPI implementation standards, organisation are spending long time on SPI initiatives to realise the real benefits of this approach [7; 35]. More than half of the practitioners have cited 'commercial pressures', 'inexperienced staff', and 'project

manager's lack of technical knowledge' as 'medium' value de-motivators. Other most frequently cited 'medium' value de-motivators are: 'cumbersome processes', 'inadequate communication', 'irrelevant objectives/deliverables', 'lack of evidence of direct benefits', 'lack of feedback', 'lack of overall support' and 'negative/bad experience'.

We are surprised that the 'inertia' is the most common 'low' value de-motivator. In the study conducted with more than 200 UK practitioners, 'inertia' was identified as one of the major de-motivators [14]. We argue that this is a critical de-motivator as one of the biggest obstacles to introducing SPI initiatives is the unwillingness of practitioners to participate in this initiative. Other most frequently cited 'low' value de-motivators are: 'fire fighting' and 'isolated best practices'.

5.3 SPI De-motivators Identified by Different Groups of Practitioners

Table 3 shows the list of SPI de-motivators cited by developers and managers. We suggest that understanding the similarities in SPI de-motivators across different groups of practitioners can help to develop effective SPI implementation strategies. This is because, where respondents from different groups of practitioners consider that a de-motivator has an impact on SPI implementation then that de-motivator needs to be taken very seriously. This is because we have a de-motivator that is replicated across all groups of practitioners.

'Lack of resources' is the most common 'high' value de-motivator cited by developers (5 out of 8). Management often agrees for SPI initiatives without sufficient knowledge of the investment required for the initiative. In some organisations the management assume that SPI initiative will occur with very little investment. In other organisations, the management does not consider SPI initiative as a real project and hesitate to allocate resources [33]. Different studies have described the importance of resources for SPI initiatives: Florence [9] discusses the lessons learned in attempting to but not getting software CMM Level 4 at The MITRE corporation, and states that the organisation achieved CMM Level 3 due to sufficient resources were provided, but failed to achieve Level 4 because of lack of resources; Kautz and Nielsen [10] describe why implementation of SPI was not successful in one company than another company: "the project managers were hesitant to use resources from their own projects on any improvement activity" [10:pp4]. Other most common 'high' perceived value de-motivators cited by developers are: 'lack of SPI management skills', 'lack of overall support' and 'workload'.

75% of the developers consider 'inexperienced staff' is the 'medium' value de-motivator of SPI initiatives. We argue that experienced staff should be involved in SPI initiative, since they have detailed knowledge and first hand experience of SPI implementation. With experienced staff, less rework of the documentation items is required, real issues can be resolved, and chances of destruction are reduced [10; 36]. Different accounts have discussed this de-motivator: Kautz and Nielsen [10] describe the reason that one company failed its implementation of SPI: "the staff and technical director had no prior experience with SPI and its potential benefits" [10:pp4]; Moitra [36] describes the problems and difficulties of managing change for SPI and identifies inexperienced staff as one of the barriers for SPI: "the quality and process improvement people are often quite theoretical – they themselves do not understand

quite well the existing software development processes and the context in which they are used” [36:pp202]. ‘Negative/ bad experience’ is considered to be ‘low’ value de-motivator (4 out of 8).

Table 3. SPI de-motivators identified by different group of practitioners

SPI de-motivators	Developers (n=8)					Managers (n=15)				
	H	M	L	Z	NS/ NR	H	M	L	Z	NS/ NR
Budget constraints	1	4	2	0	1	6	4	3	0	2
Cumbersome processes	1	4	1	0	2	6	7	1	1	0
Commercial pressures	1	3	1	0	3	4	9	2	0	0
Customers	1	5	1	1	0	2	5	5	1	2
Fire fighting	0	3	1	1	3	2	3	6	1	3
Imposition	2	3	1	1	1	3	4	5	1	2
Inadequate communication	3	3	1	0	1	2	8	3	1	1
Inadequate metrics	2	3	0	1	2	3	6	2	1	3
Inertia	3	1	3	0	1	4	4	5	0	2
Inexperienced staff	1	6	1	0	0	3	6	4	2	0
Irrelevant objectives/deliverables	2	4	1	1	0	2	7	3	2	1
Isolated best practices	1	3	2	1	1	6	3	5	1	0
Lack of evidence of direct benefits	2	5	0	0	1	4	6	3	1	1
Lack of feedback	3	4	0	1	0	4	7	3	0	1
Lack of management direction/commitment	2	4	1	0	1	9	5	0	1	0
Lack of resources	5	3	0	0	0	8	5	2	0	0
Lack of SPI management skills	4	3	0	0	1	6	6	3	0	0
Lack of standards	3	4	1	0	0	4	9	2	0	0
Lack of overall support	4	3	1	0	0	4	8	3	0	0
Large-scale programmes	0	4	3	0	1	1	9	3	1	1
Low process priority	3	3	1	0	1	6	4	3	0	2
Negative/bad experience	2	2	4	0	0	3	9	2	1	0
Organisational changes	2	4	2	0	1	3	6	4	0	1
Personality clashes	1	5	2	0	2	5	3	3	1	1
Project manager’s lack of technical knowledge	1	5	1	0	1	2	7	3	2	1
Reduced creativity	2	3	3	0	1	3	4	3	3	1
Staff turnover	3	2	1	0	2	0	6	5	2	2
Time pressure/constraints	1	4	1	0	2	5	5	3	0	2
Workload	4	1	1	2	0	7	3	3	0	2

H=High, M=Medium, L=Low, Z= Zero, NS/ NR=Not sure/ No response

Table 3 shows that 60% of managers have cited ‘lack of management commitment’ as a ‘high’ value de-motivator. Nearly half of the managers have considered ‘lack of resources’ and ‘workload’ as ‘high’ value de-motivators. We believe that managers struggle with allocation of time for different activities relating to SPI and day to day software development during SPI initiatives [37], since software practitioners are expected to do SPI activities in addition to their day-to-day software development

activities [33]. More than half of the managers have cited ‘commercial pressures’, ‘inadequate communication’, ‘lack of standards’, ‘lack of overall support’, ‘large-scale programmes’, and ‘negative/bad experience’ as ‘medium’ perceived de-motivators.

Table 4. SPI de-motivators identified by SM and Large organisations

SPI de-motivators	Small and Medium (n=19)					Large (n=4)				
	H	M	L	Z	NS/ NR	H	M	L	Z	NS/NR
Budget constraints	7	5	4	0	3	0	3	1	0	0
Cumbersome processes	7	7	2	1	2	0	4	0	0	0
Commercial pressures	5	9	2	0	3	0	3	1	0	0
Customers	2	9	5	2	1	1	1	1	0	1
Fire fighting	2	5	6	1	5	0	1	1	1	1
Imposition	5	5	5	2	2	0	2	1	1	0
Inadequate communication	3	9	4	1	2	2	2	0	0	0
Inadequate metrics	4	6	2	2	5	1	3	0	0	0
Inertia	6	5	7	0	1	1	0	1	0	2
Inexperienced staff	4	10	3	2	0	0	2	2	0	0
Irrelevant objectives/deliverables	4	8	3	3	1	0	3	1	0	0
Isolated best practices	5	6	5	2	1	2	0	2	0	0
Lack of evidence of direct benefits	6	8	2	1	2	0	3	1	0	0
Lack of feedback	7	7	3	1	1	0	4	0	0	0
Lack of management direction/commitment	9	7	1	1	1	2	2	0	0	0
Lack of resources	10	8	1	0	0	3	0	1	0	0
Lack of SPI management skills	8	8	2	0	1	2	1	1	0	0
Lack of standards	6	11	2	0	0	1	2	1	0	0
Lack of overall support	7	9	3	0	0	1	2	1	0	0
Large-scale programmes	1	10	6	1	1	0	3	0	0	1
Low process priority	6	7	4	0	2	3	0	0	0	1
Negative/bad experience	4	9	5	1	0	1	2	1	0	0
Organisational changes	5	7	6	0	1	0	3	0	0	1
Personality clashes	5	6	5	1	2	1	2	1	0	0
Project manager’s lack of technical knowledge	2	10	3	2	2	1	2	1	0	0
Reduced creativity	5	5	5	3	1	0	2	1	0	1
Staff turnover	3	6	5	2	3	0	2	1	0	1
Time pressure/constraints	4	8	3	0	4	2	1	1	0	0
Workload	9	3	3	2	2	2	1	1	0	0

5.4 SPI De-motivators Identified by Practitioners of Large and Small-Medium Sized Organisations

We hypothesized that small and medium (SM) sized organisations would have a different pattern of response than large organisations. For example we expect that SM

sized organisations are more resource-constrained and require benefits to be returned within shorter periods.

Table 4 shows the list of SPI de-motivators cited by small and medium (SM) sized and large sized organisations. Most of the SM sized organisations have cited 'lack of resources' as the 'high' perceived value de-motivator. Nearly 50% of SM sized organisations have cited 'lack of management commitment' and 'workload' as the 'high' perceived de-motivators. More than 50% of SM sized organisations have cited 'inexperienced staff', 'lack of standards', 'large-scale programmes', and 'project manager's lack of technical knowledge' as the 'medium' perceived value de-motivators.

Table 4 shows that most practitioners of large organisations have cited 'lack of resources' and 'low process priority' as the 'high' perceived value de-motivators. 'Cumbersome processes' and 'lack of feedback' are the most common 'medium' perceived value de-motivator cited by large organisations (4 out of 4).

6 Validity

Construct validity is concerned with whether or not the measurement scales represent the attributes being measured. The attributes are taken from a substantial body of previous research [14; 27; 32]. The responses from the practitioners show that all the attributes considered were relevant to their workspace. External validity is concerned with the generalisation of the results to other environments than the one in which the initial study was conducted [38]. External validity was examined by conducting survey with 23 practitioners from eight organisations.

There are some limitations that we considered worth mentioning. A disadvantage of the questionnaire survey method is that respondents are provided with a list of possible de-motivators and asked to select from that list. This tends to pre-empt the de-motivators investigated and to limit them to those reported in existing studies - respondents only focus on the de-motivators provided in the list. It is also possible that respondents may misinterpret the de-motivator provided in the questionnaire. However, we tried to address this issue by explaining the meaning of each de-motivator included in the questionnaire (Please see Appendix A for meanings of the de-motivators). Another issue is that the questionnaire surveys are usually based on self-reported data that reflects what people say happened, not necessarily what they actually did or experienced. Our results are limited to the respondents' knowledge, attitudes, and beliefs regarding the factors that de-motivate them to support SPI initiatives in their organisation. This situation can cause problems when practitioners' perceptions may be inaccurate or factors identified as important SPI de-motivators may not be important at all. However, like the researchers of many studies based on opinion data (e.g. [32; 39; 40]), we also have full confidence in our findings because we have collected data from practitioners working in quite diverse roles and directly involved in SPI activities within their organisations. Sample size

may be another issue as we collected data from only 23 practitioners from 8 Vietnamese organisations. To gain a broader representation of Vietnamese practitioners' views on this topic, more practitioners and organisations need to be included in a study. Our participants belonged to only one country, Vietnam, which is another limitation as the findings cannot be widely generalized to practitioners from other countries.

7 Summary and Conclusion

We report on our empirical study of SPI de-motivators. We analysed the experiences, opinions and views of practitioners in order to identify factors that de-motivate them from SPI initiatives. Five types of SPI de-motivators assessments (high, medium, low, zero, or do not know) were identified that have led us to the notion of a 'perceived value' associated with each SPI de-motivator (from high value to no value). To have a perceived high value, we refer to a de-motivator that is common across all practitioners.

In order to describe the notion of perceived value within SPI de-motivators, we have used the following criterion:

- If a perceived value of a SPI de-motivator is cited in the questionnaire surveys with a frequency of $\geq 50\%$, then we treat it as critical.

In order to answer RQ1, using above criterion, we have identified 'lack of resources' as a 'high' perceived value SPI de-motivator. We have identified 5 SPI de-motivators that have 'medium' perceived value: 'commercial pressure', 'inexperienced staff', 'lack of standards', 'large-scale programmes', and 'project manager's lack of technical knowledge'.

In order to answer RQ2, our results show the opinion of each individual practitioner group. Developers are de-motivated by: 'lack of resources', 'lack of SPI management skills', 'lack of overall support' and 'workload'. Managers are de-motivated by: 'lack of management direction/commitment' and 'lack of resources'.

In order to answer RQ3, our results show that SM sized organisations are de-motivated by 'lack of resources'. The large organisations are de-motivated by: 'inadequate communication', 'isolated best practices', 'lack of management direction/commitment', 'lack of resources', 'lack of SPI management skills', 'low process priority', 'time pressure/constraints', and 'workload'.

Our results show a 'lack of resources' high perceived value de-motivator common to all practitioner groups. This de-motivator is also common between SM sized and large organisations. We have also found 2 'medium' perceived value de-motivators common to all practitioner groups: 'lack of standards' and 'large-scale programmes'. We suggest that focusing on these de-motivators offers SPI practitioners short-term opportunities for successfully implementing SPI initiatives. This is because different practitioners who were tackling real issues on a daily basis frequently cited these de-motivators.

References

1. Standish-Group: Chaos - the state of the software industry. Standish group international technical report, pp. 1–11 (1995)
2. Standish-Group: Chaos - the state of the software industry (2003)
3. SEI: Capability Maturity Model® Integration (CMMISM), Version 1.1. SEI, CMU/SEI-2002-TR-029, Software Engineering Institute, USA (2002)
4. ISO/IEC-15504: Information technology - Software process assessment. Technical report - Type 2 (1998)
5. Staples, M., Niazi, M., Jeffery, R., Abrahams, A., Byatt, P., Murphy, R.: An Exploratory Study of Why Organizations Do Not Adopt CMMI, in press for publication, *Journal of Systems and Software* (2007)
6. Leung, H.: Slow change of information system development practice. *Software quality journal* 8(3), 197–210 (1999)
7. SEI: Process Maturity Profile. Software Engineering Institute Carnegie Mellon University (2004)
8. Davis, A., Hickey, A.: Requirements Researchers: Do We Practice What We Preach? *Requirements Engineering Journal* 2002 7, 107–111 (2002)
9. Florence, A.: Lessons learned in attempting to achieve software CMM Level 4, *CrossTalk* pp. 29–30 (August 2001)
10. Kautz, K., Nielsen, P.A.: Implementing software process improvement: Two cases of technology transfer. In: *Proceedings of the 33rd Hawaii Conference on System Sciences*, vol 7, pp. 1–10 Maui, USA (2000)
11. SEI: Process maturity profile of the software community. Software Engineering Institute (2002)
12. Ngwenyama, O., Nielsen, P.: A Competing values in software process improvement: An assumption analysis of CMM from an organizational culture perspective. *IEEE Transactions on Software Engineering* 50, 100–112 (2003)
13. Khalil, O.E.M., Zawacki, R.A., Zawacki, P.A., Selim, A.: What Motivates Egyptian IS Managers and Personnel: Some Preliminary Results. *SIGCPR* 97, 187–196 (1997)
14. Baddoo, N., Hall, T.: De-Motivators of software process improvement: An analysis of practitioner's views. *Journal of Systems and Software* 66(1), 23–33 (2003)
15. McDermid, J., Bennet, K.: Software Engineering research: A critical appraisal. *IEE Proceedings on software engineering* 146(4), 179–186 (1999)
16. Hall, T., Wilson, D.: Views of software quality: a field report. *IEEE Proceedings on Software Engineering* 144(2), 111–118 (1997)
17. Hall, T., Wilson, D.: Perceptions of software quality: a pilot study. *Software quality journal* 7, 67–75 (1998)
18. Goldenson, D.R., Herbsleb, J.D.: After the appraisal: A systematic survey of Process Improvement, Its benefits, And Factors That Influence Success. SEI, CMU/SEI-95-TR-009, Software Engineering Institute, USA (1995)
19. Stelzer, D., Werner, M.: Success factors of organizational change in software process improvement. *Software process improvement and practice* 4(4) (1999)
20. El-Emam, K., Fusaro, P., Smith, B.: Success factors and barriers for software process improvement. Better software practice for business benefit: Principles and experience. IEEE Computer Society, Los Alamitos (1999)
21. Rainer, A., Hall, T.: Key success factors for implementing software process improvement: a maturity-based analysis. *Journal of Systems & Software* 62(2), 71–84 (2002)

22. Rainer, A., Hall, T.: A quantitative and qualitative analysis of factors affecting software processes, *Journal of Systems & Software*, Accepted awaiting publication (2002)
23. Humphrey, W.S.: Three Process Perspectives: Organizations, Teams, and People. *Annals of Software Engineering* 14, 39–72 (2002)
24. Chidamber, S.R.: An Analysis of Vietnam's ICT and Software Services Sector, *The Electronic Journal on Information Systems in Developing Countries*, pp. 1–11 (2003) (Last accessed November 01, 2005), <http://www.ejisdc.org>
25. Niazi, M., Wilson, D., Zowghi, D.: A Maturity Model for the Implementation of Software Process Improvement: An empirical study. *Journal of Systems and Software* 74(2), 155–172 (2005)
26. Coolican, H.: *Research Methods and Statistics in Psychology*. Hodder and Stoughton, London (1999)
27. Niazi, M., Cox, K., Verner, J.: An empirical study identifying high perceived value requirements engineering practices. In: *Fourteenth International Conference on Information Systems Development (ISD'2005)* Karlstad University, Sweden (August 15-17, 2005)
28. Trewin, D.: *Small Business in Australia: 2001*. Australian Bureau of Statistics report 1321.0 (2002)
29. Basili, V.R., McGarry, F.E., Pajerski, R., Zelkowitz, M.V.: Lessons learned from 25 years of process improvement: The rise and fall of the NASA software engineering laboratory. In: *International Conference on Software Engineering*, Orlando, Florida, USA, pp. 69–79 (2002)
30. Butler, K.: *Process lessons learned while reaching Level 4*, CrossTalk (May 1997)
31. Pitterman, B.: *Telcordia Technologies: The journey to high maturity*, IEEE Software, pp. 89–96 (July/August 2000)
32. Niazi, M., Wilson, D., Zowghi, D.: Critical Success Factors for Software Process Improvement: An Empirical Study. *Software Process Improvement and Practice Journal* 11(2), 193–211 (2006)
33. Niazi, M., Wilson, D., Zowghi, D.: Critical Barriers for SPI Implementation: An empirical study. In: *IASTED International Conference on Software Engineering (SE 2004)*. Austria pp. 389–395 (2004)
34. Zahran, S.: *Software process improvement - practical guidelines for business success*. Addison-Wesley, London (1998)
35. Niazi, M.: *Software Process Improvement: A Road to success*. In: *The 7th International Conference on Product Focused Software Process Improvement*, LNCS, pp. 395–401 (2006)
36. Moitra, D.: Managing change for (SPI) initiatives: A practical experience-based approach. *Software Process Improvement and Practice* 4(4), 199–207 (1998)
37. Baddoo, N., Hall, T., Wilson, D.: Implementing a people focused SPI programme. In: *11th European Software Control And Metrics Conference and The Third SCOPE Conference on Software Product Quality*, Munich (2000)
38. Regnell, B., Runeson, P., Thelin, T.: Are the Perspectives Really Different-Further Experimentation on Scenario-Based Reading of Requirements. *Empirical Software Engineering* 5(4), 331–356 (2000)
39. Baddoo, N., Hall, T.: Motivators of software process improvement: An analysis of practitioner's views. *Journal of Systems and Software* 62, 85–96 (2002)
40. Beecham, S., Tracy, H., Austen, R.: *Software Process Problems in Twelve Software Companies: An Empirical Analysis*. *Empirical software engineering* 8, 7–42 (2003)

Appendix A: Definition of SPI De-motivators [14]

SPI de-motivators	Definition
Budget constraints Cumbersome processes Commercial pressures	Limited budget for SPI activities Difficult and bureaucratic processes Pressure to satisfy commercial objectives of organisation
Customers Fire fighting Imposition	Interference from customers Short term policies for tackling problems Imposing SPI without consultation with practitioners
Inadequate communication Inadequate metrics Inertia Inexperienced staff Irrelevant objectives/deliverables	Lack of communication between practitioners Inadequate metrics for SPI Resistance to SPI Staff with limited SPI knowledge SPI objectives are not tailored to real organisational needs
Isolated best practices Lack of evidence of direct benefits Lack of feedback	Best practices are not shared within the organisation Practitioners are not provided with the evidence of the success of SPI Practitioners are not given feedback of the SPI outcomes
Lack of management direction/commitment Lack of resources Lack of SPI management skills	No commitment for SPI from higher management The organisation does not have the resources for SPI Insufficient personnel with the appropriate skills for SPI
Lack of standards Lack of overall support Large-scale programmes Low process priority	There are no overall standards to software development SPI is not overwhelming supported by the practitioners The SPI initiative is too big for the organisation SPI is given low priority as compared to other project activities
Negative/bad experience Organisational changes	Previous negative experiences of SPI Organisational changes impact negatively on ongoing SPI programmes
Personality clashes Project manager's lack of technical knowledge Reduced creativity Staff turnover Time pressure/constraints Workload	Personal politics frustrates the SPI effort Project managers do not possess technical knowledge of software production SPI takes away individual creativity High staff turnover undermine SPI initiatives Pressure to deliver product on time Practitioners have too much work and have insufficient time for SPI

Defining Software Processes Through Process Workshops: A Multicase Study

Finn Olav Bjørnson¹, Tor Stålhane¹, Nils Brede Moe², and Torgeir Dingsøy²

¹ Department of Computer and Information Science,
Norwegian University of Science and Technology
NO-7491 Trondheim, Norway

{bjornson, stalhane}@idi.ntnu.no

² SINTEF Information and Communication Technology
NO-7465 Trondheim, Norway

{Nils.B.Moe, Torgeir.Dingsoyr}@sintef.no

Abstract. We present the application of the process workshop method to define revised work processes in software development companies. Through two empirical action research studies, we study the impact of company premises and goals on the execution and subsequently on the results of the method. We conclude that both premises and goals will influence the workshops, and suggest how the focus of the workshops should be altered to achieve better results depending on the context. We also strengthen previous claims that the process workshops are a good arena that fosters discussion and organizational learning, and that involvement in the workshops leads to higher acceptance and usage of the resulting process.

Keywords: Software Process Improvement, Project Workshop, Empirical Study, Action Research.

1 Introduction

The way we develop and maintain software, or the software process, has long been regarded as crucial for software quality and productivity [16]. In many companies, software development is performed in a rather informal fashion, and problems of late and unsatisfactory deliveries are not uncommon.

Problems related to the use of informal development include problems with transferring competence from one project to another, difficulties in establishing best practices, and the widely varying nature of problems to be solved. In order to address these challenges and to improve the quality of the software development process, a lot of companies develop process guides to structure their work.

The process workshop (PWS) method was designed as a lightweight method to help facilitate the development of such process guides. Apart from the original introduction of the process workshop [11] and a Finnish application of the same method [19], there is little empirical evidence on the practical application of this method. This paper aims to add to the body of knowledge on process workshops as a

tool for software process improvement, and describes how company context and goals affects the execution of the method and its results.

In the following we describe our work in two companies, hereafter referred to as Alpha and Beta Company. One is a small and one is a medium sized software company, and they both used process workshops to define their software process. Our focus is on the process workshop itself and how processes were constructed. The description of this process, i.e., how it will later appear in an electronic process guide, and the cost-benefit of the process workshop method is as such outside the scope of this paper. Our research goal which we want to answer in this paper is:

How do available information, company context and goals affect the execution and results of process workshops?

The paper is structured as follows: In chapter 2 we take a closer look at related work, and the method we adapted for our cases. Chapter 3 describes the research method employed in each case. Chapter 4 gives a deeper introduction to each case. Chapter 5 discusses the differences between the cases and our findings. Chapter 6 concludes our findings and describes possible routes for further research.

2 Related Work

When companies choose to design their own development processes, one option is to assign the task to a group of expert “process engineers” as described by Becker-Kornstaedt [7, 8]. One or more process engineers elicit process data from interviews, documents, surveys, e-mails and observation, and then interpret this data to produce a process model. This approach relies heavily on the experience and skill of the process engineer. Therefore, without any structured method, quality and repeatability cannot be ensured. It is, however, unlikely that the use of qualitative methods alone can compensate for experience in process modeling and software engineering [8]. When using a process engineer to formulate a process model, it is common to create a descriptive model. A descriptive model is a model, which expresses processes currently in use. Descriptive software process modeling is an important part of any software process improvement (SPI) program, because descriptive modeling allows process engineers to understand existing processes, communicate process and analyze existing practices for improvement [8]. For this reason, much work has been done on proposing languages, techniques and tools for descriptive process modeling.

An alternative to using process engineers is to involve the employees more in designing the process models, for example through workshops [1, 17]. This type of work takes up the heritage from employee participation in organizational development, a part of Scandinavian work tradition as well as in most work on improvement, from the Total Quality Management principles [10] to the knowledge management tradition in Communities of Practice [25]. Participation is also one of the most important foundations of organization development and change [17], and one of the critical factors for success in software process improvement [13].

Some studies have found that employee involvement lead to organizational effectiveness, measured through financial performance, turnover rate and workforce

morale [21, 24]. Another potential effect of participation is increased emotional attachment to the organization, resulting in greater commitment, motivation to perform and desire for responsibility. Riordan et al. [21] use a framework with four attributes to define employee involvement:

- Participative decision
- Information sharing
- Training
- Performance-based rewards

There are several techniques available for achieving participation. Examples are search conferences [20], survey feedback [6], autonomous work groups [14], quality circles [14, 15]. All of which are predicated on the belief that increased participation will lead to better solutions and enhanced organizational problem-solving capability.

In software development, the software developers and the first-line managers are the ones who are into the realities of the day-to-day details of particular technologies, products, and markets. Hence, it is important to involve all who are part of the software process, and have decisions regarding the development of process guides made by those who are closest to the problem.

Consequently, and in order to get realistic descriptions with accurate detail as well as company commitment in an efficient manner, all relevant employee groups should be involved in defining the processes. This can be done by arranging several process workshops [17] in the form of quality circles [15] as a tool to reach a consensus on work practice. A quality circle is composed of volunteers who arrange regular meetings to look at productivity and quality problems, and discuss work procedures [15]. The strength of the circle is that they allow employees to deal with improvement issues that are not dealt with in the regular organization. The quality circles used in the process workshop have all been temporary, and created with a relative well-bounded mandate to be fulfilled. Once a sub-process has been accomplished, the circle is disbanded. This kind of quality circles is also known as “Task forces” [14].

2.1 The Process Workshop Method

In the studies reported in this paper, we used a method called process workshop [11]. The method is designed to involve the users of the future process in discussing and defining the processes. It ensures that people discuss how they work – which fosters learning even before the process guide is available in the company. It also assures quality – the process guide is developed by people who know how to do the work; it does not describe how external consultants or senior staff imagine what “ideal” development processes should look like.

The process workshop approach to defining process(es) consists of six main steps and five sub-steps as shown in Figure 1 below. Since the focus of our work is on the process workshop itself, we only provide details of the relevant substeps here. More details on the process workshops method can be found in [11].

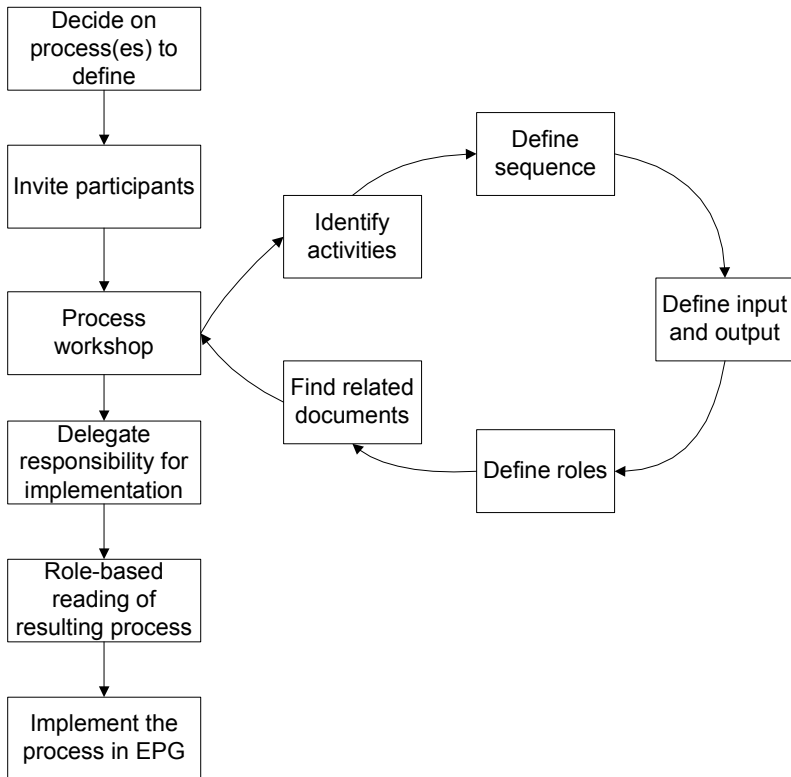


Fig. 1. Steps to define process in a workshop

The theoretical approach of the five sub steps are:

- *Identify activities.* Brainstorming on the main activities of the process by using the KJ process [22] and documenting the result. The KJ is a creative group technique to organize and find relations between seemingly unrelated ideas.
- *Define the sequence of the activities.* A suitable workflow between the activities from the previous phase is found.
- *Define inputs and outputs.* Identify documents or artifacts that must be available to start a given sub-process, and which documents that mark the end of such sub-processes. Conditions that must be satisfied to begin or exit the sub-process can be described in checklists.
- *Define roles.* Defining which roles should contribute in each activity.
- *Related documents.* Identify documents that either already exist in the company, or new documents that would be helpful in carrying out the activities. Such documents can be templates, checklists and good examples of input or output documents.

A process workshop can be used both to make a descriptive process model and to directly formulate a new and improved process. In the latter case process models are improved directly in the workshops through the discussions, without an analysis of the present situation.

3 Research Method

This study reports on two separate empirical studies. Each study investigated the application of process workshops to define software processes for software development companies. However, the research method differed slightly between the two cases, and the companies are also at different stages in their improvement efforts. The research method and the difference in application to the two companies are described in this chapter. Two of the authors of this paper were responsible for the research at the Alpha Company, while the two others handled the research at the Beta Company.

Both Alpha and Beta were involved in the same national research project, aimed at investigating software process improvement in software engineering. Due to the cooperative nature of this research project, the research method adopted for both companies was the participative research method, action research [4]. In order to properly describe and differentiate the research methods used, we describe them according to the five principles suggested by Davison et al. [9] (table 1) and the three aspects of control structures suggested by Avison et al. [3] (table 2).

Table 1. The five principles of canonical action research, by Davison et al.

Principles of canonical action research	
1.	The principle of the researcher-client agreement.
2.	The principle of cyclical process model.
3.	The principle of theory.
4.	The principle of change through action.
5.	The principle of learning through reflection.

Table 2. Forms and Characteristics of the major AR control structures, by Avison et al.

Control aspect	Forms	Characteristics
Initiation	Researcher	Field experiment
	Practitioner	Classic action research genesis
	Collaborative	Evolves from existing interaction
Authority	Practitioner	Consultative action warrant
	Staged	Migration of power
Formalisation	Identity	Practitioner and researcher are the same person
	Formal	Specific written contract or letter of agreement
	Informal	Broad, perhaps verbal, agreements
	Evolved	Informal og formal projects shift into the opposite form

At the Alpha company, the research on how to use project workshops to define software process was carried out during 2003. The process was later implemented in an electronic process guide, and the use of the guide over time was studied [17]. The research on project workshops to define their software process at Beta Company was carried out during 2005, in other words after the study at Alpha company. Since the goal of the company was close to that of Alpha, we decided to adopt the method of process workshops to define the process. The company wanted to define their process, and the researchers got a chance to empirically evaluate the method previously suggested and used at Alpha.

Concerning the first principle of researcher-client agreement, this research was done in a general project on software process improvement, where both companies wrote an improvement plan and the researchers wrote a research plan for each company.

The research followed the action research model (principle two) proposed by Susman and Evered [23] in discussing the situation at the companies, planning action, taking action, evaluating action, and finally specifying for learning. The research has gone through three “evolutionary” cycles at Alpha, however our focus for this paper is on the first cycle in which the process workshops were held to establish the process. At Beta we have only done one evolutionary cycle at the present time.

The third principle of theory, was satisfied for both companies through the research questions and our focus on developing and testing the method based on the theory of user involvement [14, 15, 21, 24].

The fourth principle of change through action was satisfied through the actions of holding the project workshops. The results form the basis for a new electronic process guide, which includes examples based on the defined process. These results have been used to implement the new defined process at Alpha, whereas Beta has not come this far yet.

The fifth principle of learning through reflection was achieved at Alpha through project meetings in which the researchers and company representatives discussed actions that were taken and analyses made by the researchers. At Beta the results were discussed in a series of meetings, we held a post mortem analysis (PMA) [23] of the project workshops to evaluate it at the end, and conducted an interview with the person responsible for the process improvement initiative at the company.

From the aspect of control structures on action research, we can put the following characteristics on the research projects. The initiation was collaborative for both projects. Both the company and the researchers were in a common research project aimed at improving software processes, and the research plan was developed from the joint wishes of practitioners and researchers.

The authority of the projects is where we observe the main difference. At Alpha it is characterised as staged. In the beginning, the researchers were heavily involved with developing the workshops, while the company assumed more of the responsibility and workload towards the end. At Beta we also characterise the authority as staged, but the opposite effect was seen. In the beginning, the company was very much involved with developing a solution, but as an external project

demanded more and more of their resources, power was transferred to the researchers who had to carry much of the workload.

The formalization of both projects can be said to have evolved from formal in the beginning, with a clear structure and plan, to more informal at the end.

4 Empirical Results from the Two Software Companies

In this chapter we describe the two companies in which we conducted our research in greater detail. We describe the context, the practicalities surrounding the process workshops, how the companies used the data from the workshops, and finally an evaluation of the workshops themselves.

4.1 Alpha Company

Alpha Company was founded in 1984, and is one of the leading producers of receiving stations for data from meteorological and Earth observation satellites. The company has worked with large development projects, both as a prime contractor and as a subcontractor. The company has approximately 60 employees, many with master's degrees in computing science, mathematics or physics.

The size of typical product development projects are 1000-4000 work hours. Customers range from universities to companies like Lockheed Martin and Alcatel to governmental institutions like the European Space Agency and the Norwegian Meteorological Institute. Most of the software systems that are developed run on Unix, many on the Linux operating system. Projects are managed in accordance with quality routines from the European Corporation for Space Standardisation and ISO 9001-2000 [5].

The company had an extensive quality system which was cumbersome to use because of the size and existence partly on file and partly on paper. Since it also did not emphasize such aspects as incremental and component development, the QA system came under increasing pressure to change. It became impossible to follow the standards and even more impossible to do effective quality assurance work in the projects. As part of being certified according to ISO 9001-2000, the company decided to develop a process-oriented quality system [18].

Defining New Processes

Management of the project for defining the new processes was kept with the Quality Assurance (QA) department. One of the two persons working in the QA department had earlier worked as a developer and was now member of the top management. This way this project was anchored both among the developers and managers.

In an initial workshop with both developers and managers it was defined that the process descriptions had to:

- Reflect the “best practices” currently used within the company (take the best from the earlier system into the new system).
- Comply with modern methodologies like the Unified Process and Component Based Development.

- Integrate the process descriptions with important tools for development (e.g. requirements definitions and use-case description).
- Be easy to tailor when a new project is started.
- Be released when the first processes are defined, so it becomes possible to give instant feedback and then keep up the involvement

From these requirements it was decided that the new processes should be created based on “best practice” in the company, with important input from the existing system and engineering tools. It was never an option to first analyse the existing processes and then improve them. This was because they wanted to get the new processes defined quickly to meet the new ISO standard, and to use as little time as possible to keep up the enthusiasm among the developers. The process workshop also provided the possibility to discuss and improve today’s working processes without a thorough analysis. It was also decided that the process descriptions were going to be developed in “process workshops” to achieve participation.

After the requirements were defined, seven process workshops were arranged. Alpha identified four main project types, and they chose “Product Development” - the most common one - as a starting point for the subsequent process workshops. “Product Development” was divided into four sub processes: “Specification”, “Elaboration”, “Component Construction” and “System Integration”.

More than 20 people (1/3 of the staff) participated in one or more workshops. The people who participated in the process workshops were selected by the quality department to represent a variety of roles, experience and opinions. The workshops usually lasted half a day.

Each workshop started by defining the sub-processes in the main process. Then we defined each sub-process activities and their sequence. We used the KJ process [22] for brainstorming and documenting the result. The KJ is a creative group technique to organize and find relations between often seemingly unrelated ideas. After the activities were identified and organized in workflows, the documents for input and output to the process were defined. These documents could be already existing templates, checklist and good examples. Next we identified related roles to each process. After all the sub-processes were defined, the responsibilities for implementing the processes into the electronic process guide.

Implementing the Processes

The implementation was executed by QA personnel in a self-made tool and released on the intranet. The first prototype was ready after only a few weeks, and even though the process guide was incomplete it was possible to start real-life testing with a few projects. The projects were encouraged to respond immediately to the process descriptions if they are unclear, uncompleted or unusable. In this way the users were still involved in developing the process descriptions.

The company used 180 work hours in workshops and 1049 work hours in total for development of the first version of the process guide.

4.2 Beta Company

The Beta Company has 20 employees. Their main activities are hiring out consultants as developers, developing complete solutions for customers, and hiring out consultants and project managers as advisors for selecting technology, strategy or process. Typically, no more than four to five consultants are at any time working for the same customer.

The managers of the company wish to leverage the company in the market by providing solutions to the problems of their customers. The solutions should make them stand out and increase the probability that the customers later return with new projects. In order to do this, they wish to foster an environment where all ideas and knowledge are shared freely among the employees, and where the employees can draw upon the experience of each other to provide good services to their customers. This work is difficult since a lot of the employees at any given time are out at the customers' site where they don't have direct access to their colleagues.

One of the identified stumbling blocks for experience sharing and reuse was the lack of a common process and a common set of document templates. In order to remove, or at least reduce this problem, the company wanted to define, document and implement a framework that could be used for development, consultancy and operation. The framework should be easily accessible for all employees and should help them to do their jobs better than today and to show Beta as a highly competent consultancy company. The company started to drift away from this goal after approximately six months and decided instead to document how they worked now. A shift from prescriptive to descriptive modeling. Although never explicitly stated, the focus was on identifying the documents – artifacts – that were produced, who produced them and how. In addition it was important for the company to create an awareness of and understanding for the use of a process that encompassed all development activities. At present, the developers thought in terms of jobs – things to do – not in terms of processes and artifacts. One of the goals was to make them think and work in terms of processes and process steps.

Defining New Processes

When the researchers became involved, we saw it as a good opportunity to further test the process workshop method to document their process. We used a sequence of process workshops – one for each of the identified main processes that the company used. The input to the workshops was mainly the developers' experiences with the way they had worked in previous projects. Since the company had no single, defined process and each project more or less invented its own, this was a quite diverse source of information and experience. Each participant brought with him experiences from several processes.

Since part of the goal of the Beta Company was to see which artifacts were needed, we tried to use the standard process worksheet, which has a separate area for documents. However, the workshop participants ignored this area and preferred to

mix activities and documents in the same diagram. One of the reasons for this may be that different workshop participants had different ideas about what was done in a project. It was much easier to agree on the documents that are developed than to agree on how they are produced.

We held a total of six workshops over a period of 12 months. Five of the developers participated in two or more of the workshops while an extra five participated in at least one. The workshops treated the processes: requirements, estimation, analysis, implementation, testing and project control and follow-up activities. In addition, we arranged a Post Mortem Analysis (PMA) [12] workshop to assess the whole process workshop series.

We will not treat the results from each workshop in any detail but will instead focus on the workshop process and its results. In addition, we will discuss some of the results from the workshop PMA.

We used the KJ process to create the diagrams during the workshops. Based on the resulting diagrams it was straight forward to see which documents were generated. It is important to note that while the workshop participants were fairly clear on which documents to produce, they are rather vague on the process steps.

Implementing the Processes

Even though documents such as use-case descriptions were generated in this process, all of the documents created in the requirements process will resurface in later processes and will be refined there. In the developers' view it was therefore unreasonable to claim that a certain document "belonged to" a certain process or process step. For this reason, the company decided on the following approach to get a unified process concept:

- Identify all documents and code them with information on the process they are generated in and where they later are refined or used.
- Identify all document dependencies, i.e. which documents use which other documents.
- Store templates and examples for all documents that are used in one or more processes.
- Define a discussion tread for each document. This will enable all developers to give input on their experience, what works, what does not and how can we improve on the templates.

Evaluating the Workshop Approach

When all the company's processes had been analyzed in a process workshop we arranged a PMA to identify strong and weak points in the workshop process used. Most of the negative points related to the lack of participation from the company's management and does not contribute to our understanding of the use of process workshops. The KJ diagram for the positive points is shown below in figure 3.

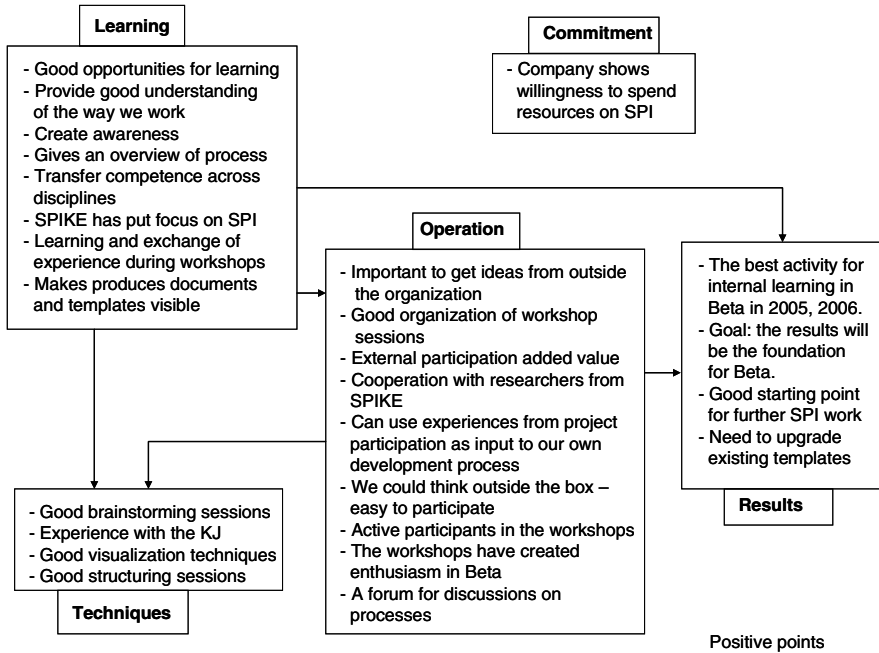


Fig. 3. Positive KJ diagram from the PMA

Our main experiences can be summed up as follows:

- In a company with many and varied versions of the same process it is easier to focus on documents than on process steps. Dependencies between documents will enforce a sequence of activities but the focus will be on *what*, not on *how*.
- Among the developers, the process workshops are conceived as a positive contribution in several ways, e.g.:
 - Gives an opportunity for active participation - not just asked what you do but be able to use your own experience to contribute to the company’s processes.
 - Get a better understanding of the way the company works – an opportunity for learning.
 - External participation – in this case the researchers – added value to the workshops by introducing an outside view on the way the company works

5 Discussion

In this section we discuss our experience with conducting process workshops in different contexts, and elaborate on what we have observed to be the strengths and weaknesses of this approach to software process improvement.

Let us first examine some differences between the two companies and how they chose to employ the process workshop, we have made a comparison in Table 3 below:

Table 3. Comparing Alpha and Beta

Alpha Company	Beta Company
Medium sized company 60 employees	Small sized company 20 employees
Mostly in-house projects for external customers	Mostly external projects at customer sites
ISO 9001-2000 certified	No formal certification
Extensive quality system was already in place before the researchers arrived, but it had become outdated and was too cumbersome to use.	No quality system or defined process in place. Each project followed its own process.
Management of the improvement project was handled by a separate Quality Assurance department.	No Quality Assurance department exists, the improvement project was handled by a project manager.
The improvement project had good anchoring with both management and developers through the QA department.	The improvement achieved good anchoring with the developers who participated in the workshops but suffered from poor anchoring with management.
PWS used to define the future process based on best practice. (Prescriptive modeling)	PWS used to understand the current process. (Descriptive modeling)
More than 20 people, 1/3 of the employees, participated in one or more workshops.	5 developers participated in two or more workshops while another 5 participated in one of the six workshops.
Half work-day workshops. ~4 hours	~3 hour workshops after office hours.
Responsibility for documentation of the workshop results was distributed among the participants.	Responsibility for documentation of the workshop results was left to the researchers.
Activity focus in the workshops.	Document focus in the workshop.
Evaluation of the PWS based on researcher observations and observations of the use of the electronic process guide.	Evaluation of the PWS based on researcher observations, post mortem analysis with PWS participants, and interview with the project manager responsible for the SPI effort.

The largest difference between the workshop methods employed in the two companies is the focus of defining future processes based on best practice in Alpha vs. defining the current process in Beta. Originally Beta wanted to define a future process, but given the different processes that emerged through the workshops, it was decided at an early stage to focus on the current processes. In retrospect we can explain the difference in focus with the situation the companies was in at the beginning of the improvement projects. The employees at Alpha were already used to using a defined process, while Beta had no experience on using a company process. This can also be linked to the project profile in the companies. While Alpha had fairly

homogeneous projects, Beta had a heterogeneous profile, with many consultants spread over several external customer sites.

The difference in previous process knowledge also manifested itself in the discussions and subsequently in the results of the workshops. While the employees at Alpha was more comfortable discussing activities, or *how* things should be done, the employees at Beta gravitated towards discussing documents or artifacts, or *what* should be done. That being said, the discussions at both companies kept discussions on the activities of the process to a fairly high level. Neither descended into a detailed description of how an activity should be carried out. The tendency of workshop participants to keep the discussion on a high level is also noted in the study by Pikkarainen [19].

Another result from our two case companies is that management support and involvement is a major success factor. This is nothing new in the literature [13], but we believe it deserves mentioning. At Alpha we had the support of top management through the QA department. At Beta top management was interested, but did not have the time or resources necessary to follow the project. This resulted in other external projects taking precedence over the improvement project. There was also no external drive towards formal certification like there was at Alpha, which could have increased the importance of the improvement project. This can also be explained through Beta's relatively small size, with only 20 employees, putting bread on the table and paying the bills came first. There were not enough resources to dedicate an employee to driving the project. The practical result has been that the researchers have had to provide some of the drive, and the project has taken longer time than anticipated.

Even though there were differences in the premises for the process workshops and slight differences in the execution, both Alpha and Beta employees praised it as a good arena for learning. The project workshops provided an arena where employees from several departments could meet and discuss. This gave the participants a broader view of how work was conducted in the organization. Through this open forum, the employees could discuss and reflect on their own work methods. Not being forced into a new process by external consultants or a distant QA department, creates an arena and opportunity for what Argyris and Schön [2] describes as double looped learning. Pikkarainen et al. [19] also found the workshop approach a good support for organizational learning.

Another effect we observed in both Alpha and Beta was that involvement in the process workshop created ownership of the resulting process. This effect was studied in Alpha, where it was shown that the participants of the workshop used the resulting process guide much more than the employees who did not participate. Although Beta has not implemented the resulting process yet, there have already been indications that there is a difference between those who participated and those who did not.

6 Conclusion and Further Work

We have conducted empirical studies on the application of the process workshop approach in two software companies. Our research question was "*How do available information, company context and goals affect the execution and results of process workshops?*" Based on the results and the previous discussion, we can conclude that:

- The premises of the company will strongly influence the execution of the project workshops. If the employees of a company are used to working according to a

process, the workshops can be used to formulate the starting point of a new process based on best practice. If, however, no clear process exists, the focus of the workshops should be on reaching an agreement on the current process before improvement is suggested.

- If the PWS approach is used to reach an agreement on the current process, a good starting point is to focus the discussion on artifacts, or what should be produced, rather than how it should be produced.
- If the PWS approach is used to specify future processes based on best practice, the discussions should be focused towards activities, or how the projects should be run.

In addition to answering our research question, we have made three observations pertaining to organizational learning and some related issues:

- The PWS approach is a good tool for organizational learning. Through the discussions in the workshops, the employees start the learning process, even before the process is available through a process guide.
- Involvement in the workshops fosters ownership of the resulting process, and as such it is a good way to get the developers to actually use the process later.
- The process workshop is a lightweight approach to defining a process for companies. As such it is well suited to small and medium sized companies. It does, however, require some resources to be truly successful and therefore, management support is important.

Further work in this area will be to investigate methods to spread the acceptance and usage of the resulting process. In a previous study [17] we showed that participants had a higher usage level of the resulting process than those who did not participate. In the empirical studies reported in this paper, we had a participant level of about 1/3 of the employees in each company. The challenge now becomes how to get the rest of the employees involved to foster a higher acceptance level of the resulting process.

Acknowledgments. This work was conducted as a part of the Software Process Improvement through Knowledge and Experience research project, supported by the Research Council of Norway through grant 156701/220. We are very grateful to our contact persons in the software consulting company for providing stimulating discussions and help with organizing the work.

References

1. Ahonen, J.J., Forsell, M., Taskinen, S.-K.: A Modest but Practical Software Process Modeling Technique for Software Process Improvement. *Software Process Improvement and Practice* 7(1), 33–44 (2002)
2. Argyris, C., Schön, D.A.: *Organizational Learning II: Theory, Method and Practise*. Addison Wesley, London (1996)
3. Avison, D., Baskerville, R., Myers, M.: Controlling Action Research Projects. *Information Technology & People* 14(1), 28–45 (2001)
4. Avison, D., Lau, F., Myers, M., Nielsen, P.A.: Action Research. *Comm. ACM* 42(1), 94–97 (1999)
5. Avison, D.E., Fitzgerald, G.: *Information Systems Development: Methodologies, Techniques and Tools*, 2nd edn. McGraw-Hill, New York (1995)

6. Baumgartel, H.: Using employee questionnaire results for improving organizations: The survey feedback experiment. *Kansas Business Review* 12, 2–6 (1959)
7. Becker-Kornstaedt, U.: Towards Systematic Knowledge Elicitation for Descriptive Software Process Modeling. In: Bomarius, F., Komi-Sirviö, S. (eds.) *PROFES 2001*. LNCS, vol. 2188, pp. 312–325. Springer, Berlin, Heidelberg (2001)
8. Carvalho, L., Scott, L., Jeffery, R.: An exploratory study into the use of qualitative research methods in descriptive process modelling. *Information and Software Technology* 47(2), 113–127 (2005)
9. Davison, R., Martinsons, M.G., Kock, N.: Principles of canonical action research. *Information Systems Journal* 14(1), 65–86 (2004)
10. Deming, E.W.: *Out of the Crisis* (first published in 1982 by MIT Center for Advanced Educational Services). The MIT Press, Cambridge, Massachusetts (2000)
11. Dingsøyr, T., Moe, N.B., Dybå, T., Conradi, R.: A workshop-oriented approach for defining electronic process guides - A case study. In: Acuña, S.T., Juristo, N. (eds.) *Software Process Modelling*, Kluwer International Series on Software Engineering, pp. 187–205. Kluwer Academic Publishers, Boston (2005)
12. Dingsøyr, T.: Postmortem reviews: purpose and approaches in software engineering. *Information and Software Technology* 47(5), 293–303 (2005)
13. Dybå, T.: An Empirical Investigation of the Key Factors for Success in Software Process Improvement. *IEEE Transactions on Software Engineering* 31(5), 410–424 (2005)
14. Guzzo, R.A., Dickson, M.W.: Teams in organizations: Recent research on performance and effectiveness. *Annual Review of Psychology* 47, 307–338 (1996)
15. Lawler, E.E., Mohrman, S.A.: Quality Circles - after the Honeymoon. *Organizational Dynamics* 15(4), 42–54 (1987)
16. Lehman, M.M., Belady, L.A.: *Program Evolution: Processes of Software Change*. Academic Press, San Diego (1985)
17. Moe, N.B., Dingsøyr, T.: The impact of process workshop involvement on the use of an electronic process guide: a case study. In: 31st EUROMICRO Conference on Software Engineering and Advanced Applications, pp. 188–195 (2005)
18. Nilsen, K.R.: Process improvement through development of an extended electronic process guide - from electronic process guide to integrated work tool. In: *EuroSPI 2004*. Trondheim (2004)
19. Pikkarainen, M., Tanner, H., Lehtinen, J., Levonmaa, M., Hyry, H., Abrahamsson, P.: An Empirical Evaluation of the Process Workshop Approach. In: 3rd International Conference of Software Development (2005)
20. Purser, R.E., Cabana, S.: Involve employees at every level of strategic planning. *Quality progress* 30(5), 66–71 (1997)
21. Riordan, C.M., Vandenberg, R.J., Richardson, H.A.: Employee Involvement Climate and Organizational Effectiveness. *Human Resource Management* 44(4), 471–488 (2005)
22. Scupin, R.: The KJ Method: A Technique for Analyzing Data Derived from Japanese ethnology. *Human Organization* 56(2), 233–237 (1997)
23. Susman, G., Evered, R.: An assessment of the scientific merits of action research. *Administrative Science Quarterly* 23(4), 582–603 (1978)
24. Vandenberg, R.J., Richardson, H.A., Eastman, L.J.: The Impact Of High Involvement Processes on Organizational Effectiveness. *Group & Organization Management* 24(3), 300–339 (1999)
25. Wenger, E.: *Communities of practise: learning, meaning and identity*. Cambridge University Press, Cambridge, UK (1998)

Improving an Industrial Reference Process by Information Flow Analysis: A Case Study

Kai Stapel, Kurt Schneider, Daniel Lübke, and Thomas Flohr

Software Engineering Group, Leibniz Universität Hannover,
Welfengarten 1, 30167 Hannover, Germany
{Kai.Stapel, Kurt.Schneider, Daniel.Luebke,
Thomas.Flohr}@inf.uni-hannover.de

Abstract. Reference processes are supposed to be the basis for collaboration and mature cooperation in software development. Large business organizations need large and diverse reference processes. However, process conformance is a constant concern. There are many explanations why a project may deviate from its reference process. This is especially true in larger software companies with a lot of different projects and variants modeled in a single reference process. During an industrial cooperation we have identified a phenomenon that adds to the problem: Unclear and incorrect information flows. Process modeling notations and practices in many large organizations nurture information flow anomalies. We improved the information flows in the reference software process by means of information flow analysis and flow patterns. A comprehensible reference process with reasonable information flows is easier to understand and therefore gains acceptance in the project team.

1 Introduction

Mature software organizations define and maintain software development processes based on CMMI [1] or SPICE (ISO 15 504). Modeling a reference process for software projects is a mandatory task in maturing environments. However, reference process models generate new problems. Correctness and conformance have been concerns for many years.

In this contribution, we report on collaboration with a financial institution. We were asked to check and improve their large reference process model. During that work, we identified an interesting class of problems that we traced back to information flow anomalies. We applied information flow analysis to tackle the problems.

The implementation of a large software development reference processes is difficult in itself. Challenges include:

- **Complex, unclear, incomprehensible reference processes:** To cover every intended use many processes tend to be complex. More documents and possible branches are modeled rather than keeping it simple and understandable. Unfortunate name assignments as well as sloppy descriptions make processes and

activities unclear. Furthermore the information flows are obscure. Complexity and the lack of clarity as well as badly designed information flows lead to incomprehensibility of the processes.

- **Unrealistic requirements:** Many processes require a lot of documentation work. Almost every activity demands a document as a result. Most of the time this is due to the fact that most process modeling techniques lack the ability to represent the flow of information in other representation media than documents. The reference process analyzed in our case study requires each and every project to produce between 60 and 165 documents.
- **Inflexible reference processes:** Since processes are designed to fit many needs they are often neither suitable for large nor for small projects. That is the reason why tailoring is needed before each project.
- **Faulty reference processes:** Reference processes are faulty in two ways: syntactically and semantically. Syntactical mistakes are caused by not following modeling standards and guidelines, unthoughtful modeling or insufficient support of the notation by the modeling tool. Semantic mistakes like multiple preparations of identical contents or parallel alteration of the same information are caused by e. g. distributed modeling and missing interface coordination. Hence, information flows are not only obscure but also faulty.

At least some of these problems can be found in most reference software development processes. As a consequence, many reference processes are not accepted by project teams. We analyzed such a software development process in a case study at an information technology service provider in the financial sector. We tried to narrow the gap between reference and actual processes by means of information flow analysis. Optimizing the information flows of a process leads to a more comprehensible and sound process which again affects its acceptance positively. The basic concepts of information flow analysis will be described in the following section.

2 Information Flow Analysis Concepts (FLOW Project)

In this section the basic concepts of information flow analysis are presented, as far as they are needed to understand the case study described in section 3. A detailed introduction can be found in [2, 3]. Our FLOW project was initiated at the Leibniz Universität Hannover in 2004. Besides the information flow analysis, FLOW is concerned with active strengthening and coordination of flows. We develop specific techniques and tools that improve information flows [3, 4]. However, these aspects are beyond this paper.

2.1 Goals of Information Flow Modeling

Information that flows in a process or project is modeled to accomplish several goals:

- Reflection and manual analysis by experts helps to remove flow anomalies and to shape information flows more adequate.
- Some anomalies can be described as information flow patterns. Pattern search can then be semi-automated.

- Techniques and tools can be developed that specifically affect information flows.
- Known existing techniques can be reframed and used for information flow improvement by specifying their information flow behavior.

2.2 Postulates of Information Flow Analysis

The approach of information flow analysis is based on some fundamental beliefs and assumptions. A detailed description can be found in [5] and [6]:

- *Information flows link processes and direct communication* and, therefore, also connect conventional and agile approaches.
- *Experience is a special kind of information* which is being modeled explicitly. It often influences activities and acts as a catalyst.
- We introduce the *state of information*: “Fluid” information is verbal or non-objectively reproducible information including e-mails and personal notes third parties cannot access or reproduce. “Solid” information refers to written or recorded information (like videos) which is long-term accessible to third parties.
- *Coarse modeling of information content*. Usually just requirements are modeled on their way through the project. In special cases (a few) different types of information can be used.

The last two points require explanation. Referring to information metaphorically as either fluid or solid states of information is typical for FLOW. It points out that the same information can appear in different shapes. Different states implicate different characteristics similar to fluid and solid matter.

- Solid information can be recalled at any time. It is stored in documents or recordings (video or audio). Access as well as storage cost time and effort, but they are repeatable. Solid experience is a special kind of solid information. It is available through checklists, best practices and (design) patterns.
- Fluid information, on the other side, is bound to people or other volatile media. We call “fluid” whatever someone has in mind and which cannot be obtained or reproduced by third parties. Fluid information can easily be transmitted by conversations and supported by some hand writings. But fluid information can easily be lost or forgotten. Fluid experience slowly grows in a person’s mind while the person is doing or observing something.

Self-restriction at modeling is also important: We explicitly do not aim for “as precise as possible” definitions and detailed distinctions between information flows. We rather prefer modeling as coarse as possible. Detailed distinctions in meaning or different forms of representation of information (as tables, texts, pictures) will not be differentiated in our FLOW models. Modeling information flows aims for qualitative optimization and not for exact mapping.

2.3 FLOW Notation






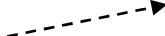

Information flows could be represented as data flow diagrams like in the 1970's [7]. The basic idea was – and still is – *not* to follow the control flow. It is rather important where information flows at all. A visual notation helps to clarify certain situations and

patterns. It ought to transport the basic concepts and still may be open for interpretations at some points. After all, it is a medium of communication for people and not a programming language. It is supposed to comply with the following requirements:

- Very easy understandable even for non computer scientists. Few, easy explainable symbols. Ability to extend existing process notations.
- Means of expression for fluid and solid, respectively for information storages as well as for information flows.
- Means of expression for experience and “other” information.
- The ability to establish relations to existing process notations.

Very, very simple but flexible notations were most appropriate wherever we used information flow analysis so far [3, 8-10]. That was also the case in the following case study. Table 1 depicts the symbols that satisfy the mentioned requirements.

Table 1. Symbols of information flow models; they can be used to extend other process notations

information state	store	information flow	experience flow	activity
solid	 <identifier>	 <information type> (optional)	 <experience> (optional)	 <activity>
fluid	 <identifier>	 <information type> (optional)	 <experience> (optional)	

The symbol for a fluid information store (smiley) is supposed to bring to mind that someone has the information in his or her head. Experience is distinguished from other information through a different color (in this case gray). The distinction between experience and other information is mentioned here for the sake of completeness. It is not needed in the following case study.

Additionally there is a link between FLOW and the used process notation. It is defined via the activity symbol (usually a rectangle) which is then available in both notations (FLOW and process). An activity incorporates incoming information and reissues it as an outgoing flow. The activity symbol adopts an extended meaning. This mechanism can be used to refine and structure combined FLOW-process models into hierarchies. Formally speaking, the respective activity symbol belongs to both notations.

Fig. 1 pictures a combined FLOW-process model. The process part with activities and documents is shaded gray. Below the process, FLOW symbols show requirements and design activities in a certain situation. Both solid (documents) and fluid (people) information flows appear. This composition needs to be optimized considering the characteristics of solid and fluid information. This is the goal of a FLOW software process improvement activity.

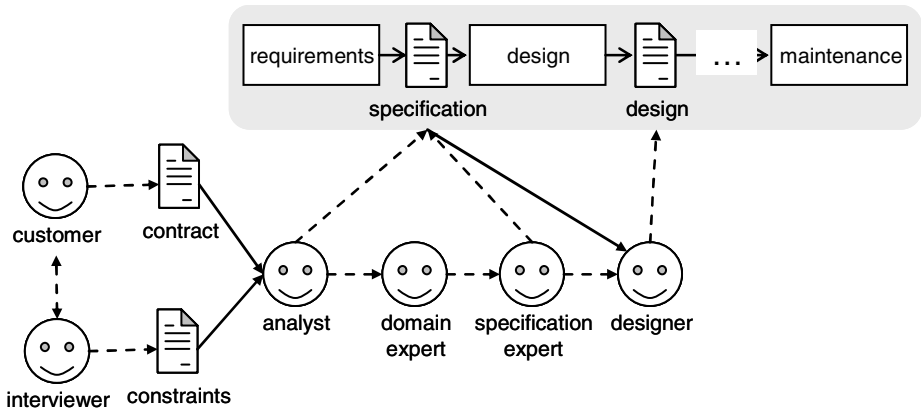


Fig. 1. Part of a combined FLOW-process model

A fairly large financial service provider asked us to help in optimizing their reference software development process. We used this opportunity to test the methods of FLOW in a case study.

3 Case Study

In a master thesis [11] we analyzed a very large existing reference software development process of an information technology service provider in the financial sector. We applied information flow optimization as described above. The respective company has more than 2500 employees. It has a large process model with 5 so called process groups, one of which is the analyzed software development process. It is an iterative incremental process aligned to RUP [12] that has grown historically. The software development process is structured in 6 layers to be clear and concise. All processes are maintained by a dedicated process department. It coordinates several process modelers from different specialist departments, the so called process experts. The process experts perform the actual process modeling. Many process experts and process managers have contributed to the reference process as it stands today.

The analyzed software development process contains most of the problems raised in the introduction, because of its size, its historical development and the many different modelers. It is complex, inflexible, unrealistically demanding and syntactically faulty. However, it is not just a messy and useless process model that the company should or could get rid of – instead, it is a typical representative of reference models in industry. Our goal was to identify causes and propose solutions for some problems that we would find using FLOW optimization. Since the only modeled information carriers in the process are documents and the document flow extraction is fairly easy we started with a document flow analysis.

3.1 Document Flow

Document flows are a special kind of information flows where the information is passed on exclusively by documents. For a detailed differentiation between document and information flows see [11].

In order to analyze document flows they first need to be extracted from the process model. Documents are modeled as prerequisites (inputs) and results (outputs) of activities. Activities are connected through a relationship of dependency or sequence. We say a “document flows” between activities A and B if the document is output of activity A and input to activity B, and if the two activities are connected via the directed sequence relationship. In the notation used, this relationship is denoted by an arrow. Document flow extraction is not as straight-forward as it might appear. In principle, one flow gets created for each document in the model.

The company uses a process modeling tool which is able to export the process in a proprietary XML graph format. We developed a tool that extracts the document flows from the XML representation and stores them in an open source graph file format, namely the Graph eXchange Language [13]. The advantage of using an open format is that any tool capable of reading and illustrating this format can be used to visualize the extracted document flows. Such visualization helps to understand a flow and makes it easier to find faulty flows.

We searched the process for anomalies aided by the extracted and visualized document flows. Identified problems fall into two groups: either syntactic or semantic document flow anomalies. Semantic anomalies are more severe and require manual analysis steps. Syntactic anomalies, however, can be detected automatically during the extraction process. The automatically found problems can be marked in the visualization to guide their tracking. For example, we found 33 documents that are created but never used, 70 documents that are needed in several activities but are never created and even 113 documents that are contained in the model but are never referenced by any activity, neither as input nor output. Altogether, 62% of all documents in the process are affected by one of these anomalies. It is no wonder that many project teams find the process confusing. Fortunately, purely syntactic anomalies are fairly easy to fix. Sometimes the process modelers just forget to connect a document to the according activity or two different names are used for the same document. Both can be fixed during revision. Many not referenced documents can be deleted from the process model, because they are not needed at all.

Semantic anomalies are more demanding, but also more rewarding to resolve. For instance, we found several cases in which the branching after activities that affect the further document flow (conditional activities) was missing. Fig. 2 depicts such a situation in a proprietary process notation used by the analyzed company.

The left side of Fig. 2 shows the quality assurance process as is. A design document is to be improved. A review step is supposed to make sure that the quality of the improved design is okay. A review just “looks” at a document but does not alter it. No changes are made during the review process. The actual problem with the left side process is that no matter what outcome of the review, the improvement process is finished according to the modeled process. Since a review is a conditional activity the process should look like the one on the right side including the condition and the branching. If the quality of the improved design is assured the process can stop and

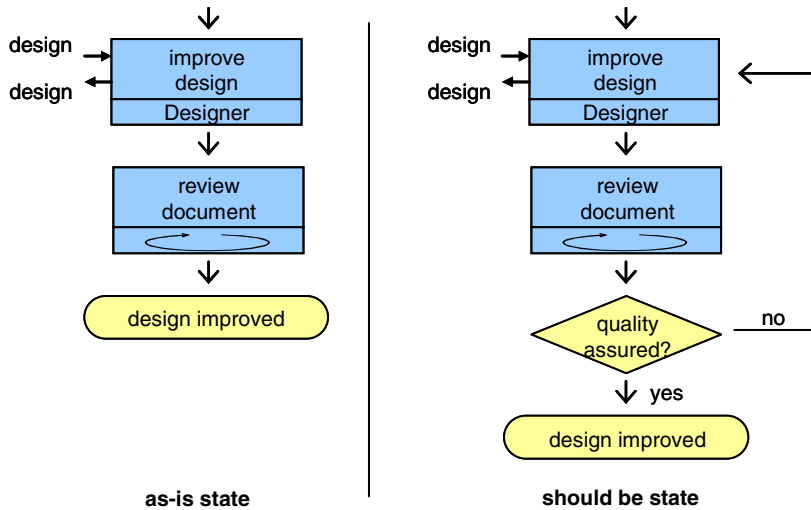


Fig. 2. Left side: as-is problem: no branching after conditional activities. Right side: Fixed process with condition and branching.

the document can be passed on (“yes” branch). But if the design is not good enough yet it has to be improved again (“no” branch).

Multiple occurrence of this problem leads to more confusion with the people who are supposed to work according to the process (the project teams). And finally confusion leads to less acceptance of the process. Teams are asking: “The process does not make sense to us, so why should we follow it”?

Many of the above-mentioned problems may sound ridiculous or avoidable. Nevertheless, we had seen many of them in different settings at different companies before. We decided it was more use to improve those real processes rather than to deny their existence.

What can be done to avoid these problems? It looks like the process modeling did not proceed accurately enough. The company should revise its process model and for instance add branches after each decision where they are not present already. We also proposed some extensions to the modeling guideline, so future revisions will be able to avoid the detected anomalies. The former modeling guideline did not include hints from the document flow point of view. That might be another reason for the many syntactic document flow anomalies.

Document flow analysis identified a lot of anomalies already. Most of them were syntactic problems with minor consequences caused by neglectful design. But there still are problems that cannot be identified by means of pure document flow analysis. What if a document is used to model direct communication, because there is no other way to do this with a given process notation? What if direct communication is not modeled at all?

3.2 Information Flow

Information flows are not limited to documented information being passed on. More effective ways of communication will also be considered, like conversations, phone calls, chats or e-mails (fluid information).

There are several ways to extract information flows from a process. The automatic extraction from the reference process model can be tried. A so-called document dependency graph can be used as a starting point. Such a graph is built by adding an edge from document A to B whenever A is output of an activity of which B is input. Assuming that the information in A depends on B an information flow from B to A can be inferred. With this assumption and the document dependency graph, information flows can automatically be extracted from the reference process. Documents as containers channel information through the project. The flows obtained in that way have two major disadvantages. First, the assumption does not always apply. Second, flows occur only between documents. People and direct communication are left out again.

This is due to a common weakness in most process notations: there is no way to refer to direct communication. To gain information flows including direct communication a process needs to be analyzed in a real project situation. An information flow expert could accompany a project and thereby note all flows. This is often not possible because of secrecy issues. Interviewing project participants is a less demanding approach.

In this case study we observed two extraction methods: The extraction from the reference process model and the extraction via interviews. Although these methods are not as effective and as accurate as e. g. the project monitoring they still produce useful information flows.

Reference Process Extraction

Our first approach of analyzing the information flows was the automatic extraction from the reference process model. To accomplish this we used the method described above. First, we created the document dependency graph from the exported process model. After that, presuming that document dependencies infer information flows, we extracted the flows and even found some anomalies.

One problem we found was a not fully connected document dependency graph. Most likely it was not fully connected because direct communication paths occurring in actual processes could not be modeled in the reference process. Several disjoint document clusters in the document dependency graph raise the question why there should be totally independent information flows in a single development process. Usually a project pursues only *one* goal. A process model should represent that by producing a fully connected document dependency graph. The single goal of a software development process is the creation of software. The main cluster of the document dependency graph correctly contains the corresponding documents. The other smaller clusters stem from some sub processes and should be connected to the main cluster, since the sub processes do not create information independently from the other development activities. However, they are not connected according to the reference process.

Two causes may be responsible for that anomaly: Modeling without due care and attention and shortcomings in the process notation. The first situation usually occurs because the modelers don't pay attention to the information flow perspective or they may not know about the right information flow analysis methods and tools. Even in case they are aware of information flows and breakdowns, they do not have or find a tool to deal with them.

The second situation is the lack of ability to address non documented information flows in most process notations. In *actual* software development processes there *is* a connection between the document clusters. The documents are not connected directly but via non documented information flows (fluid) like e-mails or verbal speech and notes like e. g. in meetings. So the anomaly is caused because fluid flows cannot be depicted in the model. Usually this kind of a problem gives the project teams a bad feeling. They know that something is wrong. But they do not exactly know what and where it is in the process when using the process view alone. Using the information flow perspective clarifies things because it helps to identify the problem and where exactly it occurs.

Other anomalies we identified were cyclic flows. They occur when two or more documents depend on each other in a cyclic way. In the majority of cases this problem can be solved by looking at the documents at the level of paragraphs. Usually paragraphs do not depend on each other anymore. Modeling granularity, thus, is a key concept to effective information flow analysis.

The anomalies derived by means of reference process extraction are a good starting point for more specific information flow analyses, like interviews of the people who perform the affected process. The information flow perspective also helps to explain suspicious parts of the process that were not clear from the process view alone.

Extraction by Interviews

The second information flow extraction method conducted was the extraction through interviews. Two project leaders of two different projects were interviewed. Both were supposed to use the same reference process model. But both described actual processes were much simpler and easier to understand. Hence, the information flows were not as confusing as the ones obtained from the reference process. We even found some information flow patterns. The *document creation process* looked similar in both actual processes and for each document to be created in them. First, the document is created on the basis of a template. Then the document iteratively gets extended or changed accompanied by a quality assurance step. This is the case for all types of documents like specifications, product manuals and even the source code. These patterns seem to be due to the compliance of the projects with the reference process. So there already is a certain level of compliance. The many anomalies usually hide that.

Beside the regular patterns we also found some problematic ones. One of which is the problem of creating a document afterwards. An example of this problem is depicted in Fig. 3.

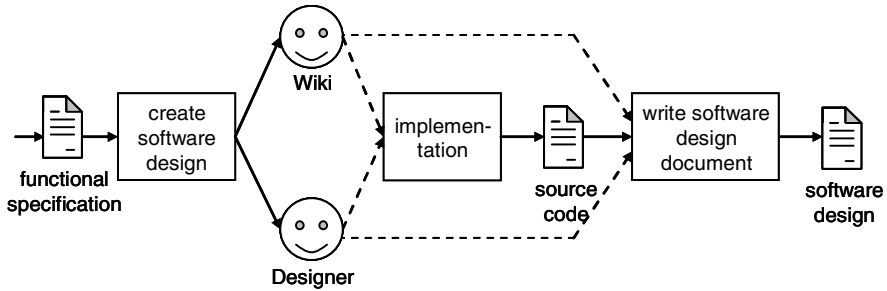


Fig. 3. Problem: Creation of design document afterwards

This FLOW shows a part of the actual software development process as described by one of the interviewed project leaders. Starting from the functional specification a software design is created. The devised design is kept in the minds of the designers, handwritten notes and a system similar to a wiki. This information is then used to implement the software. After that the design gets documented in the software design document although the implementation is already finished. But why did the document get created afterwards? That is because the reference process demands it. The process department supervises the reference process compliance of each project and penalizes non-compliances. To avoid the penalties the project created the document afterwards.

What is the solution to that problem? The process department would answer: “Use the reference process depicted in Fig. 4 where starting from the functional specification the software design gets created first and then based on that document the software gets implemented”. The project team would answer: “Use the process depicted in Fig. 3 without the additional creation of the software design document afterwards”. Both parties have good reasons for their opinions. The process department wants to make sure that everything gets documented correctly. The project team however wants to successfully finish the project in time and on budget. Too much documentation work slows things down unnecessarily.

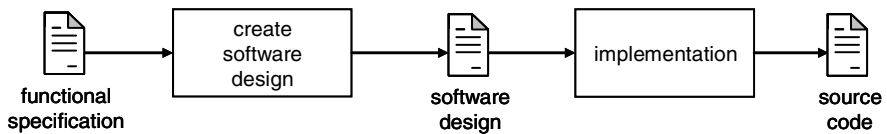


Fig. 4. Software development information flow according to reference process

FLOW provides a solution in terms of decision support rules:

- **Solid information flow:** Use a solid information flow whenever the information needs to be documented permanently. The solid state ensures traceability, third party accessibility and repeatable unmodified accessibility of the information.

- **Fluid information flow:** Use a fluid information flow whenever effective information transport is important. The fluid state (especially verbal communication) ensures the fast and extensive exchange of information.

These rules can be used to tailor a process. The result is a more effective process that produces adequate documentation.

In our case the project team and the process department should get together to decide what is most appropriate in the given situation by means of the FLOW decision support rules. This combined decision making also contributes to the narrowing of the gap between the reference and the actual development process.

In this case study we found most of the reference process problems stated in the introduction. By using FLOW techniques we could identify information flow anomalies and propose solutions. More concise and thus understandable information flows lead to better comprehensibility. The documentation requirements can be cut down by using fluid flows whenever reasonable. All this leads to a better acceptance of the reference process and therefore to better reference process conformance in the projects.

4 Related Work

Zuser et al. [14] requests communication flows to be analyzed and documented well (by models) in order to prevent redundant and superfluous flows. Efficient communication flows neither involve too many roles nor inadequate document forms. In this way loss-free communication can be established. Traceability is important to prevent the loss of already made decisions. That is why especially decisions need to be documented well. A modeling technique is introduced incorporating three kinds of communication flows: verbal communication, communication via written documents and communication via e-mail and talk memos. Furthermore, any flow holds information about the flowing data (e.g. a weekly report). The approach presented by Zuser et al. is fairly basic, because it does not mention patterns or techniques to capture these flows explicitly. Furthermore, it is limited to analyze communication in one work group and not in the whole process.

Hansen and Kautz [15] used so-called Knowledge Maps to identify and analyze knowledge flows. The technique aims to detect and optimize knowledge flows in order to distribute knowledge within a learning organization. Knowledge flows connect roles, individuals or organizational units, if knowledge is exchanged between these entities. These flows can be unidirectional or bidirectional. If necessary information about frequency, intensity, contents, context and importance can be attached to the flow. Hansen and Kautz identified four knowledge flow patterns: hubs, black-holes, springs and missing links. Hubs are connected to many other entities through flows; black-holes only consume knowledge while springs only produce it. Finally, missing links indicate situations where a flow should be established but it is absent. The authors successfully applied the Knowledge Map technique in a medium-size organization. However, the identified patterns are very basic: When using FLOW it is possible to detect other patterns like Chinese Whisper, in which information is passed along many hubs. Thereby, information is lost and gets modified.

5 Conclusions and Outlook

In this paper we presented an approach for optimization of development processes by means of information flow analysis. We started with an introduction of the FLOW project which investigates methods, techniques and tool support for information flow analysis. We then presented our experiences made using FLOW in industry. A large, complex, highly demanding and faulty reference process was analyzed. Problems from the information flow perspective were identified and solutions were proposed. This helped to clarify and optimize the reference process. Such an improved reference process will gain acceptance in the project teams and will therefore be implemented better. The gap between reference and actual development processes will be narrowed.

We first analyzed document flows in the case study because they can be extracted fairly easy from the present reference process. The document flows already contained a lot of anomalies. Most of them were of syntactical nature and hence easy to fix. The elimination of these anomalies already leads to a clearer and easier to understand process. But it still requires a lot of documentation work and some problems could not be identified with pure document flow analysis, yet.

We then performed the actual information flow analysis. In interviews positive information flow patterns were identified. Incorporating them into the reference process helps to improve comprehensibility. Aside from the positive patterns we also identified patterns indicating anomalies. Documents were created after they could have been used purposefully just because the reference process required it. The relevant information had been passed on before without the use of “solid” documents. The FLOW decision support rules help to correct this situation by establishing either fluid (e.g. direct communication) or solid (e.g. documents) information flows. Doing both is unnecessary extra work.

Some anomalies detected were due to the missing ability of most process notations to model fluid information flows in the reference process. Existing process modeling tools need to be extended to incorporate effective communication methods (conversation, e-mail, chat, video conference). One way to accomplish that would be to enable combined FLOW-process models.

References

1. Ahern, D.M., Clouse, A., Turner, R.: CMMI® Distilled: A Practical Introduction to Integrated Process Improvement. Addison-Wesley, Reading, MA (2001)
2. Schneider, K.: Software Process Improvement from a FLOW Perspective. In: Accepted for the Workshop on Learning Software Organizations (LSO, 2005). Kaiserslautern (2005)
3. Schneider, K., Lübke, D.: Systematic Tailoring of Quality Techniques. In: World Congress of Software Quality, Munich, Germany (2005)
4. Schneider, K.: Rationale as a By-Product. In: Dutoit, A.H., et al. (ed.) in Rationale Management in Software Engineering, Springer, Berlin (2006)
5. Schneider, K.: Aggregatzustände von Anforderungen erkennen und nutzen. In: GI-Fachgruppentreffen Requirements Engineering 2005. Hannover, Germany (2006)
6. Schneider, K.: Software-Engineering nach Maß mit FLOW. In: SQMcongress 2006. Düsseldorf, Germany: SQS (2006)

7. DeMarco, T.: *Structured Analysis and System Specification*. Prentice-Hall, Englewood Cliffs (1979)
8. Allmann, C., Winkler, L., Kölzow, T.: *The Requirements Engineering Gap in the OEM-Supplier Relationship*. In: LSO+RE 2006. Hannover, Germany (2006)
9. Lübke, D., Schneider, K.: *Leveraging Feedback on Processes in SOA Projects*. In: Richardson, I., Runeson, P., Messnarz, R. (eds.) *Software Process Improvement*. LNCS, vol. 4257, Springer, Heidelberg (2006)
10. Schneider, K.: *LIDs: A Light-Weight Approach to Experience Elicitation and Reuse*. In: Bomarius, F., Oivo, M. (eds.) *PROFES 2000*. LNCS, vol. 1840, Springer, Heidelberg (2000)
11. Stapel, K.: *Informationsflussoptimierung eines Softwareentwicklungsprozesses aus der Bankenbranche*, in *FG Software Engineering*. Leibniz Universität Hannover: Hannover (2006)
12. Kruchten, P.: *The Rational Unified Process: An Introduction*, 3rd edn. Addison-Wesley Professional, London (2003)
13. Holt, R., et al.: *Graph eXchange Language* (2002), <http://www.gupro.de/GXL/>
14. Zuser, W.: *Software-Engineering mit UML und dem Unified Process*. München: Pearson Studium, p. 377 (2001)
15. Hansen, B.H., Kautz, K.: *Knowledge Mapping: A Technique for Identifying Knowledge Flows in Software Organisations*. In: Dingsøy, T. (ed.) *EuroSPI 2004*. LNCS, vol. 3281, pp. 126–137. Springer, Heidelberg (2004)

Connecting the Rationale for Changes to the Evolution of a Process

Alexis Ocampo and Martín Soto

Fraunhofer Institute for Experimental Software Engineering
Fraunhofer-Platz 1
67663, Kaiserslautern and Germany
{ocampo,soto}@iese.fraunhofer.de

Abstract. In dynamic and constantly changing business environments, the need to rapidly modify and extend the software process arises as an important issue. Reasons include redistribution of tasks, technology changes, or required adherence to new standards. Changing processes ad-hoc without considering the underlying rationale of the process design can lead to various risks. Therefore, software organizations need suitable techniques and tools for storing and visualizing the rationale behind process model design decisions in order to optimally introduce future changes into their processes. We have developed a technique that support us in systematically identifying the differences between versions of a process model, and in connecting the rationale that motivated such differences. This results in a comprehensive process evolution repository that can be used, for instance, to support process compliance management, to learn from process evolution, or to identify and understand process variations in different development environments. In this article, we explain the underlying concepts of the technique, describe a supporting tool, and discuss our initial validation in the context of the German V-Modell XT process standard. We close the paper with related work and directions for future research.

1 Introduction

The field of software process modeling has become established within the software engineering community. An explicit process model is a key requirement for high productivity and software quality. The process description content might be collected in several ways, for example by observing real projects, describing intended activities, studying the literature and industry reports, or interviewing people involved in a project [10]. Usually, considerable effort is invested into the definition of such processes for an organization. Once the process is defined and institutionalized, modifying it further becomes unavoidable due to various reasons, such as the introduction of a new software development technology in a development team (e.g., new testing support tools and techniques), a new/updated process engineering technology (e.g., a new process modeling technique), new/updated standards/guidelines for software development or process engineering, new/updated regulatory constraints, or new/updated standards/guidelines for software development or process engineering. Such changes must be reflected accordingly in the corresponding process models. Achiev-

ing a compromise that satisfies such a challenge usually depends on the information available for rapidly judging if a change is consistent and can be easily adopted by practitioners.

Having information about the reasons for process changes (i.e., the *rationale*) at hand can be of great help to process engineers for facing the previously mentioned challenges. Currently, the common situation is that there is a lack of support for systematically evolving process models. Combined with other facts such as budget and time pressure, process engineers often take shortcuts and therefore introduce unsuitable or inconsistent changes or go through a long, painful update process. In many cases, precipitous and arbitrary decisions are taken, and process models are changed without storing or keeping track of the rationale behind such changes.

According to our experience, systematically describing the relationships between an existing process and its previous version(s) is very helpful for efficient software process model evolution [2]. Such relationships should denote differences between versions due to distinguishable modifications. One can identify the purpose of such modifications if one can understand the rationale behind them. Rationale is defined as the justification of decisions [8]. Rationale models represent the reasoning that led to the system or, in our case, to the process in their current form. Historically, much research about rationale has focused on software/product design. By making rationale information explicit, decision elements such as issues, alternatives, criteria, and arguments can improve the quality of software development decisions. Additionally, once new functionality is added to a system, the rationale models enable developers to track those decisions that should be revisited and those alternatives that have already been evaluated.

We are currently working on transferring rationale concepts into the process modeling domain. We do this based on the assumption that the rationale for process changes can be used for understanding the history of such changes, for comprehensive learning, and for supporting the systematic evolution of software processes. We are looking at the possibilities that can be used for documenting changes and connecting them to their corresponding rationale.

This article presents a technique for comparing process models, recognizing a set of standard changes, and connecting them to their respective rationale as follows: In Section 2, we present the conceptual model for capturing rationale; In Section 3, we present the technique for identifying changes. In Section 4, we illustrate the connection of changes to rationale; In Section 5, we briefly discuss our implementation of this technique, as well as our experience in applying it to the German V-Modell XT [33] process standard. In Section 6, we present a short description of related approaches for comparing models and for capturing rationale; and finally, in section 7, we provide a summary and future research questions to be resolved.

2 Process Rationale

The following is a conceptual model that can be considered a second version of our attempt to understand the information needs for capturing the rationale behind process changes (see Fig 1). The results of our first attempt have been documented in [21]. In

order to complement our previous work, we decided to take the IBIS [14], QOC [18], and DRL [15] approaches as the basis. These approaches are called *argumentation-based* because they focus on the activity of reasoning about a design problem and its solution [8]. Based on our previous experience in collecting rationale [21], we assessed these approaches and defined a small subset of entities (see shadowed classes in Fig 1) that suited our goal, namely capturing rationale in a more structured way, while being careful of keeping the involved costs under control especially because this issue has been highly criticized by rationale experts. We connected these basic entities to three additional entities relevant for us, namely, event, changes, and process entities (the non-shadowed classes in Fig. 1).

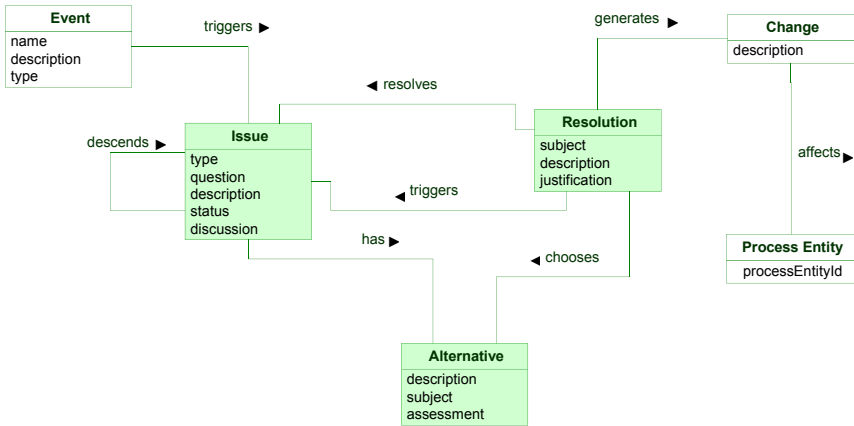


Fig. 1. Rationale Model (UML static structure diagram)

An *event* is considered to be the trigger of issues. Events can be *external* or *internal*. Examples are:

- External: new/updated process engineering technology (e.g., a new process modeling technique); new/updated regulatory constraints
- Internal: responses to failures to pass internal or external appraisals, assessments or reviews (e.g., changes needed to address a failure in passing an FDA audit); new/updated best practices emerging from "lessons learned" in just-completed projects (e.g., a new "best practice" approach to handling design reviews).

Issues can be problems or improvement proposals that are related to a (part of a) process and that need to be addressed. Issues are stated usually as questions in product-oriented approaches. In this work, the question has the purpose of forcing process engineers to reason about the situation they are facing. Additionally, an issue also contains a long description, a status (open, closed) and a discussion. The discussion is intended for capturing the emails, memos, letters, etc. where the issue was treated by process engineers. Additionally, an issue can be categorized by a type. This type can be selected from a classification of issues that needs to be developed or customized for an organization. It is possible to start with a preexisting classification (see for example [21]) as a basis, which can be refined based on experience gained from process evolution projects.

Alternatives are proposals for resolving the issue. Alternatives can be captured with subject (short description) or long descriptions. Alternatives are evaluated and assessed by process engineers regarding their impact and viability.

A *resolution* chooses an alternative that gets implemented by changing the process model. At the same time, one resolution could lead to opening up more issues. Note that a resolution has a subject (short description), a long description, and a justification. The justification is intended for capturing a summary of the analysis of the different alternatives, the final decision, and the proposed changes. *Changes* are the result of the decision captured in the resolution. They are performed on process entities. Some examples of changes performed to process entities are: activity x has been inserted; artifact y has been deleted; activity x has been moved to be a sub-activity of activity z .

Identifying which changes affect which process element is not an easy task. So far, we have developed a mechanism that allows us to document the rationale information proposed in Fig. 1 directly in the process model being altered [22]. The actual rationale information can be documented in special tables at the end of the process model description. The process engineer can then introduce the rationale information, perform the changes to the respective process entities, and then establish a reference to the corresponding rationale element. Then with the support of a special tool, the process evolution repository is updated. Although the approach proves to be suitable, we saw the need to provide more flexibility during the identification of process entities being changed and the documentation of the rationale behind such changes. In the following section, we present the technique we developed for that purpose.

3 Pattern-Matching-Based Change Identification

Our change identification technique is based on the Delta-P approach for process evolution analysis [28, 29]. This technique makes it possible to handle a wide variety of types of changes in a completely uniform way, to flexibly define the types of changes that are considered interesting or useful (this can be based on the structure and semantics of the process' metamodel), and to restrict the results to only certain types of changes, or even to certain interesting portions of a model.

3.1 A Normalized Representation for Process Models and Their Comparisons

Our first step consists of representing models (and later their differences) in such a way that a wide range of change types can be described using the same basic formalism. The representation we have chosen is based on that used by RDF [19] and similar description or metadata notations. For our purposes, this notation has a number of advantages over other generic notations:

- Being a generic notation for graph-like structures, it is a natural representation for a wide variety of process model types.
- It has a solid, standardized formal foundation.
- As shown below, the uniformity of the notation, which does not differentiate between relations and attributes, makes it possible to describe a wide range of changes with a straightforward pattern-matching notation.

- Also as shown below, the fact that many model versions can be easily put together into a single model makes it possible to use the same pattern-matching notation for single model versions and for comparisons.

Fig. 2 shows an example of this representation. The graph contains only two types of nodes, which we will call *entity* nodes (ovals in the figure) and *value* nodes (boxes in the figure). Entity nodes have arbitrary identifiers as labels. Value nodes are labeled by the value they represent, which can belong to a basic type (string, integer, boolean, etc.)

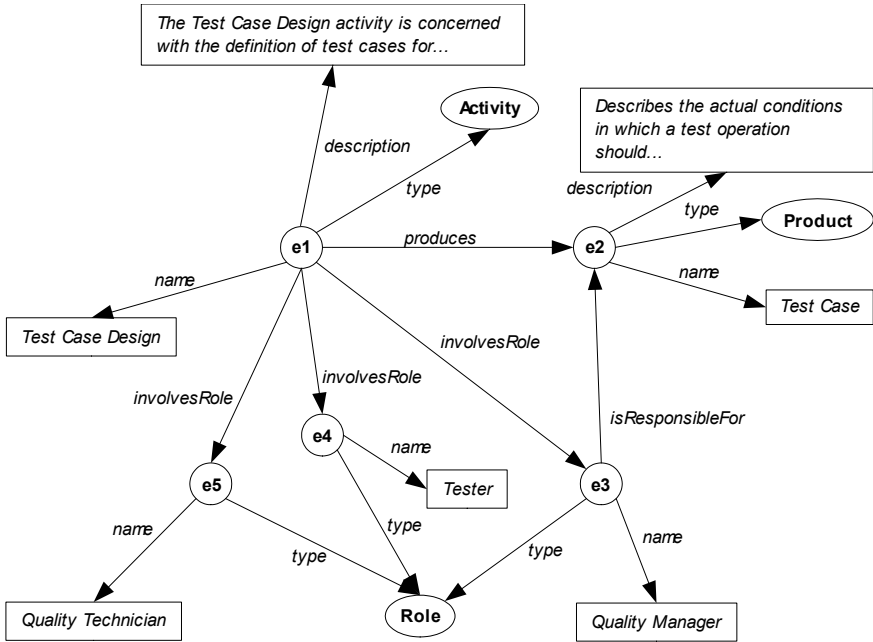


Fig. 2. A process model in normalized form

Arrows represent typed directed relations (type is given by their labels). Relations may connect two entity nodes, or may go from an entity node to a value node. It is not allowed for a relation to leave a value node. It is also not allowed for a node to exist in isolation. All nodes must be either the start or the end point of *at least* one relation. It follows that the graph is characterized completely by the set of the relations (edges) present in it, since the set of nodes is exactly the set of all nodes that are the start or the end of an edge.

The correspondence between attributed graphs and this normalized form is straightforward:

- *Entities and types correspond to entity nodes.* For each entity instance and entity type in the original graph, there is an entity node in the normalized graph. There is also a *type* relation between each node representing an entity and the node representing its type.
- *Attributes correspond to entity-value relations.* For each entity attribute in the orig-

inal graph, there is a relation labeled with the attribute name that connects the entity with the attribute value (that is, attributes in the original metamodel are converted into relation types). The value is a separate (value) node.

- *Entity relations correspond to entity-entity relations.* For each relation connecting two entities in the original graph, a relation connecting their corresponding entity nodes is present in the normalized graph.¹

Fig. 3 shows an evolution of the model presented in Fig. 2, using the normalized notation, with changes also highlighted. Formally, this graph respects exactly the same restrictions as the normalized model representation. The only addition is that edges are *decorated* (using interrupted and bolder lines) to state the fact that they were deleted, added, or simply not touched. This leads us to the concept of a *comparison graph* or *comparison model*. The comparison model of two normalized models A and B contains all edges present in either A or B, or in both, and only those edges. Edges are marked (decorated) to indicate that the edge is only in A, only in B, or in both A and B.

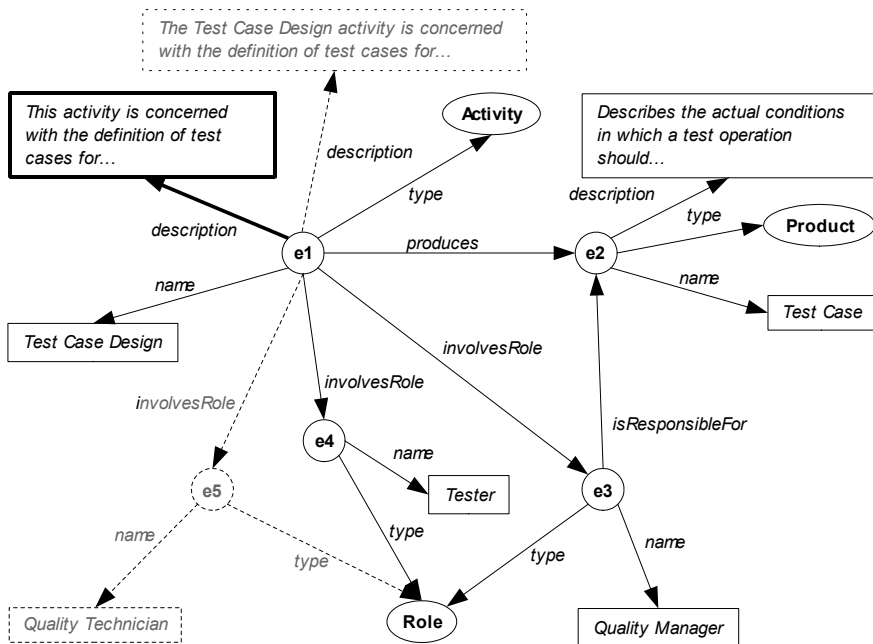


Fig. 3. A process model comparison in normalized form

The main aspect to emphasize here is the fact that all changes are actually reduced to additions and deletions of relations between nodes. This results in part from the fact that attributes are represented as relations, but also from the fact that nodes cannot exist in isolation. It is possible (and safe) to identify entity additions and deletions by looking for additions and deletions of *type* relations in the model.

¹ Relations with attributes can be modeled by introducing entity nodes that represent them, but the details are beyond the scope of this paper.

The fact that the normalized representation reduces all changes to sets of relation additions and deletions is useful because it permits describing many types of changes uniformly. On the other hand, an adequate formalism to describe changes must have clear, unambiguous semantics, and must be, at the same time, accessible to users. The following sections discuss with the help of examples, the mechanism that we have chosen for this task: a graphical pattern-matching language.

3.2 Example 1: Additions and Deletions

Our first example is related to one of the simplest possible model changes: adding or deleting process entities. Fig. 4 shows four patterns that identify changes of this type with different levels of generality. The pattern in Fig. 4a) matches all additions of process activities, and for each match, sets the *?id* variable with the identifier of the new activity. In a similar way, the pattern in Fig. 4b) matches all deletions of process products. These patterns can be generalized to identify arbitrary additions and deletions: the pattern in Fig. 4c) identifies all entity additions, and instances an additional variable with the type of the added entity. Finally, Fig. 4d) shows a pattern that not only finds new activities, but sets a variable with the corresponding name, a useful feature if the results of matching the pattern are used, for example, to produce a report.

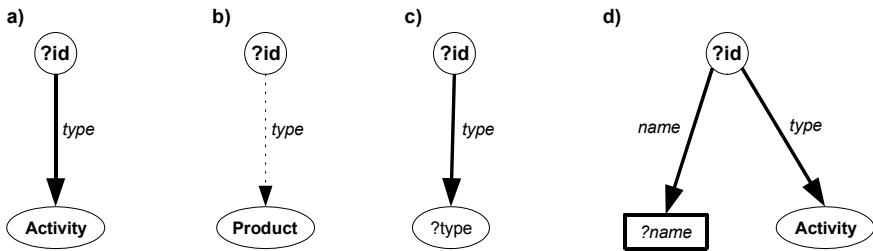


Fig. 4. Patterns for identifying entity additions and deletions

3.3 Example 2: Changes in Attribute Values

Just as important as identifying entity additions and deletions is finding entities whose attributes were changed. Fig. 5 shows three patterns that describe changes in attribute values. Fig. 5a) is basically an excerpt from the comparison graph in Fig. 3, which captures the fact that an attribute *description* was changed. This pattern, however, matches only the particular change shown in the example. The pattern in Fig. 5b) is a generalized version of the first one. By using variables for the entity identifier, as well as for the old and new property values, this pattern matches all cases where the *description* attribute of an arbitrary entity was changed. Note that each match sets the value of the *?id* variable to the identifier of the affected entity, and the values of *?oldValue* and *?newValue* to the corresponding old and new values of the *description* property. The pattern in Fig. 5c) goes one step further and uses a variable for the attribute labels as well, which means it matches all attribute value changes. Note that

these patterns match once for *each* changed property in *each* object. Finally, the pattern in Fig. 5d) constitutes a specialization of its peer in Fig. 5c): it is restricted to all attribute value changes affecting process activities.

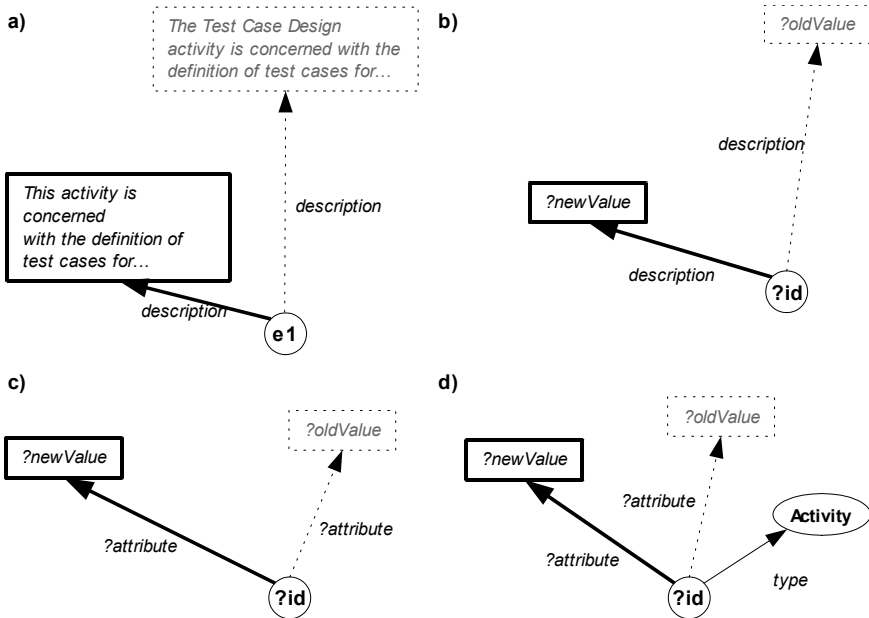


Fig. 5. Four patterns for identifying attribute value changes

Changes identified in this way can be fed into additional algorithms that perform attribute-specific comparisons, such as, for example identifying added or deleted individual words or characters in text-based attributes. This way, potentially expensive specific comparison algorithms are only applied to relevant items.

4 Connecting Rationale to Process Changes

RDF has been designed with the intention of supporting the interchangeability of separate packages of metadata defined by different resource description communities. We have defined a separate RDF model containing the rationale concepts described in Fig. 1. This allows us to reference the comparison model. Fig. 6 elaborates on the previous example and illustrates how we connect the comparison model to the rationale model. The figure can be interpreted as having two separated RDF models, one for the comparison of processes and the other one for the rationale for changes. Let us assume that a review board met, discussed, and documented issue (i1) concerning the expensive performance of the activity Test Case Design (e1) by more than one role, i.e., Quality Manager (e3), Tester (e4), and Quality Technician (e5). The review board analyzed two different alternatives (a1) and (a2), resolving to reduce costs (r1) by removing the Quality Technician (e5) from the list of roles responsible for the Test Case Design (e1). Afterwards, appropriate changes were performed to the process.

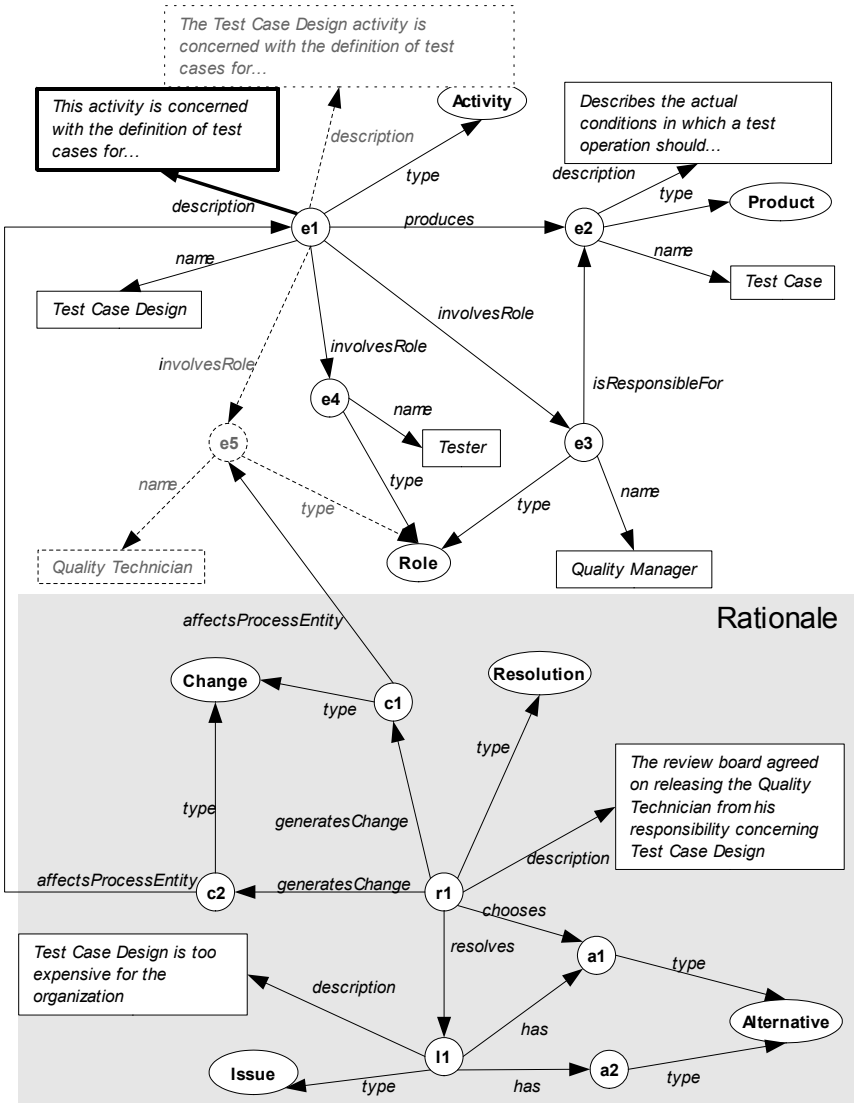


Fig. 6. Connecting the rationale model to the comparison model

Using the pattern for identifying entity additions and deletions (see Fig. 4), we can detect that entity *e5* has been deleted. This can be counted as a change and labeled as *c1*. We can then connect change *c1* to the process entity *e5* through the property *affectsProcessEntity*. Also, by using the pattern for identifying attribute value changes, we can detect that the description has been modified. This change can be counted and labeled as *c2*. As in the previous case, we use the property *affectsProcessEntity* to connect *c2* to *e1*. Both changes were generated by a single decision, which in this case corresponds to *r1*. We can then connect *r1* to *c1* and *c2* through the *generatesChange* property. The resolution *r1* is also connected to the other elements of the rationale

model, namely alternatives and issues. This way we can go through comparison models identifying changes and connecting them to their corresponding rationale model.

5 Implementation and Validation

An implementation of the pattern-matching based change identification technique presented in the previous sections is available as part of the *Evolyzer* tool [29]. We have tested our approach and tools by applying them to the various official releases of the V-Modell XT [33], a large prescriptive process model intended originally for use in German public institutions, but finding ever increasing acceptance from the German private sector. As of this writing, the *Evolyzer* tool still lacks a graphical editor for change patterns. However, patterns can be expressed as textual queries using a syntax that basically follows that of the emerging SPARQL [25] query language for RDF. Expressed as queries, patterns can be executed to find all their occurrences in a model. The V-Modell XT constitutes an excellent testbed for our approach and implementation. Converted to the normalized representation defined in Section 3.1, the latest public version at the time of this writing (1.2) produces a graph with over 13,000 edges. This makes it a non-trivial case, where tool support is actually necessary to perform an effective analysis of the differences between versions. Our first trial with the V-Modell XT consisted of analyzing the evolution of the V-Modell XT itself, where we compared 559 model versions that were produced in 20 months of actual model development. First, we normalized each release by using a parser we implemented in the interpreted, object-oriented Python programming language [17] which is able to navigate through the XML-specific version of the V-Modell XT, identifying the entities and properties, and moving this information to a process evolution repository. The rationale model was obtained from processing the information stored in the bug tracking system used for the continuous improvement of the V-Modell XT. This system supports the change management process designed for the model. Users can report problems and provide improvement suggestions in the form of change requests. Such change requests are processed by the team responsible for the model. The resulting analysis and decisions are also documented in the tracking system. Once the approved changes are finished (using specialized model editing tools) a new version of the V-Modell XT is stored in a configuration management system. In order to keep track of the decisions implemented, the V-Modell XT team member provides a short description of the change with a reference to the respective change request. We processed the information contained in the bug tracking system as well as in the configuration management system to create a rationale RDF model. With this rationale model as a basis, we proceeded to calculate changes between versions, connect them to the rationale model (as explained in the previous section), and store them in our process evolution repository.

By using our patterns, we found 19104 changes between version 1 and version 559. 30% corresponded to entity additions, 27% to entity deletions, and 63% to attribute modifications.

Fig. 7 shows the results of querying the rationale for process evolution repository using SPARQL queries via the *Evolyzer* tool. The query shows for each change the

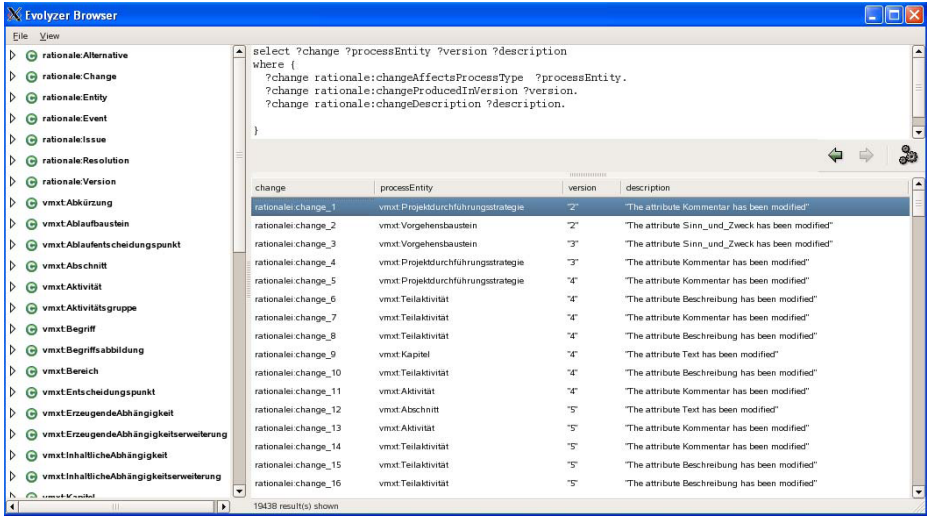


Fig. 7. Querying the Rationale for the V-Modell XT via the *Evolyzer* tool

process entity that affected, the version where this occurred and the description of such a change.

6 Related Work

Several other research efforts are concerned in one way or another with comparing model variants syntactically and providing an adequate representation for the resulting differences.

[1] and [16] deal with the comparison of UML models representing diverse aspects of software systems. These works are oriented towards supporting software development in the context of the Model Driven Architecture. Although their basic comparison algorithms are applicable to our work, they are not concerned with providing analysis or visualization for specific users.

[20] presents an extensive survey of approaches for software merging, many of which involve comparison of program versions. The surveyed works mainly concentrate on automatically merging program variants without introducing inconsistencies, but not, as in our case, on identifying differences for user analysis.

[3] provides an ontology and a set of basic formal definitions related to the comparison of RDF graphs. [32] and [11] describe two systems currently under development that allow for efficiently storing a potentially large number of versions of an RDF model by using a compact representation of the raw changes between them. These works concentrate on space-efficient storage and transmission of change sets, but do not go into depth regarding how to use them to support higher-level tasks (like process improvement).

An extensive base of theoretical work is available from generic graph comparison research (see [13]), an area that is concerned with finding isomorphisms (or correspondences that approach isomorphisms according to some metric) between arbitrary

graphs whose nodes and edges cannot be directly matched by name. This problem is analogous in many ways to the problem that interests us, but applies to a separate range of practical situations. In our case, we analyze the differences (and, of course, the similarities) between graphs whose nodes can be reliably matched in a computationally inexpensive way.

Concerning rationale, Dutoit et al. [8] introduce the term *software engineering rationale*, claiming that this term is more useful for discussing rationale management in software engineering. They emphasize that the software development life cycle contains several activities where important decisions are taken, and where rationale plays an important role. In software engineering, most approaches have contributed to the rationale domain by providing new ideas and mechanisms to reduce the risk associated with rationale capture. Such approaches were conceived having in mind the goal of providing short-term incentives for those stakeholders who create and use the rationale. For example, SCRAM [30], an approach for requirements elicitation, integrates rationale into fictitious scenarios that are presented to users or customers so that they understand the reason for them and provide extra information. It is expected that they can see the use and benefit of rationale. Something similar happens in the inquiry cycle [24], which is an iterative process whose goal is to allow stakeholders and developers to work together towards a comprehensive set of requirements.

Most of the approaches developed for software engineering rationale offer tool support provided as either adaptations or extensions of specific requirements and development tools, e.g., SEURAT [6], Sysiphus [9], DRIMER [23], or the Win-Win Negotiation Tool [34], REMAP [26], and C-ReCS [12].

Little work has been done in other areas apart from design and requirements. One of them is the process modeling area. Here, the need and value have been identified, and a couple of research initiatives have been followed with the goal of generating rationale information from project-specific process models. One approach developed by Dellen et al. [7] is Como-Kit. Como-Kit allows automatically deducing causal dependencies from specified process models. Such dependencies could be used for assessing process model changes. Additionally, Como-Kit provides a mechanism for adding justifications to a change. The Como-Kit system consists of a modeling component and a process engine. Como-Kit was later integrated with the MVP approach [5]. The result of this integration effort was the Minimally Invasive Long-Term Organizational Support platform (MILOS) [31]. Sauer presented a procedure for extracting information from the MILOS project log and for justifying project development decisions [27]. According to Sauer, rationale information could be semi-automatically generated. However, the approach does not capture information about alternatives that were taken into account for a decision.

7 Conclusions and Outlook

Due to factors like model size and metamodel differences, the general problem of identifying and characterizing changes in process models is not trivial. By expressing models in a normalized representation, we are able to characterize interesting changes using a graphical pattern matching language. Graphical patterns provide a well-de-

fined, unambiguous and, arguably, intuitive way to characterize common process model changes, as our examples show.

Our implementation of pattern queries in the *Evolzyer* system demonstrates that our pattern-based change identification technique can be used in practical situations involving very large process models like the V-Modell XT. It is important to stress, however, that the technique requires the process entities in compared models to have stable identifiers that are used consistently across versions. This is normally the case when comparing versions of the same model, but not when comparing models that were created independently from each other.

Using RDF allowed us to connect two different data sets and create an even more comprehensive one that makes it easier process engineers or stakeholders to analyze and understand the evolution of a process. The information that we processed from the V-Modell XT bug tracking system and stored in the process evolution repository as a rationale model allowed us to answer questions that we would otherwise have to guess, such as: Which process elements were affected by a change? Which issue had the largest impact on a process? , Which type of issues demand the highest number of changes? A remaining important research question deals with the visualization of the large amount of information stored in a process evolution repository like the one of the V-Modell XT. We are currently investigating mechanisms that facilitate visualization, e.g., we are trying to identify a set of “most wanted queries” based on the special interests of organizations interested in managing process evolution. Such queries can be deduced from the goals of the organization and reduce the scope of the information to be analyzed.

Acknowledgements. We would like to thank Sonnhild Namingha from Fraunhofer IESE for proofreading this paper, and the members of the V-Modell XT team for providing us with the information needed to perform this work. This work was supported in part by the German Federal Ministry of Education and Research (V-Bench Project, No. 01|SE 11 A).

References

1. Alanen, M., Porres, I.: Difference and Union of Models. In: Stevens, P., Whittle, J., Booch, G. (eds.) UML 2003 - The Unified Modeling Language. Modeling Languages and Applications. LNCS, vol. 2863, pp. 2–17. Springer, Heidelberg (2003)
2. Armbrust, O., Ocampo, A., Soto, M.: Tracing Process Model Evolution: A Semi-Formal Process Modeling Approach. In: Oldevik, Jon. (ed.) u.a.: ECMDA Traceability Workshop (ECMDA-TW) 2005- Proceedings. Trondheim, pp. 57-66 (2005)
3. Berners-Lee, T., Connolly D.: Delta: An Ontology for the Distribution of Differences Between RDF Graphs. MIT Computer Science and Artificial Intelligence Laboratory (CSAIL) (last checked 2006-03-30) Online publication <http://www.w3.org/DesignIssues/Diff>
4. Bohem, B., Egyed, A., Kwan, J., Port, D., Shah, A., Madachy, R.: Using the WinWin Spiral Model: A Case Study. IEEE Computer 31(7), 33–44 (1998)
5. Bröckers, A., Lott, C.M., Rombach, H.D., Verlage, M.: MVP-L Language Report Version 2. Technical Report 265/95, Department of Computer Science, University of Kaiserslautern, Germany (1995)

6. Burge, J., Brown, D.C.: An Integrated Approach for Software Design Checking Using Rationale. In: Gero, J. (ed.) *Design Computing and Cognition '04*, pp. 557–576. Kluwer Academic Publishers, Netherlands (2004)
7. Dellen, B., Kohler, K., Maurer, F.: Integrating Software Process Models and Design Rationales. In: *Proceedings of 11th Knowledge-Based Software Engineering Conference (KBSE '96)*, Syracuse, NY, pp. 84–93 (1996)
8. Dutoit, A., McCall, R., Mistrík, I., Paech, B. (eds.): *Rationale Management in Software Engineering*. Springer, Berlin (2006)
9. Dutoit, A., Paech, B.: Rationale-Based Use Case Specification. *Requirements Engineering Journal*. 7(1), 3–19 (2002)
10. Heidrich, J., Münch, J., Riddle, W.E., Rombach, H.D.: People-oriented Capture, Display, and Use of Process Information. In: Acuña, Silvia T (ed.) u.a.: *New Trends in Software Process Modeling*. Singapore: World Scientific, Series on Software Engineering and Knowledge Engineering, vol. 18, pp. 121–179 (2006)
11. Kiryakov, A., Ognyanov, D.: Tracking Changes in RDF(S) Repositories. In: *Proceedings of the Workshop on Knowledge Transformation for the Semantic Web, KTSW 2002*. Lyon, France (2002)
12. Klein, M.: An Exception Handling Approach to Enhancing Consistency, Completeness, and Correctness in Collaborative Requirements Capture. *Concurrent Engineering Research and Applications* 5(1), 37–46 (1997)
13. Kobler, J., Schöning, U., Toran, J.: *The Graph Isomorphism Problem: Its Structural Complexity*. Birkhäuser (1993)
14. Kunz, W., Rittel, H.: Issues as Elements of Information Systems. Working Paper No. 131, Institut für Grundlagen der Planung, Universität Stuttgart, Germany (1970)
15. Lee, J.: A Qualitative Decision Management System. In: Winston, P.H., Shellard, S. (eds.) *Artificial Intelligence at MIT: Expanding Frontiers*, vol. 1, pp. 104–133. MIT Press, Cambridge, MA (1990)
16. Lin, Y., Zhang, J., Gray, J.: Model Comparison: A Key Challenge for Transformation Testing and Version Control in Model Driven Software Development. In: *OOPSLA Workshop on Best Practices for Model-Driven Software Development*, Vancouver (2004)
17. Lutz, M.: *Programming Python*, 2nd edn. O'Reilly & Associates, Sebastopol, California (2001)
18. MacLean, A., Young, R.M., Belloti, V., Moran, T.: Questions, Options, and Criteria: Elements of Design Space Analysis. *Human-Computer Interaction* 6, 201–250 (1991)
19. Manola, F., Miller, E. (eds.): *RDF Primer*. W3C Recommendation (2004) (last checked 2006-03-22) available from <http://www.w3.org/TR/rdf-primer/>
20. Mens, T.: A State-of-the-Art Survey on Software Merging. *IEEE Transactions on Software Engineering* 28(5) (2002)
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.) *Software Process Change*. LNCS, vol. 3966, pp. 334–334. Springer, Heidelberg (2006)
22. Ocampo, A., Münch, J.: The REMIS Approach for Rationale-driven Process Model Evolution. Submitted to ICSP 2007 (submitted, 2007)
23. Pena-Mora, F., Vadhavkar, S.: Augmenting Design Patterns with Design Rationale. *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing* 11, 93–108 (1996)
24. Potts, C., Bruns, G.: Recording the Reasons for Design Decisions. In: *Proceedings of the 10th International Conference on Software Engineering (ICSE'10)*. Los Alamitos, CA, pp. 418–427 (1988)
25. Prud'hommeaux, E., Seaborne, A. (eds.): *SPARQL Query Language for RDF*. W3C Working Draft (2006) (last checked 2006-10-22) available from <http://www.w3.org/TR/rdf-sparql-query/>

26. Ramesh, B., Dhar, V.: Supporting Systems Development by Capturing Deliberations During Requirements Engineering. *IEEE Transactions on Software Engineering* 18(6), 498–510 (1992)
27. Sauer, T.: Project History and Decision Dependencies. Diploma Thesis. University of Kaiserslautern (2002)
28. Soto, M., Münch, J.: Process Model Difference Analysis for Supporting Process Evolution. In: Richardson, I., Runeson, P., Messnarz, R. (eds.) *EuroSPI 2006*. LNCS, vol. 4257, Springer, Heidelberg (2006)
29. Soto, M., Münch, J.: The DeltaProcess Approach for Analyzing Process Differences and Evolution. Internal report No. 164.06/E, Fraunhofer Institute for Experimental Software Engineering (IESE) Kaiserslautern, Germany (2006)
30. Sutcliffe, A., Ryan, M.: Experience with SCRAM, a Scenario Requirements Analysis Method. In: *Proceedings of the 3rd International Conference on Requirements Engineering*, 1988, Colorado Springs, CO, pp. 164–173 (1998)
31. Verlage, M., Dellen, B., Maurer, F., Münch, J.: A Synthesis of Two Process Support Approaches. In: *Proceedings of the 8th International Conference on Software Engineering and Knowledge Engineering (SEKE'96)*, June 10–12, 1996, Lake Tahoe, Nevada, USA, pp. 59–68 (1996)
32. Völkel, M., Enguix, C.F., Ryszard-Kruk, S., Zhdanova, A.V., Stevens, R., Sure, Y.: *SemVersion - Versioning RDF and Ontologies*. Technical Report, University of Karlsruhe (2005)
33. V-Modell XT (last checked 2006-03-31) Available from <http://www.v-modell.iabg.de/>
34. WinWin. The Win Win Spiral Model. Center for Software Engineering University of Southern California <http://sunset.usc.edu/research/WINWIN/winwinspiral.html>

Use of Non-IT Testers in Software Development

Vineta Arnicane

University of Latvia, Raina blvd. 19, Riga, Latvia
vineta.arnicane@lu.lv

Abstract. Because of a shortage of IT specialists, many companies which are not involved in software development business are forced to use employees who have minimal or no any knowledge about software development and IT as testers (let's call them non-IT testers). The author of this paper has used years of experience in working with such testers to provide a description of them, looking also at their most typical testing styles and the problems which occur for testers, their colleagues and bosses, and the overall software development processes. Non-IT testers often feel like second-class employees, because they are forced to work in an environment in which they do not have sufficient skills. This paper reviews issues which should be taken into account when training these testers, including the question of what exactly they should be taught. Examples and conclusions are used to provide advice on the more effective use of non-IT testers to achieve positive results.

Keywords: Software testing, testing process improvement, tester training.

1 Introduction

In recent years, greater and greater attention in the world of software development has been devoted to testing. At higher education institutions, testing-related training has not attracted sufficient focus, however. Software design companies usually try to deal with the issue of training testers on their own. Various educational programs have been designed by software development companies. Alternatively, training services are offered by specialized organizations or companies.

Companies at which software design is not the basic area of operations (non-IT companies) lack professional testers. Often they lack well-qualified IT specialists as such. Many companies hold on to the myth which says that testing is easy and uncomplicated, that it can be handled by any employee [1]. Even software development companies sometimes use beginners in the field as testers.

Non-IT companies often bring in non-IT testers - people who know nothing about testing and, in some cases, nothing about IT as such or their knowledge is very weak. However often they come from ranks of users and domain experts [2, 3]. Non-IT testers are trying self-educate in testing, but it is very hard issue to them because there are very few sources of information suitable for them. Most of literature requires some IT background for reader.

When testing does not yield the expected results, companies find themselves ready to pay for the training of these individuals in the field of testing.

This article is based on the author's more than 10 years of experience in various jobs in the industry. She has worked with non-IT testers and dealt with their training. This paper reviews the most important observations and conclusions vis-à-vis issues such as things which non-IT testers understand intuitively and without training, the things about testing which they must be taught, the testing methods which they are able to comprehend and apply, and the way in which non-IT testers should be managed.

The author of the paper has conducted several training sessions for non-IT testers, both informally and individually, and by developing training courses and teaching specialized lessons for client organizations (the most detailed course involved 6 working days of study).

Section 2 of this paper characterizes non-IT testers, their advantages and disadvantages. Section 3 shows the testing style of non-IT testers if they are not trained in testing. Section 4 describes some lessons learned about training of non-IT testers. Section 5 deals about some managing aspects of non-IT testers. Section 6 presents results of case studies and Section 7 summarizes conclusions and suggests future work.

2 Non-IT Testers

People who are used as testers by companies can have different levels of knowledge in the area of IT and testing, as well as in the area of the relevant company's business. Companies lack professional testers, and it is hard to find people with good IT skills to work as testers [4] – even if they have little or no knowledge about the software development process as such. This means that companies are forced to turn to people who know about the company's business but have weak or no knowledge about testing or IT as such.

Non-IT people engage in testing on the basis of various circumstances. Sometimes they are given full-time jobs at company IT departments as testers. Other times testing is just a part of a broader set of duties.

The advantage of non-IT people is that they can have in-depth knowledge about the area of business for which IT software is being designed [3]. Such employees often have years of experience in the relevant area, but they have little knowledge about the technologies that are used in system development and the architecture of the resulting system. They don't know much about testing methods and they may have difficulties in preparing of good reports on problems that have been identified.

Deep knowledge about a company's business allows non-IT testers to do good work at the highest level of testing, where functional testing or usability testing must be conducted – acceptance testing and system testing. If the non-IT tester has to do low-level testing such as unit testing or low-level integration testing, however, serious problems may very well occur. It would be equally complicated for that person to make use of specific testing techniques such as performance testing. In regression testing it is hard for them to analyse which areas of the system are functionally linked to software improvements or additions and to test those areas.

3 The Intuitive Testing Style of Non-IT Testers

Before we provided training in each organization we interview non-IT testers, their management, colleagues - programmers and analysts. Our goal is bring to light what problems they have, what they expect from training and as result of it. So we have found out that the most of problems are common in all companies. Let's take a look at them.

If non-IT people are not trained as testers, they do the work intuitively. There are several characteristics which different testing styles have in common. Let us review the most important aspects of the work of non-IT testers:

- **They use functional testing:** Non-IT testers are more likely to engage in ad hoc testing – without any system or consistency, often evaluating a software application just from the visual perspective, testing standard usage, choosing the “right path” with the “right data”, and doing work in the order in which it is usually done;
- **They are afraid of destroying the system:** Perhaps intuition or experience tells a non-IT person that the work of software could be damaged, but the person does not do this so as not to create problems for colleagues and to allow them to continue their work and to finish the development project on time. Potential problems are not examined or analyzed, and the necessary problem reports are not filed;
- **They consider a test case to be only a series of operations, not the relevant data:** Data planning, if any, is too primitive. All input data are sometimes seen as identical, as factors which cannot really change testing results or procedures, so the data which are entered and used during the testing process are not recorded. The results are often evaluated casually – “that’s approximately how things should be”;
- **They intuitively use boundary values and partitioning of data in equivalence classes:** If attention is focused on test data, border instances are intuitively examined. This testing method is used at a fairly normal level;
- **They cannot work with a database:** Testers usually cannot prepare data in a database, read data from tables, determine data which are not shown in user interfaces or LOG data, or they cannot test whether the information on screen or in a report is in line with data in the database;
- **In testing, they use test data which are close to reality, establishing typical situations and chains of activities:** This is a great advantage for non-IT testers if they come from a business with business knowledge. They know HOW and WHERE the system will be used. On the other hand, they find it unnecessary to test non-standard usages of the software, assuming that users know what they are doing;
- **They do not support testing as early as possible:** These people do not see the need for early testing, and they have problems in establishing testing examples on the basis of requirements. It is hard for them to establish and examine a potential system. Non-IT persons see this exclusively as the testing of ready-made software – i.e., they need a program which can be really executed with test cases. They think that systems must be tested and tests must be prepared only when the software is ready, when it can be seen and used in real terms;

- **Sometimes they have stereotypical thinking:** If an organization has already used an IT system that puts a stamp on thinking about the business organization and about how the system should appear and what it should do. It is hard to ensure a fundamental shift in approach, even though this is occasionally needed when new technologies or necessary changes in business processes are implemented;
- **They have problems in noting and describing strange behavior of software:** Non-IT testers do not react to “odd” behavior in the system even if it happens repeatedly, pointing to various secondary factors which might have created the problem. Often they do not react even if they spot something problematic in system’s behaviour, because they think that the situation is as it should be or has been created by a parallel processes. They do not try to ensure a repeat of the questionable situation on purpose;
- **They may not be able to prepare good problem reports:** Usually there are problems in preparing problem reports which go to the heart of the problem, which are clear, precise and brief. Very rarely does anyone check the report to see whether the software designer will be able to find mistakes on the basis of what has been reported;
- **They do not know how to briefly and properly document their work:** Testers not only do not know how to document what they are doing, but they are sometimes afraid to do so, because they do not feel sufficiently competent. Sometimes they do not want to spend time on documentation, because they consider that to be a pointless waste of resources;
- **They too often obey implicitly to IT people– programmers, analysts, their managers [5, 6]:** Non-IT testers feel insecure, they do not defend their judgments in those cases when requirements are incomplete (e.g., when there are no requirements as to usability or performance). If system developers for one reason or another do not want any official registration of a problem, then they insist that the discovered problem is not a mistake and must not be registered so as to improve statistics or to put an official end to a stage in design. This often means that problems are not registered even though they are very important, albeit not readily visible. The result is that fundamental system aspects do not attract sufficient attention, and sometimes they are not tested at all. Non-IT testers put up with more or less useful software that has been developed by the system developers, instead of insisting on their own views and ensuring that problems are recorded in reports or demands for change. Testers wait until real users report a similar problem before reporting on other, identical problems.

Non-IT testers have problems with situation in which the knowledge of a software development or professional tester is needed about issues such as:

1. The way to test a real business situation – long-lasting use of a system, changing of the server time and date to the end of the month or year;
2. The way to select test examples so that the testing is sufficiently effective, using one test example to test various aspects of the system (introducing the synchronization point);
3. The way to fill in a database with test data if the system does not have an appropriate user interface for data entry or if the data come from outer interfaces in the system;

4. The way to prepare larger volumes of testing data;
5. The way to prepare, maintain or commission test beds;
6. The way to test external interfaces.

Non-IT testers are happy to learn about testing procedures if the process is not too complex or long-lasting, but there are problems with training materials. People without a grounding in IT can find it difficult to use existing books or other sources related to testing, because these usually require knowledge about computer sciences and, particularly basic software design skills.

On the other hand, companies wish to train non-IT people to do testing quickly so as to achieve better results. This means a search for people who can adapt training materials to company needs and can provide special training.

4 Training of Non-IT People for Software Testing

When we prepare the training course in testing for non-IT people we have to consider following items to meet their needs:

1. Their duties and responsibilities in software project, collection found in literature [7,8] should be corrected;
2. Their problems stated in interviews and problems we know our experience they might have;
3. Their experience in testing and project domain;
4. Their education background and age, gender.

These are the issues which non-IT people often hope for when it comes time for training in the area of testing:

1. There will be no “technical details” – bytes, bits and symbol codes shouldn’t be mentioned at all;
2. There will be no need to study the software development methodology – that is a problem for the bosses, we will do what we are told to do;
3. The process will be as close as possible to the would-be used working style – “we already do all our work properly”;
4. There will be as many examples as possible from the system and situation that are to be tested so that the method can be learned more easily and put to use;
5. There will be nothing complicated even that seems to be the case at start – no techniques that is hard to be understood, even if they are effective in use;
6. There will be no requirements related to anything which has to do with programming or the technologies of software development.

What can be taught to non-IT people when it comes to testing? What are the skills and areas of knowledge which can be learned comparatively easily? What will allow such people to make use of their strengths – their high level of knowledge about the relevant business?

- **Defining uses on the basis of requirements:** Non-IT testers can make use of their extensive knowledge about their business and its processes and dependencies. Of use here is knowledge in the drafting of models and the use of cause-and-effect diagrams [9, 10] so as to justify and document the testing scenario which has been chosen;

- **Use of boundary values and equivalence classes in the selection of testing data:** Intuitive methods can be improved with additional knowledge, or else testers who are unfamiliar with these methods can simply be given a powerful weapon in terms of coming up with good testing examples;
- **Methods to reduce the number of test examples:** Here we refer to knowledge about screen form elements that are grouped according to dependencies, to the use of orthogonal arrays, decision tables and analysis of combinations;
- **Recording of test examples:** People must learn how to record the test, the entered data and the expected results;
- **Knowledge about usability requirements:** Non-IT people should be taught about desirable placement of elements, the appearance of screen forms and reports, undesirable practices and common mistakes. This knowledge allows testers to have a greater belief in themselves, it allows them to justify their views and to stand up against the pressure of system developers. Sometimes companies have their own guidelines as to the appearance of user interfaces, and then testers can look to see whether the guidelines are of an acceptable level of quality;
- **Risk analysis:** Risk analysis [9, 11] makes it possible to organize work more effectively under conditions of limited time and resources, testing the most important requirements and engaging in the most critical tests. Sometimes testers do not want to do this, arguing that risk evaluation requires too much bureaucracy;
- **Basic skills in SQL queries:** These allow testers to look at data in a database. In more complicated cases, it is better if the query is written by someone from the development group, but testers must know how to use the queries as necessary. Queries of this kind, for instance, allow testers to check the LOG records of the system;
- **The role of the tester in projects and the aims of testing [10, 11, 12]:** Testers have a better sense of their job of finding mistakes than of proving that the system is faultless. They understand their capabilities in terms of those mistakes which they can and cannot define. Testers must understand that they alone cannot ensure the quality of software, and they are not responsible for any shoddiness on the part of the software developers [4];
- **Writing problem reports:** Testers must know how to prepare such reports, and they must be aware of the required content [5]. It is desirable to remind them of simple methods to note problems – how to obtain a snapshot of the screen, how to use graphics editing to cut the necessary part out of an image, how to place the image in the problem report itself, etc.;
- **Real examples:** Training should be based on examples from the system which is to be tested – requirements, screen forms, reports, etc. In that case the training uses concepts and business terms with which testers are familiar. It is easier to absorb and remember new knowledge. Even if there is use of materials related to a part of the system which has been tested and approved for use, people are sometimes surprised at how many problems of that system part have gone unnoticed;
- **Metrics [10, 11, 13]:** How to measure size and coverage of own and team work, how to feel problems in planning and designing of tests;

- **Tester group's place in project team, psychological aspect of testing and team work [4, 5, 6, 12]:** Testers should know advantages and disadvantages of internal organization of software development team. Effect of psychological climate in tester group and in all development team.

5 Specifics in Managing the Work of Non-IT Testers

All testers have many management problems in common, irrespective of how much they know about IT. Management of non-IT testers, however, can be a specific situation, largely as a result of the fact that non-IT testers feel greater humility is dealing with system developers. That is because they feel insecure about their own competence in the field of IT. However some issues are the same as IT testers have [4, 8, 12, 14, 15, 16].

- **Testers must feel that they are authorities and that their work is necessary:** During training, there is often revelation of things which testers do not dare say on an everyday basis in conversation with their bosses – that there is insufficient interest in their work, that they feel abandoned, that the results of their work are ignored, that instructions have been given to define or not to define a problem, etc. No one is interested in the fact that these people know or do not know something specific about their work. Testers don't know where to go for help. Training of testers, therefore, should be offered in the presence of each tester's immediate superior;
- **Managers must monitor the time use of the tester:** Non-IT testers will need additional time to design tests, at least at first time period. If there is insufficient time, then testing will be limited to functionality in relation to standard usage. Eventually such a tester will start to believe that there has been enough testing. When repairs of mistakes are investigated, for instance, only the test which found the problem is carried out, and there is no thought to the possible side-effects to the fix;
- **It is dangerous if non-IT testers are subordinated to the leader of the developer group:** Non-IT testers often yield before the views of IT people without insisting upon their own experience and views about what the system should be like. If the leader of the group is not interested in learning about mistakes for one reason or another, then the tester is told that the mistakes are not mistakes, that no problem report must be written and that the mistakes do not have to be registered. For instance, there is no acceptance of mistakes related to the usability of software – ones that are the result of the poor quality of the system's design and ones that are expensive to fix now. The consequences is that the tester calmly waits until users report the problems, because group leaders usually respect the views of users;
- **System developers and testers must both establish internal problem registers (may be private):** Sometimes project managers can try to reduce the number of problem reports, instructing testers not to report some classes of errors. For instance, they can insist that there be no reports on errors which are difficult to reproduce. They can instruct testers not to report light-weight errors, especially if already it is expensive to correct them. For instance, the testers are kept from

reporting on user interface problems, because “testers don’t know what users need” or “users are satisfied with this interface.”

Shortages of time sometimes force software designers to present a module for testing even though they know that there are mistakes therein.

In such and similar cases both developers and testers make their own internal problem registers, usually in the form of spreadsheets. Testers can use the register to remember those cases when there were suspicions of a mistake, but it was not possible to repeat or to localize it sufficiently to prepare a problem report. If such suspicious cases are kept in mind, it is possible to reveal the mistakes more effectively later, when they have appeared again. For developers such register is like reminder of mistakes that should be caught as soon as possible.

- **A dictionary is necessary:** Software developers and testers from the business side of the operation must speak the same language and use the same contexts in the same sense. There must be agreement on terminology. In practice it has been found that there is great chaos in the use of testing terms.

6 Case Studies

Let us now look at three cases in which non-IT testers became involved in software testing.

6.1 Case I

Our first case for review involves a governmental financial institution with some 500 employees. The organisation had a central office and few branches in various parts of Latvia. The financial institution bought and then adapted software, installing it over the course of two years after the purchase. The first work was done by a group of three people, but eventually that group expanded to 12 people, including seven software analysts and designers, two representatives from business operations who determined the organisation’s specific requirements vis-à-vis the system, and three consultants from the company which developed and sold the software.

The first levels of testing were handled by the development group. Non-IT testers were more involved in the stages of system testing and acceptance testing.

At the very beginning of the project, one or two authorized users were defined in each department of the organization to provide consultations to system developers how the software should be created. In most of cases user representatives were non-IT testers - without any education in IT or testing.

User representatives had two tasks. During development they had access to the testing environment, and they were ordered to test all of the operations which they might use in their work as soon as appropriate parts of software were added to testing environment. It was their part of system testing. The second task was acceptance testing – for few working days they handled as many of their everyday operations as possible in the testing environment, and they also tested the operations which must be handled at the end of a month or a year.

The user representatives were trained to use the system, but they were not trained in the area of testing.

Initially there was user representatives were very passive in use of the system. Statistics about the number of log-ins and the amount of time spent in the system were distributed, and then testers tried to hook up to the system. After reviewing it a bit, they stopped the work. They did not log out of the system, however, hoping to improve statistics as to the amount of time that had been spent in the system.

When statistics were made available about the screen forms used in the system, the number of reports and the time when these were presented, true work in the system began, but not in a planned way. The authorized users tested the functionality when requested to do so by the system developers.

User representatives did the testing in parallel to their everyday work. They were paid extra money for their work during the acceptance testing, when the work was done on weekends and holidays.

The results: The authorized users were passive as non-IT testers until it was proven that their work was being registered. There was no planned or overall testing.

They said that one problem was busyness during everyday work, also speaking of a lack of knowledge about how to do the testing.

Non-IT testers were irreplaceable in complex functional tests when it was necessary to know whether testing results were correct or incorrect.

When problems were found or questions about the system occurred, non-IT testers forwarded these via e-mail to the developers who then checked out the situation and filed a problem report.

Non-IT testers avoided working with situations which sometimes occur on an everyday basis, are permissible, but are atypical – entering specific data, for instance.

The non-IT testers found several functional problems, however, and once the system went on line, they helped colleagues to learn how to use it. This improved the speed and quality with which the new system was learned.

6.2 Case II

The second case relates to an insurance company with some 1,500 employees, a central office, 30 branches, and many small sales facilities all over Latvia. Here, too, software was bought and then adapted by the company's own IT department. The developers conducted unit testing and some of the integration testing. The IT department had an independent testing group which conducted system testing and planned acceptance testing. That was done by users – non-IT testers.

When the software was almost ready, one or two volunteers from each affiliate and department of the company were asked to represent users. They were non-IT testers and were trained on using the system. They received one working day of training on how to conduct the testing.

The non-IT testers had access to the testing environment. First they were allowed to learn about the system on their own, playing around with it and asking the developers for consultations. Then there were planned testing activities. The professional IT testers from IT department prepared use cases which each non-IT tester had to handle. All of the system's requirements were covered in this way. The non-IT testers were encouraged to supplement the test cases or scenarios from their own experience, choosing test data and activities on the basis of their own views.

Some of the assignments had to be carried out on a specific date and time and by all testers at once. This tested the performance of the system to operate close to true capacity – many connections and many demands simultaneously. User training was also used to test the system's performance.

The volunteer non-IT testers and users helped to test the reaction of the software to major usage – something that is complicated and expensive to simulate.

The results: Later the volunteers helped colleagues to learn how to use the system once it was on line. The carefully considered tests made it possible to examine several aspects of the testing simultaneously. Not all of the non-IT testers operated equally well or with equal motivations, however. The more active volunteers and those who posted the best results in terms of defining problems later took regular part in the testing of new versions of the software.

6.3 Case III

The third case also relates to an insurance company with many branch offices. This company decided to design its own software, outsourcing some of the work. The development team included company's software developers, analysts and testers. The testers were subordinate to the manager of the software development group.

It took several years to develop the software, and there was much personnel turnover in the development group. At the end of the process, there were only few software designers or analysts who had been with the group from the very beginning.

Testers were brought in when the software development was almost completed. They were non-IT testers who had worked as full time testers on the company's old software. Some of them had 10 and more years of experience with testing, but they all arrived at the process as users, and they had never been properly trained to conduct the tests.

The software development process involved authorised users from each of the organisation's department too. Their duty was to help the software developers when there were questions, as well as to approve the system requirements – software requirements, prototypes, screen forms, reporting forms, etc. The developers conducted unit testing, while the authorised users conducted acceptance testing. The rest of the testing was handled by no-IT testers.

The testers were free to read the recorded system requirements, which were in slightly formalized language. They drew up testing plans, test designs and scenarios. The testing had to be conducted quickly, because the development process took longer than had been expected, and the deadline for presenting the full system was approaching.

The software had very few resources to make the work of the testers easier – for instance it had not records in the LOG table (file) or the ability to receive intermediate results of calculations. This happened because the software requirements referred only to functional needs, but staff turnover and a lack of time meant that software designers had to observe software requirements and designs very precisely.

There was serious training in the field of testing for non-IT testers– more than six working days in all.

Testers were accustomed to using a database from the actual system in their testing, as opposed to establishing their own testing data. They assume that the

database will contain all necessary incidents. Because of size and data confidentiality, however, the actual database was not made available this time, and testers had serious problems in preparing test data. The system received some key data from external systems, not through the use of screen forms, and there were no tools for entering those data into the database.

The results: Because of a lack of time, developers could not support the testers sufficiently, and the testers mostly focused on the most important aspects of functionality. The testers found it difficult to establish test data without a user interface to enter them. Testers also had problems in testing the external interfaces of the system.

Because of insufficient technical knowledge and capabilities, testers found it difficult to conduct low-level integration testing. Basically the testers conducted functional system testing.

The training allowed testers to learn about the requirements to be added to the system requirements so as to make the testing easier. Training improved the qualifications of the testers, and it enhanced understanding among testers, developers and bosses, but the change in working culture and processes occurred slowly, and there were no rapid improvements in the quality of the testing. Serious work now began to restructure the whole testing process.

6.4 Lessons Learned from Case Studies

Non-IT testers can best be used in combination with professional IT testers, because they supplement one another. Non-IT users can test the system's correspondence to business processes, the use of the software, and the user interfaces – screen forms and reports.

The work of non-IT testers should be supported by software developers. They can help in placing test data in the database, establishing SQL queries, making testware and handling other issues which require specific IT skills.

Non-IT testers can also establish usage scenarios and test data which are close to reality, taking into account boundary value analysis and analysis of the equivalence classes. Developers often cannot do this, because they know less about the nuances of the relevant area of business. They can prepare simple SQL queries in the database to check the data and to study records in LOG tables or files.

Non-IT testers can provide good assessment of the applicability of the system's user interface – are work place comfortable, can all functions be carried out, does the system unnecessarily overburden vision or the memory, and is the system easy to use?

Training of non-IT testers in the area of testing does not always improve their work results significantly. When time is short, they cannot make adequate use of their knowledge, for instance.

7 Conclusions

Companies use non-IT testers for various reasons – there are no professional IT testers, it's hard to find them, they cost a lot of money, they don't have a sufficient understanding of the company's business, etc. On the other hand, non-IT testers know

a lot about the company's business and can learn and make effective use of many things during the testing process.

Non-IT testers can have problems with testing documentation, description of test designs, and justifying the test examples which have been chosen. This is largely because testers fear demonstrating their lack of knowledge or skills, and they do not want anyone to look at their mistakes. The company suffers as a result – testers cannot learn from one another. If a tester departs, colleagues have problems in adapting his or her test system. Even the testers themselves cannot always follow along with the systematic aspects of their testing so that the system can be updated when there are changes in the software.

It is better if non-IT tester training is not generalized. Training course should be adapted to each organization. There are necessary lots of examples, making use of the software that they actually being tested. In this case the training material is better receivable to audience.

Non-IT testers accept theory unwillingly. All material should be clearly related with their current testing activities and project where they are involved. Form the other side they are well-disposed towards information that make clear situation in project and they place in it, for instance, about software development methodology used in their project, about possible ways to organize testers work in development team.

Metrics should be presented very carefully, because the first reaction is that metrics will have direct influence on their salaries. Surprisingly that such reaction was from IT educated managers too. Even they perceived metrics as instrument to measure efficiency of labor, not as tool to measure efficiency of testing process and motivator to improve this process.

Audience often is enthusiastic to get feel of practical skills, for instance, requirements analysis, simple and effective choice of test data, reduction of number of test cases, ways for short recording of test cases.

Bosses should take part in the training, because it often lays bare problems about which the bosses had no idea before. Especially in regard how non-IT testers psychologically feel in interaction with programmers, analysts, users, their manager, what technical problems they have but not dare explain to colleagues or management.

This paper describes our findings at this moment. There should be continued research about problems non-IT testers have to face, which knowledge could help them to deal with them. Training course content and exposition methods have to be improved in order make it more efficient receivable to non-IT testers.

References

1. Black, R.: *Critical Testing Processes*. Addison-Wesley, Boston, San Francisco, New York, Toronto, Montreal, London, Munich, Paris, Madrid, Capetown, Sydney, Tokyo, Singapore, and Mexico City (2004)
2. Raccoon, L.B.S.: Definitions and demographics. *Software Engineering Notes*, vol. 26(1), pp. 82–91. ACM Press, New York (2001)
3. Craig, R.D., Jaskiel, S.P.: *Systematic Software Testing*. Artech House Publishers, Boston, London (2002)

4. Hutcheson, M.L.: *Software Testing Fundamentals—Methods and Metrics*. Wiley Publishing, Inc, Indianapolis, Indiana (2003)
5. Myers, G.J.: *The Art of Software testing*, 2nd edn. John Wiley & Sons, Inc, Hoboken, New Jersey (2004)
6. Cohen, C.F., Birkin, S.J., Garfield, M.J., Webb, H.W.: *Managing Conflict in Software Testing*. *Communications of ACM* 47(1), 76–81 (2004)
7. Watkins, J.: *Testing IT: An Off-the-Shelf Software Testing Process*. Cambridge University Press, Cambridge, United Kingdom (2001)
8. Black, R.: *Managing the testing process: Practical Tools and Techniques for Managing Hardware and Software Testing*. Wiley Publishing, Inc, New York (2002)
9. van Veenendaal, E.: *The Testing Practitioner*. UTN Publishers, Den Bosch (2002)
10. Beizer, B.: *Software Testing Techniques*. 2nd edn. The Coriolis Group, LLC, Scottsdale Arizona (1990)
11. Perry, W.E.: *Effective Methods for Software Testing*, 2nd edn. John Wiley & Sons, Inc, New York, Chichester, Weinheim, Brisbane, Singapore, Toronto (2000)
12. Kaner, C., Bach, J., Pettichord, B.: *Lessons Learned in Software Testing: A Context Driven Approach*. John Wiley & Sons, Inc, New York, Chichester, Weinheim, Brisbane, Singapore, Toronto (2002)
13. Chen, Y., Probert, R.L., Robeson, K.: *Effective test metrics for test strategy evolution*. In: *Proceedings of the 2004 conference of the Centre for Advanced Studies on Collaborative research*, pp. 111–123. IBM Press, Markham, Ontario (2004)
14. Kaner, C., Falk, J., Nquyen, H.Q.: *Testing Computer Software*, 2nd edn. John Wiley & Sons, Inc, New York, Chichester, Weinheim, Brisbane, Singapore, Toronto (1999)
15. Ash, L.: *The Web Testing Companion—The Insider’s Guide to Efficient and Effective Tests*. Wiley Publishing, Inc, Indianapolis, Indiana (2003)
16. Black, R.: *Managing the testing process*. Microsoft Press, Redmond, Washington (1999)

Requirements Management Practices as Patterns for Distributed Product Management

Antti Välimäki¹ and Jukka Käriäinen²

¹ Metso Automation Inc, Tampere, Finland

antti.valimaki@metso.com

² VTT, Oulu, Finland

jukka.kaariainen@vtt.fi

Abstract. System products need to be developed faster in a global development environment. A more efficient user requirements collection and product feature analysis become more important to meet strict time-to-market and quality constraints. The goal of this research is to study and find the best practices to support distributed business requirements management during the early phases of product development. The paper describes the process of mining requirements management organizational patterns. The experiences and improvement ideas of requirements management have been collected from a large company operating in the sector of the process automation industry. The results present issues that were found important when managing requirements in a distributed environment. The results are further generalized in the form of an organizational pattern which makes it easier for other companies to reflect on and to apply the results to their own cases.

Keywords: Requirements management, Product management, Distributed development, Organizational patterns, Practices, Mining of patterns.

1 Introduction

Products are getting more complex with customer-specific features and an increasing amount of people attending to development activities. The development environment is often global without physical boundaries. At the same time, faster time-to-market and better product quality is required as the companies should be more cost-effective in harsh business environments. This creates pressures for product development practices.

Product development is a customer-oriented activity where the accurate selection of product features is essential for successful product development projects. The selection of features is made in the early phases of product development, often referred to as “product management” activity. The role of product management is to work as a coordinator between marketing needs and requests for R&D capabilities in order to develop the products within defined timelines and budget and quality requirements [1]. Product managers are responsible for this activity. They gather product ideas and transform them into product features that concretize the ideas. After

various activities, product managers organize the features into the development projects that are responsible for realizing the features in accordance with the schedules. This process of transforming customer ideas into realizable features and assigning them to practical development projects is essential for effective product development. Therefore, it is important that tools and methods that support the management of product ideas, requirements and features are in active use from the beginning of the development process.

An engineering discipline that is responsible for managing these artifacts is requirements engineering. Requirements engineering is a set of activities that cover discovering, analyzing, documenting and maintaining a set of requirements for a system [2]. There are plenty of methods and tools to support this activity. For example, Parviainen et al. [3] have presented an inventory of existing requirements engineering and management methods. They conclude that method descriptions often lack the information of the methods' suitability to different environments and problem situations, thus making the selection of an applicable method or combination of methods to be used in a particular real-life situation, complicated. There is a need to invest more effort in both industrial application as well as research to increase understanding and deployment of the RE concepts and methods [3]. This has been indicated also in Juristo et al. [4]. They conducted a survey about the state of requirements engineering in European organisations (from a software development viewpoint). They found that, at that time, immaturity still defines current RE practices.

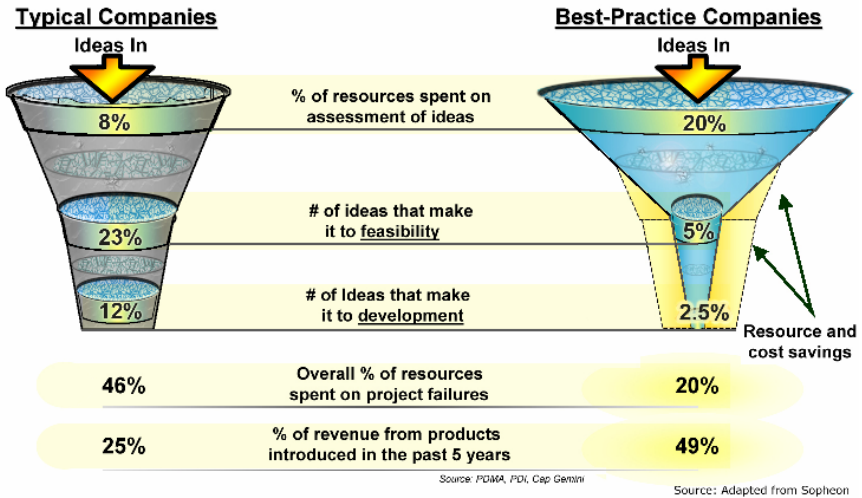
Product management viewpoint has received attention in recent studies e.g. [1], [5] and [6]. Grynberg and Goldin [1] study how efficient requirements management can facilitate the work of product managers in the telecommunication industry. Weerd et al [5] have constructed a reference framework for software product management including a requirements management as one key process area in the framework. Ebert [6] presents a field study from Alcatel where the goal was to study how to reduce project delays. The results showed that efficient requirements engineering is already needed in the early phases of product development.

The best-practice companies manage the innovation process efficiently as seen in the picture below (Fig.1) [7]. The best-practice companies will develop more new products to sell than typical companies by effective idea analysing and feature selection processes.

Nowadays, product development is distributed over multiple sites and customers might also operate globally. Globalization forces companies to find ways to overcome geographical barriers, and modern information technology offers excellent means to achieve this goal. Global development has been in active research, for instance, in [8], [9] and recently published as a special issue in [10]. Requirements management has been studied in this context, for instance, in [11], [12], [13], [14] and [15].

When exploring solutions and practical experiences for product management and its requirements management support, the authors found the knowledge somewhat fragmented that has been indicated also in [5]. Therefore, the goal of the research is to study the challenges and find practices to support distributed business requirements engineering during the early phases of product development (i.e. product management activity) (Fig. 2). Our research question was "How to systematize and improve requirements management methods of product managers in distributed development?"

How do best-practice companies manage innovation?



Source: Deloitte

Fig. 1. How do best-practice companies manage innovation?

The contribution of this paper is twofold. First, it reports issues that were found important for more effective and systematic management of requirements during the early phases of product development. The results were generalized in the form of an organizational pattern [16] (later on referred to as “pattern”), which makes it easier for other companies to reflect on and to apply the results to their own cases. Second, it reports the experiences of pattern mining process. The experiences of requirements

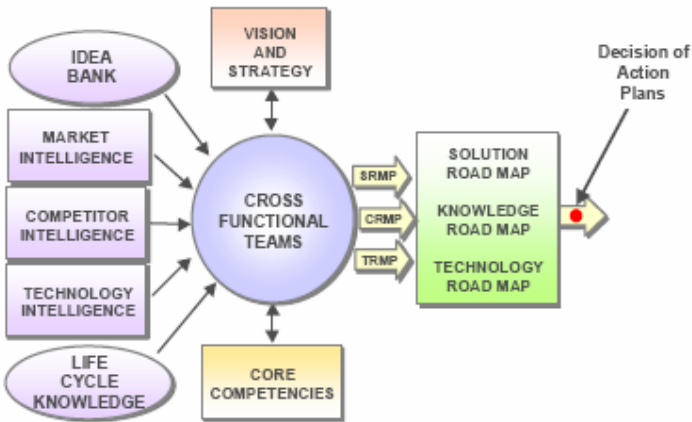


Fig. 2. Fuzzy front end of the innovation process [17]

engineering and suggested solutions have been collected from a large company operating in the sector of the automation industry.

This paper is organized as follows: The next section describes the industrial context of the research. It also discusses the methods and description techniques used in this research. Then the results are presented and discussed. Finally, Section 5 concludes the paper.

2 Research Approach

This section discusses industrial context and methods used in this research. Furthermore, it briefly introduces the concept of patterns.

2.1 Industrial Context

This case study was carried out in a large company operating in the field of the automation industry. The company operates in a multi-site environment and in the future, the work of product managers will globalize even more covering several countries. Therefore, the challenges of the global development environment were studied and suggested solutions to overcome the challenges were defined. Product development in a case company is organized according to product lines. This research focuses on two product lines that consist of several products that are partly developed in different sites. As it is no longer competitive to develop multiple products one at a time, the case company has adopted a product platform approach. Therefore, the product is based on a product platform where the customer-specific features are configured. The product line evolves when new versions of the products are produced containing whole new features or improvements to the existing features. The management of features and requirements is complex since even one product version can contain dozens of features that are further divided into hundreds of requirements.

2.2 Research Method

This research has been carried out in the following phases:

1. Problem definition and research planning
2. Questionnaire-based collection
3. Interviews
4. Analysis of results
5. Creation of patterns.

As described above, there was a two-step approach for data collection. First, an enquiry was made for 12 product managers in two product lines. The outline of the enquiry was based on the company process (The product management process of the case company). The respondents were asked for their opinion about issues that are important for efficient requirements engineering during product management activities. They were also asked for their ideas about what kinds of challenges distributed environment would place on operation.

Based on the enquiry, four experienced product managers were selected as representatives who were interviewed using semi-structured interviews according to the following framework. The framework presented the focus areas and questions for

discussions. The framework was structured based on company process and reference process model (ISO 15504). The framework was further divided into views of questions. The process phases in the framework were:

1. Collecting and processing product ideas
2. Collecting and processing business and customer requirements
3. Feature creation
4. Feature prioritization and selection
5. Adding features to product roadmap
6. From product roadmap to business plan

The views of questions in the framework were:

1. Tools: *Issues that relate to tools*
2. Process: *Issues that relate to working methods, practices*
3. Artifact: *Issues that relate to forms, attributes etc. that are used to collect and organize information about requests, requirements and features*
4. Distribution: *Issues that relate to the geographical distribution of work*

The framework was used as a checklist for an interviewer and therefore it left room for open discussion. Questionnaires and interviews produced raw data for analysis. The process of selecting, simplifying, abstracting and transforming the raw case data is called data reduction [18]. Reduced data was then presented as key observations/challenges (later on referred to as “issues”) in table format to present issues that were found important for more effective and systematic product management in a distributed development environment. Then patterns were constructed by using discussions and workshops in the organization and by applying the pattern creation approach as described in [16].

2.3 Patterns

Patterns were used to present suggested solutions to overcome the challenges in the organization. Patterns seemed to be one good method for describing solutions and guidelines. There are different kinds of forms to describe patterns. For example, the Alexandrian form [19] is used when organizational patterns were described in the book [16]. In the Alexandrian form, the body starts with the statement of the context which includes the problem. The next part is the solution for the described problem. The more complicated organizational pattern form can be found in another piece of research [20] that presents a framework of agile patterns. There are three types of agile patterns which are practice patterns, concepts, and principles. Practice patterns describe actions, e.g. the creation of a functional specification, while concepts describe the attributes of an item, e.g. a functional specification. The third type, principles, consists of guidelines for development activities. Based on the discussion above, the patterns will be presented according to the following format:

- ID:** An ID number of a pattern.
- Name:** A short name of the pattern.
- Problem:** Detailed description of the problems.
- Solution:** Activities that will solve the problem. Activities can include other patterns.
- Guidelines:** These are the guidelines on how to use this pattern.
- Consequences:** The results and trade-offs when the pattern is applied.

3 Analysis of Results

This section presents the results obtained from the enquiry and interviews of product managers. Issues that present needs or challenges for solution are classified according to the framework presented in the previous section. Patterns that describe suggested solutions to overcome the challenges were developed based on these results and literature inventory. In this paper few patterns are presented as examples.

Table 1. Collecting and processing product ideas

Viewpoint	Issues
Tool	<ul style="list-style-type: none"> - Common database for all stakeholders to collect ideas (P1 and P5) - Stakeholder specific forms & views & possibility for pre-defined and user-defined reports from database (P1, P2 and P6) - Ability to define workflow according to company process (process support) (P1) - Ability to create bidirectional links between data objects (P6)
Process	<ul style="list-style-type: none"> - Systematic but lightweight process for collecting, prioritizing and analysing product ideas as well as for making decision about the realization (P1) - Ability to recognize if the idea already occurs in the system (if exists then new idea will be linked to the existing one) (P1) - Ability to assess the benefit and feasibility of the idea (P2) - Ability to collect evaluation information about the idea (e.g. from architects) (P1) - Decision phase should provide possibility for <i>go / no go / postpone / already exists</i> decisions for ideas (P1) - Ability to describe and monitor the status of the idea in different lifecycle phases (P1)
Artifact	<ul style="list-style-type: none"> - Forms should be well-defined containing relevant attributes and multiple-choice attribute's value-ranges should be unambiguous (P1) - Clear definition which attributes are mandatory and which informative - In idea collection phase: e.g. ID, status, idea name, description, person, source, target product, benefit/importance for a customer, cost estimation - In analysing phase: e.g. status, novelty, feasibility, priority, product where planned to be realized, implementation possibilities (who, required schedule), related idea/requirement - In decision phase: e.g. status, planned release/version - Ideas at least in text format but also figures and video clips should be possible to illustrate ideas - Ability to describe traceability between ideas or between ideas and requirements (P6)
Distribution	<ul style="list-style-type: none"> - Secure global access to database regardless of time and place (P1) - Discussion forum in tool. Each response should contain information about author, date, time and discussion topic (P1, P5) - Possibility to use teleconferencing , e-meeting tool, chat, web camera etc. to increase communication efficiency (P5)

Table 2. Collecting and processing business and customer requirements

Viewpoint	Issues
Tool	- Same issues as in previous phase
Process	<ul style="list-style-type: none"> - Product manager receives requirements from different sources as customers, marketing, sales, support, service, competitor analysis, technology providers etc. (P1) - Requirements are added to the system by product manager, other stakeholders or they can already exist in system (e.g. requirements that have been created based on ideas on previous stage) (P1, P2) - Product managers are responsible for analysing to find root causes of the new requirements and making decisions about requirements - Evaluations from experts can be collected for the requirements (e.g. architects) (P1)
Artifact	<ul style="list-style-type: none"> - Tailored forms for different types of requirements (e.g. business, customer, functional requirement) - Generic attributes for customer requirements are e.g. ID, name, priority, status, description, person, source, product, benefits/savings for a customer, estimated costs, estimated incomes - Traceability should be possible between customer requirements as well as between customer requirements and other artifacts (e.g. ideas, features)(P6)
Distribution	- Same issues as in previous phase

Table 3. Feature creation

Viewpoint	Issues
Tool	- Same issues as in previous phase
Process	<ul style="list-style-type: none"> - Product manager analyses requirements and makes feature proposal (P3) - Product manager can collect evaluation information about requirements to support analysis phase (from e.g. architects and project managers) - When feature proposal has been made it will be linked to corresponding requirement(s) (bidirectional traceability) (P6)
Artifact	<ul style="list-style-type: none"> - Feature attributes are e.g. ID, name, description, priority, status, target product, target project, responsible person, estimated costs, estimated income, schedule - Traceability should be possible between features as well as between features and other artifacts (customer requirements) (P6) - Non-functional requirements are used to describe quality requirements as reliability, scalability and response time for functional requirements in textual mode - Functional requirements are used to describe requirements in more detailed level. Often used cases and tasks (project tasks) are related to the requirements. Tasks can be used to specify costs and schedule. Costs are calculated from tasks based on task efforts and schedules based on start/end dates
Distribution	- Same issues as in previous phase

Table 4. Feature prioritization and selection

Viewpoint	Issues
Tool	<ul style="list-style-type: none"> - Ability to define different criteria to make prioritization (P2) - Ability to give higher weight for a certain criterion - Ability to present results in visualized form (P2) - Ability to compare different features to each other by a specific algorithm which makes the comparison process more efficient (P2)
Process	<ul style="list-style-type: none"> - Product manager makes prioritization according to different criteria which can be e.g. value for different customer segments, value for company, risk level (e.g. technology, resource, knowledge etc.), relation to strategy, feature difference between competitors' feature etc. - Product manager is responsible for presenting features and their benefits/costs to stakeholders. Presentation can be done through graphics that help to compare different features (P2)
Artifact	<ul style="list-style-type: none"> - Graphical presentations from features and comparison algorithms for features
Distribution	<ul style="list-style-type: none"> - Trust between different product managers and other managers are important issue to make decision work more efficient - Understanding of different cultures is important in global development environment

Table 5. Adding features to product roadmap

Viewpoint	Issues
Tool	<ul style="list-style-type: none"> - Roadmap contains different views. One important view is Gantt-chart that describes a schedule for the feature realization (P3)
Process	<ul style="list-style-type: none"> - Product manager creates a product roadmap based on the company strategy and e.g. market, competitor and technology intelligences (P3) - Product manager is responsible for presenting features and their benefits/costs to stakeholders. Presentation can be done using graphics that help to compare different features (P2)
Artifact	<ul style="list-style-type: none"> - Different graphical presentations that can be found in tool. - Roadmap templates for Market, Product and Technology Roadmaps (e.g. MS PowerPoint)
Distribution	<ul style="list-style-type: none"> - Same issues as in previous phases

3.1 Results of Enquiry and Interviews

This section presents the results of enquiry and interviews organized according to the framework defined in section 2.2. Results reflect issues that the respondents and interviewees found important for more effective and systematic product management. For each process phase, issues were mapped with a tool, process, artifact or distribution -viewpoint based on the analysis (see Tables 1 to 6). Issues which were used to construct patterns were labeled with a Pattern ID (e.g. P1 = Pattern 1).

Generally, interviews indicated that currently fairly simple, proven solutions are used for the requirements management of early phases of product development. For example, e-mail, text files, presentation templates, etc. In many cases these solutions are sufficient. However, increasing efficiency demands and a shift to a distributed development environment requires that a centralized database is needed for collecting, managing and sharing requests, features and requirements.

Table 6. From Product Roadmap to Business Plan

Viewpoint	Issues
Tool	- A word processor is used to create a Business plan document
Process	- Product manager creates a Business plan (in cooperation with project managers and architects) that will be analysed according to innovation process by head of department and other product managers (P4) - Ensure traceability to detailed technical requirements that will be defined in development project (P4, P6)
Artifact	- Template for Business Plan (e.g. MS Word)
Distribution	- Same issues as in previous phases

3.2 Suggested Solutions - Patterns

The results were analyzed and afterwards, some organizational patterns were created (later on referred to as “pattern”) based on gathered information and related research materials. Some of the important patterns from the viewpoint of distributed development challenges and business requirements engineering are described below.

•**ID:** P1

•**Name:** Established Idea Database

•**Problems:** Often ideas from customers and from employees of the same company are located in product managers’ mails and personal folders. Ideas are difficult to analyze because there are no means to categorize or to manage them.

•**Solution:** When there is a common database for all ideas, they can be found from one store. The ideas can also be categorized by different attributes to make it easier to analyze them and make decisions about further measures. The effective user rights methods and role based views to see certain data make it possible to maintain all requested information security levels. For distributed idea management, it is also important to have a possibility for global access regardless of time and place as well as have a possibility to use a discussion forum inside the tool.

•**Guidelines:** The ideas are gathered from different customers and other sources and they are stored in a common database. The needed attributes of ideas are updated to make it easier to make a decision about the next actions. The ideas have their own lifecycle and different employees with different roles are responsible for managing ideas in different phases. Different measurements are also needed to make it visible to see the status of idea handling

•**Consequences:** The database management needs resources, but the time is saved in finding information more quickly. When all data is in one database, there is a lot of work to ensure that only needed information is visible to certain user groups. The

good idea management process is important because one customer's idea can be the key reason why the customer will buy the product. A good idea is also often a source of more specific customer requirements or needs.

•**ID:** P2

•**Name:** Visualize information of requirements to make prioritization

•**Problems:** A lot of information is needed to understand how valuable and useful a certain requirement is for a customer. It can also be difficult to find out afterwards who was the source of requirements. One big question is also how risky a request is to implement and is there a needed knowledge in use in a company. Prioritization of requests with different customers and different interest groups is also difficult to implement because different groups often have a different prioritization order.

•**Solution:** Customers' needs, requirements and related features need to be stored in one database. There are also needed different attributes to store important information for prioritization e.g. a value for a customer, the amount of development work, a link to strategy etc. Different prioritization views with different criteria are also important for making a decision about which feature will be implemented. Relations between requirements to features which implement the specific requirements are also important to clarify the group of features which are needed and which requirements are related to which features.

•**Guidelines:** Customer's needs and requirements and related features are stored in a database by product managers. The specified attributes are filled to make it easier to make a query for a customer to find out what is the priority order of requirements or features.

•**Consequences:** There is a lot of work to add attribute information to different features. That is why it is important to choose which features are the main features that will be updated with more specific attribute information and traceability links between different artifacts.

Other patterns only by an id and a name are:

•**ID:** P3

•**Name:** Product Roadmap is needed to gather information from different views

•**ID:** P4

•**Name:** Business Plan presents the feasibility of grouped features

•**ID:** P5

•**Name:** Communication tools and the centralized requirements database are needed for efficient distributed development

•**ID:** P6

•**Name:** Traceability based on easy to use bi-directional linking and visualized reports

4 Discussion

The results obtained from this study indicate some important issues for distributed requirements management support for product management. The importance of requirements management in the early phases of product development has also been

reported in other studies in industry, e.g. in [1] and [6]. Related research has indicated that many methods and tools exist to support requirements engineering (e.g. Parviainen et al. [3]). However, Graaf et al. [21] present that there seems to be a relatively large gap between the used and the available methods and tools. Usually fairly simple, pragmatic solutions were used which were based on proven technology. This was also detected in this study. Fairly simple solutions are sufficient when operating in local environments. However, in distributed development environments, a secure shared information repository and electronic connections (e-meetings, teleconferencing, web cameras, chat) were seen as essential solutions to support a collaborative mode of work. This has also been indicated in other case studies related to global product development. For instance, in Battin et al. [8] (e.g. intranet data sharing, teleconferencing). Other major issues that emerged from the needs for more effective requirements management for product managers were:

- need for reports from data objects to support product management activities
- ability to link (bidirectional links) information objects (traceability)
- need for rich data formats (text, video clips, graphics)
- need for a systematic, common process which is tool-supported (workflows)
- clear informative data formats for collecting information about requests, features and requirements
- effective and visual support for prioritization

Grynberg and Goldin [1] have studied how efficient requirements management can facilitate the work of product managers in the telecommunication industry. The article examines what kind of needs product managers have for requirements management. Basically, the same as in our study, there was the need to have a storage place for requests and requirements that allowed linking, organizing, tracing, reporting and sharing information. Lang and Duggan [12] have studied collaborative tool support for requirements management. They derived a core set of requirements for collaborative requirements management tools and assessed how the existing tools comply with these requirements. Many tool related requirements are the same in our study. Lang & Duggan [12] noticed that tools, at that time, did not provide sufficient support for the collaborative mode of work. This sounds challenging since our next step is to improve the existing situation in a case company with systematic process and tool support for product management. Our results also highlight needs for process support (ability for workflows) and visual support for prioritization. Visualization support for prioritization has been studied, for example, in Regnell et al. [15]. They present an industrial case study about a distributed prioritization process. They report that product management found the proposed visualizations (charts) valuable as decision-making support when selecting requirements for the next release. In our study, respondents also highlighted the importance of visualization in the prioritization process and its tool support especially when assessing the benefits and costs of different features.

Patterns seem to be an interesting way to describe solutions for specific problems in the requirement management process. It is possible to emphasize the important activities and consequences of a pattern which will improve the efficiency of a process. For

pattern mining, the research method described in this paper also seems to be a good way to gather information for patterns and their contents. The patterns which have been found in this research are useful but obviously they need to be adapted according to the restrictions and possibilities of each individual development project.

5 Conclusions

An efficient innovation process is very important for best-practice companies. Good ideas and requirements have to be handled and changed to business plans and other change requests in product development faster and more efficiently than earlier, even in a distributed development environment. This paper presents a research process and a framework for mining practices and development needs in order to improve a fuzzy front-end of an innovation process. The research method has included phases in which the results of questionnaires and interviews have been described in the form of a table. Furthermore, the results have been generalized in the form of organizational patterns and a list of important issues.

The important issues from the view of tool, process, artifact and distribution in different phases were studied. The results emphasise the need for a centralized requirements management database which is possible to use around the world. The visualized information is very important in order to improve decision making, especially in the prioritization phase. Additionally, increased organizational know how about products stored in requirements database is valuable for a company.

Organizational patterns seem to be one good method for describing solutions and guidelines to a problem. Generalized patterns make it easier for other companies to reflect on and to apply the results to their own cases. For pattern mining, the use of a developed framework and research process, as presented in this paper provided a good way to gather and organize information for patterns and their contents.

The future research directions will be the analysis of experiences with the current patterns in future development projects, the improvement of the patterns, and the creation of new patterns according to the feedback gained from projects.

Acknowledgements. This research is part of the ITEA project called TWINS (Optimizing HW-SW Co-design flow for software intensive system development). The work is funded by Tekes, Metso Automation and VTT. The authors would like to thank all the respondents and interviewees for their assistance and cooperation. The authors would also like to thank Prof. Pekka Abrahamsson from VTT and Prof. Kai Koskimies from the University of Tampere for their valuable comments and support for this study.

References

1. Grynberg, A., Goldin, L.: Product Management in Telecom Industry – Using Requirements Management Process. In: Proceedings of the IEEE International Conference on Software—Science, Technology & Engineering (SwSTE'03) (2003)
2. Sommerville, I., Sawyer, P.: Requirements Engineering: A Good Practise Guide. John Wiley & Sons, West Sussex (1997)

3. Parviainen, P., Tihinen, M., van Solingen, R., Lormans, M.: Requirements Engineering: Process, Methods and Techniques. In: Silva, A., Mate, J. (eds.) Requirements Engineering for Sociotechnical Systems, published by Idea Group (2005)
4. Juristo, N., Moreno, A.M., Silva, A.: Is the European industry moving toward solving requirements engineering problems? *IEEE Software* 19(6), 70–77 (2002)
5. Weerd, I., Brinkkemper, I., Nieuwenhuis, R., Versendaal, J., Bijlsma, L.: Towards a Reference Framework for Software Product Management. In: 14th IEEE International Requirements Engineering Conference (RE'06) (2006)
6. Ebert, C.: Understanding the product life cycle: four key requirements engineering techniques. *IEEE Software* 23(3), 19–25 (2006)
7. Deloitte presentation using sources PDMA, PCA, Gap Gemini, Sopheon, referred in Savolainen, T. Creating the basis for strategic innovation management (in Finnish), Helsinki University of Technology (Unpublished)
8. Battin, R.D., Crocker, R., Kreidler, J., Subramanian, K.: Leveraging resources in global software development. *IEEE Software* 18(2), 70–77 (2001)
9. Herbsleb, J.D., Grinter, R.E.: Splitting the organisation and integrating the code: Conway's law revisited. In: Proceedings of the 1999 International Conference on Software Engineering, May 16–22 1999, pp. 85–95 (1999)
10. Damian, D., Moitra, D.: Guest Editors' Introduction: Global Software Development: How Far Have We Come? *IEEE Software* 23(5), 17–19 (2006)
11. Damian, D., Zowghi, D.: RE challenges in multi-site software development organizations. *Requirements Engineering* 8(3), 149–160 (2003)
12. Lang, M., Duggan, J.: A Tool to Support Collaborative Software Requirements Management. *Requirements Engineering* 6(3), 161–172 (2001)
13. Hyysalo, J., Parviainen, P., Tihinen, M.: Collaborative embedded systems development: survey of state of the practice. In: 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems, ECBS 2006 (2006)
14. Sinha, V., Sengupta, B., Chandra, S.: Enabling Collaboration in Distributed Requirements Management. *IEEE Software* 23(5), 52–61 (2006)
15. Regnell, B., Höst, M., Natt och Dag, J., Beremark, P., Hjelm, T.: An Industrial Case Study on Distributed Prioritisation in Market-Driven Requirements Engineering for Packaged Software. In: *Requirements Engineering*, vol. 6 (1), pp. 51–62. Springer, London (2001)
16. Coplien, J.O., Harrison, N.B.: *Organizational Patterns of Agile Software Development*, Pearson Prentice Hall (2005)
17. Pyötsiä, J.: Innovation Management in Network Economy. In: The Tenth International Conference on Management of Technology, IAMOT 2001, Lausanne, Switzerland (2001)
18. Miles, M., Huberman, A.: *Qualitative Data Analysis: An Expanded Sourcebook*, 2nd edn. Sage, Thousand Oaks, California (1994)
19. Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I., Angel, S.: *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press, New York (1977)
20. Bozheva, T., Elisa Gallo, M.: Framework of agile patterns. In: Proceedings of European Software Process Improvement and Innovation Conference (EuroSPI) (2005)
21. Graaf, B., Lormans, M., Toetenel, H.: Embedded software engineering: The state of the practice. *IEEE Software* 20(6), 61–69 (2003)

SPI Consulting in a Level 1 Company: An Experience Report

Tomas Schweigert and Michael Philipp

SQS Software Quality Systems AG
Stollwerckstraße 11
51149 Köln
tomas.schweigert@sqqs.de
michael.philipp@sqqs.de

Abstract. It has been demonstrated by several case studies that SPI programs generate a substantial benefit, especially for organizations with immature processes. However, experience shows, that there is little buy in to SPI programs from these organizations. The reason is the step by step approach in ISO 15504 and as well in CMMi (Staged) especially the lack of ability to deliver sufficient data for planning and confirming improvements. In these cases a measurement oriented approach might work better because it increases the probability of senior management commitment by focusing aspects which are in the range of management perception.

1 Introduction

There are several case studies which demonstrate that there is a substantial benefit to be had when Software Process Improvement (SPI) is implemented. The last was published by SEI in 2005 [Gibson 2006]. It is also reported that 70% of SPI initiatives fail [Statz 1997]. The two main reasons for failure are a lack of management commitment or unrealistic expectations about what SPI can deliver. In organizations with immature processes both risks have the same origin: the absence of measurement and valid process data.

1.1 SPI from a Management Perspective

From a management viewpoint most SPI initiatives in organizations with immature processes have a common problem: they might outline the improvement in a technical manner but are not able to produce figures suitable for management confirmation of the improvement. This leads to the management dilemma that the costs of SPI are clear (As there is a defined Budget to be spent) but the benefits are not calculable. The potential sponsor is in the position to believe the SPI approach (which leads to unrealistic expectations) or to refuse it (which leads to a lack of commitment).

Alternatively an improvement project may be forced by the customer side as it was historically with CMM or currently in Automotive SPICE but even then there is a high risk that efforts are reduced to the minimum necessary to fulfill the customer

requests. From a management perspective SPI provides compliance instead of performance improvement [Constant 2005].

1.2 SPI from an Improvement Viewpoint

ISO 15504-4:2004 chapter 6 makes a clear statement that business goals should be reviewed before defining an improvement initiative but only relates to processes. After the processes are selected for improvement, ISO 15504 provides a step by step approach through the capability each process. This can result in management frustration because management has no chance to receive other quantitative information than capability levels.

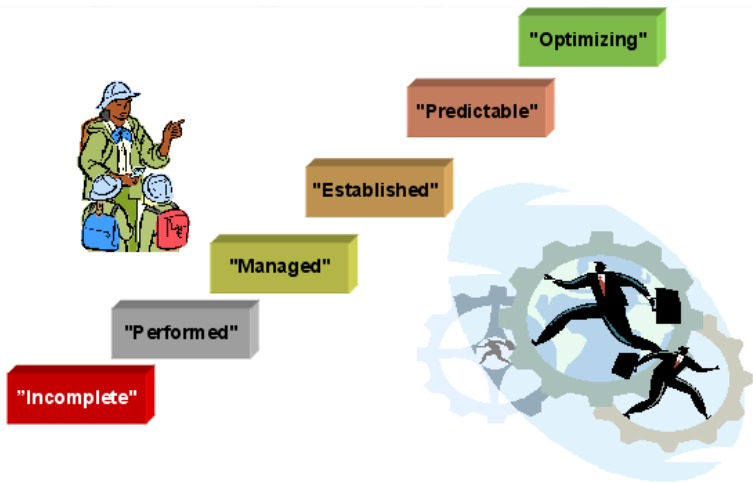


Fig. 1. The step by step approach might fit for technical staff or consultants but does not meet the management information needs

Even if the improvement project is successful in a technical way, meaning that technical people and project leaders accept the improved process and follow these procedures, it is difficult to evaluate the benefits for the business because it is not sure that the required data is available.

A common cause for this weakness is a lack of integration between quality management systems, life cycle models, procedure models and process models. A common symptom is insufficient project planning - isolated improvements for project planning or quality management are not able to fix this problem because they address the symptoms instead of the cause.

2 An Alternative Approach

During an SPI project in a German insurance company the SPI team faced the challenge that the customer was tired of changing processes without any measurable

result. Historically the customer was tired of receiving papers from other firms with recommendations derived from various sources of best practice guides. What the customer really needed was evidence of substantial and provable benefits and not just only recommendations to increase conformance.

Facing this challenge the SPI team decided to develop a recommendation with a different approach. Defining a management cockpit for software projects the team identified the core processes to deliver the needed data. These processes were selected for enhancement. Additionally a set of core metrics were derived to support the building of the management cockpit.

Program/Year	Actual (2006) -> Automated project reporting	MPM (2007)	Roadmap (2008 ff)
Business Area 1	B T Q A 	F V K A 	D S Z A
Business Area 2	B T Q A 	F V K A 	D S Z A
Business Area 3	B T Q A 	F V K A 	D S Z A
Legend	B = Budget T = Date Q = Quality A = Actuality of data	F = Business V = Planned K = Sign Off A = Actuality of data	D = Defined S = Stable Z = Conform A = Actuality of data

Fig. 2. Goal of the management cockpit is to provide information to the management at the needed level

The senior management committed to the approach and decided to spend a budget to implement the recommended improvements.

2.1 The Approach in Detail

The SPI team devised an approach which prioritized the metrics and data to define the necessary improvements.

First the team defined a management cockpit which –after implementation- will support project controlling by senior management.

Then the team defined a set of processes which addressed the technical and business needs of the organization a planned a focused SPICE assessment to get a clear picture of the current status.

After conducting this assessment and presenting the results to and discussing them with several levels of management and technical staff the team defined the data that each process has to deliver for the management cockpit.

The team then defined quality requirements for the data concentrating particularly on the consistency of information.

- From a bottom up viewpoint this means focusing on a few procedure models with standardized measuring points to allocate data, figures and qualitative information as well as looking at the usability of these results for aggregation to ensure support of management purposes.
- From a top down viewpoint this means providing an extended controlling and decision making support by building a consistent set of communication ways throughout all management levels.

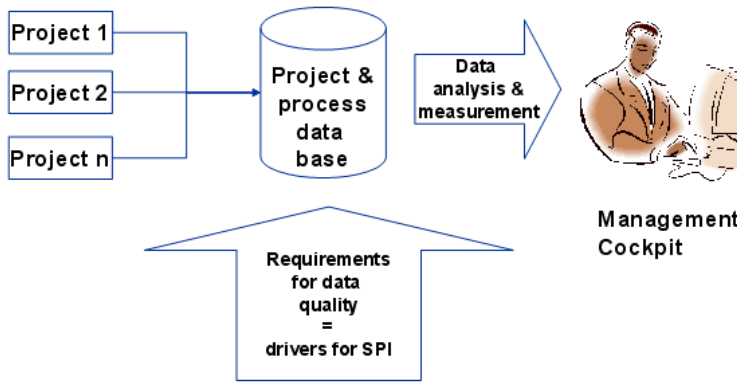


Fig. 3. The measurement driven approach uses management needs as a starting point for SPI activities

Knowing, that this recommendation is an atypical starting point for SPI in an organization with immature processes the SPI team believed in the commitment of the senior management and delivered the recommendations for operational management review.

After this step it was clear which processes were the focus of the SPI initiative - project management, quality assurance, software testing, and quality management.

As a next step the team evaluated the impacts of these requirements on the selected processes. Doing this the team looked at practices and artifacts to be implemented.

Looking at the process “project management” the team came to the conclusion that effort estimation, planning and reporting were starting points to improve the ability to control projects. This has a direct benefit for management as well as providing direct support to the projects. As a consequence the management committed to implement a standard procedure for effort, duration and cost estimation as well as a standard for detailed level project planning and aggregated project reporting. This also promotes a common understanding of artifacts and life cycle models for example, both software engineers and project managers have the same definition of an increment and an iteration.

Looking at the process “quality management”, the team recommended to give high attention to a common understanding of quality throughout the whole IT-organization. This meant having clearly defined interfaces between the processes with quality goals

for the related work products as well as quality gates to verify the quality of work products. In addition, quality campaigns would be started to enhance the quality awareness of the involved staff.

It was agreed that these requirements will lead to significant changes in the behavior of the projects.

It was also agreed, that if this improvement project brought the selected processes to an improved level this would be okay, but it was clearly stated, that the project had no stated goal to reach a certain capability level.

At an organizational level, the SPI team recommended the initiation of the integration of quality management, life cycle model, procedure model and process model to ensure the effectiveness of the measurement model and to improve the communication between management and project level.

All actions were compiled in a roadmap which included the steps necessary to implement the management cockpit.

Comparing the roadmap with a best practice based improvement it has a different structure even when several recommendations are equal.

2.2 Potential Risks

- Management commitment must be sustained throughout the whole improvement project.
- All improvements must be institutionalized and integrated into the technical and organizational infrastructure.
- Roles and responsibilities must be clearly defined including the role of an improvement promoter. This role might be part of the job description for the quality manager.
- Success must be measured and controlled continuously to ensure awareness and learning from incidental failure.
- Measurements must be enhanced in predefined steps. In the initial phase some basic measurements would be established to enable controlling in selected areas of quality effectiveness and efficiency but in later phases measurement will more and more focus on efficiency. This is a key risk in the measurement concept.

2.3 Benefits of the Approach

- Even if the customer has immature processes, and is not willing to trust in expectations based on case studies, it is possible to initiate software process improvement projects.
- The strong focus on management needs helps to create a stable commitment and to ensure adequate funding and time for the project.
- The clearly defined roadmap helps senior management to sell the improvement project to project leaders and technical staff because they have a substantial reason for the intended changes and not only another opinion.
- From the consultant viewpoint this approach is satisfactory because it helps to create a difference in competency and quality to other consulting companies which increases the probability of being selected as the preferred supplier for further business.

3 Summary

Management is not longer willing to fund the implementation of yet another process improvement methodology even where cases studies show the approach to be valid. The benefits to be gained not just for the organization but also for management must be clearly addressed and delivered.

For this reason a step by step approach which is structured by maturity levels will not work even if the implementation of single improvements is supported by isolated measures.

Necessary is a complete measurement oriented approach with a management cockpit on top.

References

- [Gibson 2006] Gibson, D.L., Goldenson, D.R., Kost, K.: Performance Results of CMMI@-Based Process Improvement, Technical Report SEI (2006)
- [Statz 1997] Statz, J., Oxley, D., O'Toole, P.: Identifying and Managing Risks for Software Process Improvement, stsc.hill.af.mil/crosstalk/1997/04
- [Constant 2005] Constant, D.: What's Gone Wrong with CMM/CMMi, Software Quality No 2/2005

On the Effects of Pair Programming on Thoroughness and Fault-Finding Effectiveness of Unit Tests

Lech Madeyski

Institute of Applied Informatics, Wrocław University of Technology,
Wyb.Wyspianskiego 27, 50370 Wrocław, Poland
Lech.Madeyski@pwr.wroc.pl
<http://madeyski.e-informatyka.pl/>

Abstract. Code coverage and mutation score measure how thoroughly tests exercise programs and how effective they are, respectively. The objective is to provide empirical evidence on the impact of pair programming on both, thoroughness and effectiveness of test suites, as pair programming is considered one of the practices that can make testing more rigorous, thorough and effective. A large experiment with MSc students working solo and in pairs was conducted. The subjects were asked to write unit tests using JUnit, and to follow test-driven development approach, as suggested by eXtreme Programming methodology. It appeared that branch coverage, as well as mutation score indicator (the lower bound on mutation score), was not significantly affected by using pair programming, instead of solo programming. However, slight but insignificant positive impact of pair programming on mutations score indicator was noticeable. The results do not support the positive impact of pair programming on testing to make it more effective and thorough. The generalization of the results is limited due to the fact that MSc students participated in the study. It is possible that the benefits of pair programming will exceed the results obtained in this experiment for larger, more complex and longer projects.

1 Introduction

Pair programming (PP) [1] is key software development practice of eXtreme Programming (XP) methodology [2] which has recently gained a lot of attention. Pair programming is a practice in which two distinct roles, called a driver and a navigator, are distinguished. They contribute to a synergy of the individuals in a pair working together at one computer and collaborating on the same development tasks (e.g. design, test, code). The driver is typing at the keyboard and focusing on the details of the production code or tests. The navigator observes the work of the driver, reviews the code, proposes test cases, considers the strategic implications [3,4] and is looking for tactical and strategic defects or alternatives [5]. The rule is that all production code is written by two people

sitting at one machine [2]. In the case of solo programming, all activities are performed by one programmer.

Test-driven development (TDD) [6,2], also known as test-first programming, is another important and well known software development practice of XP methodology, supposed to be used with pair programming. TDD is a practice based on specifying piece of functionality as a test (usually low-level unit test), before writing production code, implementing the functionality, so that the test passes, refactoring (e.g. removing duplication), and iterating the process. The tests are run frequently, while writing production code. Kobayashi et al. [7] suggested that pair programming, test-driven development and refactoring, which is the inherent part of TDD development cycle, had a very good synergy. Therefore, it seems reasonable to evaluate pair programming practice in the context of TDD.

Pair programming is supposed to be software development practice that can influence unit testing to make it more rigorous, thorough, and effective. The question is whether the impact of pair programming is significant or not.

2 Measures

Programmers who write unit tests should have a set of guidelines indicating whether their software has been thoroughly and effectively tested.

2.1 Code Coverage

Measuring code coverage is one of such guidelines which can be applied, as code coverage tools measure how thoroughly tests exercise programs [8]. However, it remains a controversial issue whether code coverage is a good indicator for fault detection capability of test cases [9]. Marick [8] shows that code coverage may be misused, but code coverage tools are still helpful if they are used to enhance thought, and not to replace it. Cai and Lyu [10] found that code coverage was a good estimator for fault detection of exceptional test cases, but a poor one for test cases in normal operations.

Kaner [9] lists 101 coverage measures. The important question is which code coverage measure should be used. Useful insights concerning this question are given by Cornett [11]. Statement coverage, also known as line coverage, reports whether each executable statement is encountered. The main disadvantage of statement coverage is that it is insensitive to some control structures. To avoid this problem, decision coverage, also known as branch coverage, has been devised. Decision coverage is a measure based on whether decision points, such as `if` and `while` statements, evaluate to both true and false during test execution, thus exercising both execution paths. Decision coverage includes statement coverage, since exercising every branch must lead to exercising every statement. However, a shortcoming of this measure is that it ignores branches within boolean expressions which occur due to short-circuit operators. For example, it can preclude calls to some methods. Unfortunately, the most powerful measures as Modified Condition/Decision Coverage (MC/DC), created at Boeing and required for aviation software, or Condition/Decision Coverage are not available

for Java software. Therefore, branch coverage measure was used in our analysis, as the best of available code coverage measures. This measure is offered by several tools e.g. Clover, JCoverage, Cobertura. A detailed analysis revealed that Clover, JCoverage and Cobertura calculate branch coverage in slightly different ways. Therefore, to validate the results obtained by Clover, which has the market leader status, branch coverage results were collected by JCoverage and Cobertura as well. Finally, it appeared that the branch coverage results obtained by JCoverage and Cobertura were in line with the results obtained by Clover, and therefore only Clover results were included in further analysis.

2.2 Mutation Score

A way to measure the effectiveness of test suites is a fault-based technique, called mutation testing, originally proposed by DeMillo et al. [12] and Hamlet [13]. Mutation analysis is a way to measure the quality of the test cases, and the actual testing of the software is a side effect [14]. The effectiveness of test suites for fault localization is estimated on the seeded faults. The faults are introduced into the program by creating a collection of faulty versions, called mutants. These mutants are created from the original program by applying mutation operators which describe syntactic changes to the programming language. The tests are used to execute these mutants with the goal of causing each mutant to produce incorrect output. Mutation score (or mutation adequacy), defined as a ratio of the number of killed mutants to the total number of non-equivalent mutants, is a kind of quantitative measurement of tests quality [15]. The total number of non-equivalent mutants is a difference between total number of mutants and the number of equivalent mutants. Equivalent mutants always produce the same output as the original program, so they cannot be killed. Unfortunately, determining which mutant programs are equivalent to the original program is a very tedious and error-prone activity, so even ignoring equivalent mutants is sometimes suggested [14]. Ignoring equivalent mutants means, we are ready to accept the lower bound on mutation score (named mutation score indicator). Accepting it results in cost-effective application of a mutation analysis and still provides meaningful information about fault-finding effectiveness of test suites.

Empirical studies have supported the effectiveness of mutation testing. Walsh [16] found empirically that mutation testing is more powerful than statement and branch coverage. Frankl et al. [17] and Offutt et al. [18] found that mutation testing was more effective at finding faults than data-flow. Fowler [19] found mutation testing tool support useful in practice.

Although mutation testing is powerful, it is not meant as a replacement for code coverage, only as a complementary approach useful to find code that is executed by running tests, but not actually tested. Moreover, it is time-consuming, and impractical to use without a reliable, fast and automated tool that generates mutants, runs the mutants against a suite of tests, and reports the mutation score of the test suite. Unfortunately, mutation tools for Java, proposed so far, have several limitations that prevent practitioners from using them. They are too slow to be used in large software projects (e.g. Jester [20]), modify source

code of the software components and may break the code making operation risky (e.g. Jester). They do not work with JUnit [21] tests, the most widely used unit testing framework (e.g. MuJava [22,23]), do not support the execution of mutants and are not freely available for download (e.g. JAVAMUT [24]). Therefore, a new mutation tool, called Judy, has been developed, using aspect-oriented approach to speed up mutation testing [25]. Judy has a build-in support of JUnit unit tests, and is under active development to offer a wide range of mutations. Mutations set, used in the experiment, consists of 14 mutations, see Table 1.

Table 1. Judy Mutation Operators

ABS – Absolute value insertion	AOR – Arithmetic operator replacement
LCR – Logical connector replacement	ROR – Relational operator replacement
UOI – Unary operator insertion	UOD – Unary operator deletion
SOR – Shift operator replacement	LOR – Logical operator replacement
COR – Conditional operator replacement	ASR – Assignment operator replacement
EOA – Reference and content assignment replacement	EOC – Reference and content assignment replacement
EAM – Accessor method change	EMM – Modifier method change

The first five operators (ABS, AOR, LCR, ROR, UOI) were taken from Offutt et al.’s research [26] on identifying a set of sufficient mutation operators. The idea of sufficient mutation operators is to minimize the number of mutation operators, whilst getting as much testing strength as possible. Recently, Ammann and Offutt [27] presented these five mutation operators along with UOD, SOR, LOR, COR, ASR as program level mutation operators dedicated to Java language. Ma et al. [28] found that EOA and EOC mutation operators can model object-oriented (OO) faults that are difficult to detect and therefore, can be thought of as good mutation operators for OO programs. Finally, EAM and EMM mutation operators were added, as there is still no determined set of selective mutation operators for class mutation operators. Thus, there is no strong reason to exclude these operators [28].

Branch coverage and mutation score indicator were used as measures to determine thoroughness and fault-finding effectiveness of the test suites.

3 Related Work

Researchers and practitioners have reported numerous, sometimes anecdotal and favourable studies of pair programming. Beck and Andres wrote that a pair is even more than twice as effective as the same two people programming solo [2]. However, empirical evidence concerning pair programming practice effort overhead and speedup ratio often points to, more or less, the opposite, see Table 2. The results of empirical studies suggest that the effort overhead is probably somewhere between 15% and 60%, and speedup ratio is between 20% and over 40%. Another important question concerning pair programming practice is

Table 2. Empirical evidence on pair programming practice effort overhead Approaches: S(Solo), P(Pair), SbS(side-by-side)

Study	Environment	Subjects	Effort overhead and speedup ratio associated with pair programming
[29]	Industry	15(5P/5S)	42% overhead, 29% speed up
[3]	Academic	41(14P/13S)	15%–60% overhead, 20%–42.5% speed up
[30]	Academic	21(5P/5+6S)	60% overhead, 20% speed up
[31]	Academic	25(5P/5SbS/5S)	50% overhead (but only 20% overhead in the case of SbS programming i.e. everyone has their own PC)
[32]	Acad./Ind.	4 case projects (4/5.5/4/4-6)	Neither P nor S had consistently higher productivity.
[5]	Industry	295(98P/99S)	P in general did not reduce the time required to solve the tasks correctly.

whether it improves the quality of software products. Empirical results concerning the impact of pair programming practice on quality of software products are summarized in Table 3. The results of empirical studies suggest that the the positive impact of pair programming on software quality is questionable.

Table 3. Empirical evidence on the impact of pair programming practice on software quality Approaches: S(Solo), P(Pair), T(TDD), C(Classic, test-last)

Study	Environment	Subjects	Impact on software quality
[33]	Academic	37(10P/17S)	P did not produce more reliable code than S whose code was reviewed.
[34,35]	Academic	188 (28CS/28TS/31CP/35TP)	There was no difference in NATP(Number of Acceptance Tests Passed) between S and P. Package dependencies were not significantly affected by P.
[32]	Acad./Ind.	4 case projects (4/5.5/4/4-6)	Lower level of defect density in the case of P was not supported.
[5]	Industry	295(98P/99S)	P in general did not increase the proportion of correct solutions.

To the author’s knowledge, there is no empirical evidence concerning the impact of pair programming on thoroughness and fault-finding effectiveness of unit tests. Therefore, the aim of this paper is to fill in this gap.

4 Experiment Description

The definition, design, as well as operation of the experiment are described in this section.

4.1 Experiment Definition

The following definition determines the foundation for the experiment [36]:

Object of study. The objects of study are software development products (developed code).

Purpose. The purpose is to evaluate the impact of pair programming practice on software development products.

Quality focus. The quality focus is thoroughness and fault-finding effectiveness of unit test suites, measured by code coverage and mutation score indicator, respectively.

Perspective. The perspective is from the researcher's point of view.

Context. The experiment is run using MSc students as subjects involved in the development of finance accounting system in Java.

4.2 Context Selection

The context of the experiment was the Programming in Java course, and hence the experiment was run off-line (not industrial software development) [36]. Java was a programming language and Eclipse was an Integrated Development Environment (IDE). All the subjects had prior experience, at least in C and C++ programming (using object-oriented approach). The course consisted of seven lectures and fifteen laboratory sessions (90 minutes each), and introduced Java programming language, using pair programming and test-driven development as the key XP practices. The subjects' practical skills in programming in Java, using pair programming and test-driven development, were evaluated during the first seven laboratory sessions. The experiment took place during the last eight laboratory sessions. The problem, development of the finance accounting system, was as close to a real one, as it is possible in academic environment. The requirements specification consisted of 27 user stories. The subjects participating in the study were mainly second and third-year (and few fourth and fifth-year) computer science MSc students. MSc programme of Wroclaw University of Technology is a 5-year programme after high school. The experiment was part of a research, conducted at Wroclaw University of Technology, with the aim of obtaining empirical evidence on the impact of pair programming and test-driven development on different aspects of software products and processes [34][35][37]. The experiment analysis was run with subjects involved in 63 projects conducted, using test-driven development approach, by 28 solo programmers (denoted as S) and 35 pairs (denoted as P).

4.3 Variables Selection

The independent variable is the software development method used. The experiment groups used solo (S) or pair programming (P) development method. The dependent (response) variables are mean values of branch coverage (denoted as *BC*) and mutation score indicator (denoted as *MSI*), described in Section 2.

4.4 Hypotheses Formulation

The crucial aspect of the experiment is to get to know and formally state what is intended to evaluate in it. The following null hypotheses are to be tested:

- $H_0_{BC, S/P}$ — There is no difference in the mean value of branch coverage (BC) between solo programmers and pairs (S and P).
- $H_0_{MSI, S/P}$ — There is no difference in the mean value of mutation score indicator (MSI) between solo programmers and pairs (S and P).

4.5 Selection of Subjects

The subjects are chosen based on convenience. They are students taking the Programming in Java course. Prior to the experiment, the students filled in a pre-test questionnaire. The aim of the questionnaire was to get a picture of the students' background. It appeared that the mean value of programming experience in calendar years was 3.7 for solos and 3.9 for pairs. The ability to generalize from this context is further elaborated, when discussing threats to the experiment.

4.6 Design of the Experiment

The design is one factor (the software development method), with two treatments (S and P). The assignment to pair programming teams took into account the subjects' preferences (i.e. they were allowed to suggest partners), as it seemed to be more natural and close to the real world practice. Thus this is a quasi-experiment [38]. In the case of two solo projects questionnaires were not filled in. In the case of one solo project, tests were not written and checked-in properly. These projects were not included in the analysis. The design resulted in an unbalanced design, with 28 solo programmers and 35 pairs.

4.7 Instrumentation and Measurement

The instruments [36] and materials for the experiment were prepared in advance, and consisted of requirements specification (user stories), pre-test and post-test questionnaires, Eclipse project framework, a detailed description of software development approaches (S and P), duties of subjects, and instructions how to use the experiment infrastructure (e.g. CVS version control system). Branch coverage and mutation score indicator values were collected using Clover [39] and Judy [25] tools, respectively.

4.8 Validity Evaluation

When conducting the experiment, there is always a set of threats to the validity of the results. Cook and Campbell [40] defined *statistical conclusion*, *internal*, *construct*, and *external validity* threats. To enable an analysis of the validity of the current study, the possible threats are discussed, based on Wohlin et al. [36].

Threats to the *statistical conclusion* validity are concerned with the issues that affect the ability to draw the correct conclusion about relations between the treatment and the outcome of the experiment, e.g. choice of statistical tests, tools and samples sizes, and care taken in the implementation and measurement of the experiment [36]. Threats to the *statistical conclusion* validity are considered to be under control. Robust statistical techniques, tools (e.g. SPSS) and large sample sizes to increase statistical power are used. Non-parametric tests are used which do not require a certain underlying distribution of the data. Measures and treatment implementation are considered reliable. However, the risk in the treatment implementation is that the experiment was spread across laboratory sessions. To minimize the risk, access to the Concurrent Versions System (CVS) repository was restricted to specific laboratory sessions (access hours and IP addresses). The validity of the experiment is highly dependent on the reliability of the measures. The basic principle is that when one measures a phenomenon twice, the outcome should be the same. The measures used in the experiment are considered reliable, because they can be repeated with the same outcomes.

The *internal* validity of the experiment concerns the question whether the effect is caused by independent variables, or by other factors. Concerning the *internal* validity, the risk of compensatory rivalry, or demoralization of subjects receiving less desirable treatments must be considered. The group using the classical method (i.e. solo programming) may do their very best to show that the old method is competitive. On the other hand, subjects receiving less desirable treatments may perform not so well as they generally do. However, the subjects were informed that the goal of the experiment was to measure different development methods, not the subjects' skills. A possible diffusion or imitation of treatments were under control of the assistant lecturers. The threat of selection was also under control, as the experiment was a mandatory part of the course. It was also checked that mean programming experience (in calendar years) was similar in each group (S and P). Moreover, according to questionnaires, mean programming experience of the subjects who took part in the experiment, and three solo subjects who were excluded from the analysis, were almost the same.

Construct validity concerns the ability to generalize from the experiment result to the concept behind the experiment. Some threats relate to the design of the experiment, and others to social factors [36]. Threats to the *construct* validity are considered not very harmful. The mono-operation bias is a threat, as the experiment was conducted on a single software development project. Using a single type of measure is a mono-method bias threat. To reduce mono-method threats, the post-test questionnaire was added, to enable qualitative validation of the results. It appeared that subjects slightly favoured a pair programming approach. Thus, there seems to be no apparent contradiction between qualitative and quantitative results. Interaction of different treatments is limited due to the fact that the subjects were involved in one study only. Other threats to construct validity are social threats (e.g. hypothesis guessing and experimenter expectancies). As neither the subjects nor the experimenters have any interest in favour of one technique or another, we do not expect it to be a large threat.

As with most empirical studies in software engineering, an important threat is the process conformance represented by the level of conformance of the subjects to the prescribed techniques. Process conformance is a threat to statistical conclusion validity, through the variance in the way the processes are actually carried out, and also to construct validity, through possible discrepancies between the processes as prescribed, and the processes as carried out [41]. The process conformance threat was handled by attempting to keep deviations from occurring, with the help of assistant lecturers. They controlled how development methods were carried out and forced subjects to follow the prescribed techniques. Moreover, the subjects were informed of the importance of following proper development methods.

Threats to *external* validity are the conditions that limit our ability to generalize the results of our experiment to industrial practice. The largest threat is that the subjects were students, who had short experience in pair programming. However, Kitchenham et al. [42] states that students are the next generation of software professionals and thus, are relatively close to the population of interest. Some indications on the similarities between student subjects and professionals are also given by Höst et al. [43]. Moreover, Tichy argues why it is acceptable to use students as subjects [44]. The threads to external validity were reduced by making the experimental environment as realistic as possible (e.g. requirements specification came from an external client).

4.9 Experiment Operation

The experiment was run at Wroclaw University of Technology and consisted of a preparation phase and an execution phase. The preparation phase of the experiment included lectures and training exercises, given directly before the experiment, in order to improve skills and practice in the areas of pair programming, test-driven development, and unit testing using JUnit. Lectures and exercises were given by the author, as well as by assistant lecturers. The goal of this preparation phase was to train student subjects sufficiently well to perform the tasks asked of them. They had to not be overwhelmed by the complexity of, or unfamiliarity with the tasks [44]. Therefore, it took seven laboratory sessions (90 minutes each) to achieve the goal. Then, the subjects were given an introductory presentation of a finance accounting system and were asked to implement it during eight laboratory sessions of the execution phase. Both, the preparation phase and the execution phase, were conducted in classroom settings under continuous supervision of assistant lecturers. The subjects were divided into S and P groups. In the experiment up-to-date development environment composed of Java Development Kit, Eclipse development environment, JUnit testing framework and also CVS repository were used. Additionally, the subjects filled in pre-test and post-test questionnaires, to evaluate their experience and opinions, as well as to enable qualitative validation of the results. The subjects were not aware of the actual hypotheses stated. The data were collected automatically by tools such as Clover and Judy (tool developed at the Wroclaw University of Technology).

5 Analysis of the Experiment

The experiment data are analysed with descriptive analysis and statistical tests.

5.1 Descriptive Statistics

Descriptive statistics of gathered measures are summarized in Table 4. Columns “Mean”, “Std.Deviation”, “Std.Error”, “Max”, “Median” and “Min” state for each measure and development method the mean value, standard deviation, standard error, maximum, median and minimum, respectively.

Table 4. Descriptive statistics for branch coverage (*BC*) and mutation score indicator (*MSI*)

Measure	Development Method	Mean (<i>M</i>)	Std.Deviation (<i>SD</i>)	Std.Error (<i>SE</i>)	Max	Median (<i>Mdn</i>)	Min
Branch Coverage (<i>BC</i>)	S	.38	.22	.042	.90	.39	.00
	P	.39	.21	.036	.83	.32	.09
Mutation Score Indicator (<i>MSI</i>)	S	.39	.22	.042	.72	.43	.04
	P	.47	.29	.049	.98	.44	.09

The first impression is that developers working in pairs (denoted as P), and developers working solo (S) performed similarly. However, it appears that pair programming seems to have some positive impact on mutation score indicator, as there is over 20% increase in the mean value of *MSI* (.47 vs. .39). This difference is supported by differences in minimum and maximum values of *MSI* but not the median. However, it is worthwhile to mention that the mean is resistant to sampling variation, whilst the median is more likely to differ across samples. This is important, as we want to infer something about the entire population. The accuracy of the mean as a model of the data can be assessed by the standard deviation which, unfortunately, is rather large (compared to the mean). The standard deviation, as well as boxplots in Figures 1 and 2 tell us more about the shape of the distribution of the results.

Summarizing descriptive statistics in correct APA (American Psychological Association) format [45], we can conclude that pairs achieved slightly higher mutation score indicator ($M = .47$, $SD = .29$) than solo programmers ($M = .39$, $SD = .22$), whilst branch coverage for pairs ($M = .39$, $SD = .21$) was similar to solo programmers ($M = .38$, $SD = .22$). It is worth noting that mutation analysis required about 30000 mutants to be created for 63 projects. To answer the question whether the impact of pair programming on mutation score indicator and branch coverage is significant, or not, statistical tests must be performed.

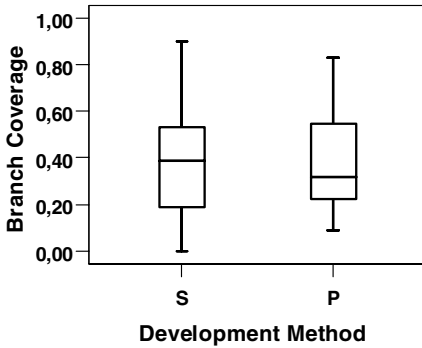


Fig. 1. Branch Coverage Boxplots

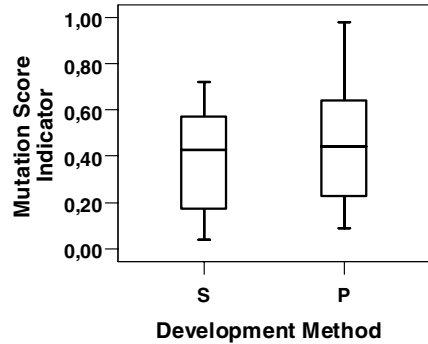


Fig. 2. Mutation Score Indicator Boxplots

5.2 Hypotheses Testing

We start from exploratory analyses on the collected data to check whether they follow the assumptions of the parametric tests (i.e. normal distribution, interval or ratio scale, homogeneity of variance). The first assumption of parametric tests is that our data have come from a population that has normal distribution. Objective tests of the distribution are Kolmogorov-Smirnov and Shapiro-Wilk tests. We find that the data are not normally distributed, see Table 5.

Table 5. Tests of Normality

	Development Method	Kolmogorov-Smirnov ¹ Statistic	df ²	Significance	Shapiro-Wilk Statistic	df ²	Significance
Branch Coverage (<i>BC</i>)	S	.121	28	.200 ³	.964	28	.423
	P	.147	35	.053	.936	35	.043
Mutation Score Indicator (<i>MSI</i>)	S	.113	28	.200 ³	.933	28	.072
	P	.125	35	.179	.922	35	.016

¹ Lilliefors Significance Correction.

² Degrees of freedom.

³ This is a lower bound of the true significance.

For the branch coverage data the distribution for pairs appears to be non-normal ($p < .05$), whereas that for solos is normal according to the Shapiro-Wilk test. It is worth noting that the Shapiro-Wilk test yields exact significance values and is thus more accurate (though less widely used) than the Kolmogorov-Smirnov test. For the mutation score indicator data results are similar. The Shapiro-Wilk test is in fact significant for pairs but not for solos. This finding alerts us to the fact that a non-parametric test should be used. Therefore the

hypotheses from section 4.4 are evaluated using the Mann-Whitney one way analysis of variance by ranks. The Mann-Whitney non-parametric tests are used for testing differences between the two experimental groups (S and P), when different subjects are used in each group.

Table 6. Mann-Whitney Test Statistics (grouping variable: Development Method)

	Branch Coverage (<i>BC</i>)	Mutation Score Indicator (<i>MSI</i>)
Mann-Whitney <i>U</i>	471.500	423.000
Wilcoxon <i>W</i>	877.500	829.000
<i>Z</i>	-.256	-.927
Asymp. Sig. (1-tailed)	.399	.177

Table 6 shows test statistics and significances. It appeared that branch coverage was not significantly affected by pair programming approach (the Mann-Whitney test statistics: $U = 471.5$, non-significant, $z = -.26$). Mutation score indicator was not significantly affected by pair programming approach (the Mann-Whitney test statistics: $U = 423.0$, non-significant, $z = -.93$), either. An effect size ($r = \frac{Z}{\sqrt{N}}$ where Z is the z -score in Table 6, and N is the size of the study i.e. 63) is an objective and standardized measure of the magnitude of observed effect. The effect size is extremely small for branch coverage ($r = -.03$) and a bit higher, but still rather small, for mutation score indicator ($r = -.12$). The later result may suggest the need for further experimentation.

Why did not pair programming result in a significant increase of testing thoroughness or fault-finding effectiveness, measured by branch coverage and mutation score indicator, respectively? The plausible explanation is that when software project is not big enough, and the requirements are decomposed into small features (user stories), the impact of pair programming practice on branch coverage and mutation score indicator may be insignificant, because development skill may, to a certain extent, compensate for the lack of a second pair of eyes.

Another possible explanation is that when the scope of the project is limited, the impact of pair programming practice on branch coverage and mutation score indicator may be insignificant, because of the limited number of tests.

6 Summary and Conclusions

The unique aspect of an experiment conducted at Wroclaw University of Technology was that it included the first ever assessment of the impact of pair programming on thoroughness and fault-finding effectiveness of unit tests. Branch coverage and mutation score indicator were examined to find how thoroughly tests exercise programs, and how effective they are, respectively. It appeared that the pair programming practice used by the subjects, instead of solo programming, did not significantly affect branch coverage ($U = 471.5$, non-significant,

$r = -.03$), or mutation score indicator ($U = 423.0$, non-significant, $r = -.12$). It means that the impact of pair programming on thoroughness and fault-finding effectiveness of unit test suites was not confirmed. The validity of the results must be considered within the context of the limitations discussed in the validity evaluation section. The study can benefit from several improvements before replication is attempted. The most significant one is conducting a larger project, while securing a sample of large enough size to guarantee a high-power design. Further experimentation in other contexts (e.g. in industry, on larger projects) is needed to establish evidence-based recommendations for the impact of pair programming practice on thoroughness and effectiveness of test suites.

Acknowledgements

The author expresses his gratitude to the students and lecturers participating in the experiment, the members of the e-Informatyka team and Norbert Radyk for their help in preparing the measurement infrastructure and collecting data. This work has been financially supported by the Ministry of Science and Higher Education, as a research grant 3 T11C 061 30 (years 2006-2007).

References

1. Williams, L., Kessler, R.: *Pair Programming Illuminated*. Addison-Wesley, London (2002)
2. Beck, K., Andres, C.: *Extreme Programming Explained: Embrace Change*, 2nd edn. Addison-Wesley, London (2004)
3. Williams, L., Kessler, R.R., Cunningham, W., Jeffries, R.: Strengthening the Case for Pair Programming. *IEEE Software* 17(4), 19–25 (2000)
4. Williams, L.A., Kessler, R.R.: All I really need to know about pair programming I learned in kindergarten. *Communications of the ACM* 43(5), 108–114 (2000)
5. Arisholm, E., Gallis, H., Dybå, T., Sjøberg, D.I.K.: Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise. *IEEE Transactions on Software Engineering* 33(2) 65–86 (2007)
6. Beck, K.: *Test Driven Development: By Example*. Addison-Wesley, London (2002)
7. Kobayashi, O., Kawabata, M., Sakai, M., Parkinson, E.: Analysis of the Interaction between Practices for Introducing XP Effectively. In: *ICSE '06: Proceeding of the 28th International Conference on Software Engineering*, New York, NY, USA, ACM Press, pp. 544–550 (2006)
8. Marick, B.: How to Misuse Code Coverage. In: *Proceedings of the 16th International Conference on Testing Computer Software* (1999), <http://www.testing.com/writings/coverage.pdf>
9. Kaner, C.: Software Negligence and Testing Coverage. In: *STAR 96: Proceedings the 5th International Conference, Software Testing, Analysis and Review*. pp. 299–327 (1996)
10. Cai, X., Lyu, M.R.: The Effect of Code Coverage on Fault Detection under Different Testing Profiles. *SIGSOFT Softw. Eng. Notes* 30(4), 1–7 (2005)
11. Cornett, S.: Code Coverage Analysis (Retrieved 2006), <http://www.bullseye.com/coverage.html>

12. DeMillo, R.A., Lipton, R.J., Sayward, F.G.: Hints on Test Data Selection: Help for the Practicing Programmer. *IEEE Computer* 11(4), 34–41 (1978)
13. Hamlet, R.G.: Testing Programs with the Aid of a Compiler. *IEEE Transactions on Software Engineering* 3(4), 279–290 (1977)
14. Offutt, A.J., Untch, R.H.: Mutation 2000: Uniting the Orthogonal. In: Mutation testing for the new century, pp. 34–44. Kluwer Academic Publishers, Norwell, MA, USA (2001)
15. Zhu, H., Hall, P.A.V., May, J.H.R.: Software Unit Test Coverage and Adequacy. *ACM Computing Surveys* 29(4), 366–427 (1997)
16. Walsh, P.J.: A Measure of Test Case Completeness. PhD thesis, Univ. New York (1985)
17. Frankl, P.G., Weiss, S.N., Hu, C.: All-Uses vs Mutation Testing: An Experimental Comparison of Effectiveness. *Journal of Systems and Software* 38(3), 235–253 (1997)
18. Offutt, A.J., Pan, J., Tewary, K., Zhang, T.: An Experimental Evaluation of Data Flow and Mutation Testing. *Software Practice and Experience* 26(2), 165–176 (1996)
19. Venners, B.: Test-Driven Development. A Conversation with Martin Fowler, Part V (Retrieved 2007), <http://www.artima.com/intv/testdrivenP.html>
20. Moore, I.: Jester a JUnit test tester. In: Marchesi, M., Succi, G. (eds.) *XP 2001: Proceedings of the 2nd International Conference on Extreme Programming and Flexible Processes in Software Engineering* pp. 84–87 (2001)
21. Gamma, E., Beck, K.: JUnit Project Home Page (Retrieved 2006), <http://www.junit.org/>
22. Offutt, J., Ma, Y.S., Kwon, Y.R.: An Experimental Mutation System for Java. *SIGSOFT Software Engineering Notes* 29(5), 1–4 (2004)
23. Ma, Y.S., Offutt, J., Kwon, Y.R.: MuJava: A Mutation System for Java. In: *ICSE '06: Proceeding of the 28th International Conference on Software Engineering*, New York, NY, USA, pp. 827–830. ACM Press, New York (2006)
24. Chevalley, P., Thévenod-Fosse, P.: A mutation analysis tool for Java programs. *International Journal on Software Tools for Technology Transfer (STTT)* 5(1), 90–103 (2003)
25. Madeyski, L., Radyk, N.: Judy mutation testing tool project (Retrieved 2007), <http://www.e-informatyka.pl/sens/Wiki.jsp?page=Projects.Judy>
26. Offutt, A.J., Lee, A., Rothermel, G., Untch, R.H., Zapf, C.: An Experimental Determination of Sufficient Mutant Operators. *ACM Transactions on Software Engineering and Methodology* 5(2), 99–118 (1996)
27. Ammann, P., Offutt, J.: *Introduction to Software Testing*. (In progress) (2008)
28. Ma, Y.S., Harrold, M.J., Kwon, Y.R.: Evaluation of Mutation Testing for Object-Oriented Programs. In: *ICSE '06: Proceeding of the 28th International Conference on Software Engineering*, New York, NY, USA, pp. 869–872. ACM Press, New York (2006)
29. Nosek, J.T.: The Case for Collaborative Programming. *Communications of the ACM* 41(3), 105–108 (1998)
30. Nawrocki, J.R., Wojciechowski, A.: Experimental Evaluation of Pair Programming. In: *ESCOM '01: European Software Control and Metrics*, pp. 269–276 (2001)
31. Nawrocki, J.R., Jasiński, M., Olek, L., Lange, B.: Pair Programming vs. Side-by-Side Programming. In: Richardson, I., Abrahamsson, P., Messnarz, R. (eds.) *EuroSPI 2005*. LNCS, vol. 3792, pp. 28–38. Springer, Heidelberg (2005)

32. Hulkko, H., Abrahamsson, P.: A Multiple Case Study on the Impact of Pair Programming on Product Quality. In: Inverardi, P., Jazayeri, M. (eds.) ICSE 2005, pp. 495–504. ACM Press, New York (2006)
33. Müller, M.M.: Are Reviews an Alternative to Pair Programming? *Empirical Software Engineering* 9(4), 335–351 (2004)
34. Madeyski, L.: Preliminary Analysis of the Effects of Pair Programming and Test-Driven Development on the External Code Quality. In: Zieliński, K., Szmuc, T. (eds.) *Software Engineering: Evolution and Emerging Technologies Frontiers in Artificial Intelligence and Applications. Frontiers in Artificial Intelligence and Applications*, vol. 130, pp. 113–123. IOS Press, Amsterdam (2005)
35. Madeyski, L.: The Impact of Pair Programming and Test-Driven Development on Package Dependencies in Object-Oriented Design — An Experiment. In: Münch, J., Vierimaa, M. (eds.) PROFES 2006. LNCS, vol. 4034, pp. 278–289. Springer, Heidelberg (2006)
36. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers, Norwell, MA, USA (2000)
37. Madeyski, L.: Is External Code Quality Correlated with Programming Experience or Feelgood Factor? In: Abrahamsson, P., Marchesi, M., Succi, G. (eds.) XP 2006. LNCS, vol. 4044, pp. 65–74. Springer, Heidelberg (2006)
38. Shadish, W.R., Cook, T.D., Campbell, D.T.: *Experimental and Quasi-Experimental Designs for Generalized Causal Inference*. Houghton Mifflin (2002)
39. Cenqua Pty Ltd: Clover project (Retrieved 2006), <http://www.cenqua.com/clover/>
40. Cook, T.D., Campbell, D.T.: *Quasi-Experimentation: Design and Analysis Issues*. Houghton Mifflin Company (1979)
41. Sørumgård, L.S.: *Verification of Process Conformance in Empirical Studies of Software Development*. PhD thesis, The Norwegian University of Science and Technology (1997)
42. Kitchenham, B., Pfleeger, S.L., Pickard, L., Jones, P., Hoaglin, D.C., Emam, K.E., Rosenberg, J.: Preliminary Guidelines for Empirical Research in Software Engineering. *IEEE Transactions on Software Engineering* 28(8), 721–734 (2002)
43. Höst, M., Regnell, B., Wohlin, C.: Using Students as Subjects — A Comparative Study of Students and Professionals in Lead-Time Impact Assessment. *Empirical Software Engineering* 5(3), 201–214 (2000)
44. Tichy, W.F.: Hints for Reviewing Empirical Work in Software Engineering. *Empirical Software Engineering* 5(4), 309–312 (2000)
45. American Psychological Association: *Publication manual of the American Psychological Association* (2001)

An Agile Toolkit to Support Agent-Oriented and Service-Oriented Computing Mechanisms

Asif Qumer and Brian Henderson-Sellers

Faculty of Information Technology, University of Technology, Sydney
2000 Broadway, NSW, Australia
{asif,brian}@it.uts.edu.au

Abstract. The complex nature of the software development paradigm and the rapid acceptance of emerging abstraction mechanisms, such as agent-oriented and service-oriented computing, highlight the increasing need for re-evaluation of existing software development approaches that focus on agile software development methodologies (primarily originating in object-oriented technology); since existing object-oriented, structure-oriented and component-oriented approaches embodied in an agile approach cannot be applied immediately to agent and service-oriented computing. Therefore, we present here, an agile toolkit (Java-based) to facilitate the construction of multi-abstraction or m-abstraction situation-specific agile processes for software development projects. This paper only presents the newly emergent abstraction concepts of agent and service, and does not discuss the well-established object-oriented mechanism used in current agile approaches.

Keywords: Agile methods, Agent-oriented, Service-Oriented, Method Engineering, M-abstraction.

1 Introduction

Agile software development methods mainly focus on object-oriented software technology [20] and lack the support for other emerging abstraction mechanisms such as agent-oriented and service-oriented computing. According to Luck *et al.* [23], the object-oriented paradigm is not immediately suitable for the development of multiagent systems because it does not address the autonomous behaviour of agents. The concepts of agents and objects are not the same. An agent is an autonomous, interactive, communication-focused and flexible complex entity (existing in some environment) that cooperates with other agents to solve a complex problem [18], [20], [36]. In addition, the concept of service in the agent-oriented context is dissimilar to the concept of services of an object. The emerging concept of service-oriented computing demands the re-evaluation of software development methodologies because the existing object-oriented, component-oriented or structure-oriented analysis and design methods are not immediately applicable for the development of service-oriented applications [13]. Service-oriented computing is a further higher-level abstraction transition from objects and distributed components [12].

There are several agile and agent-oriented methods whilst very few service-oriented application development methods have been reported; and none, to the best of our knowledge, that integrate multiple paradigmatic architectures. Indeed, agent-oriented and service-oriented methods are not yet mature enough to have attained commercial status; and most of the agent and service-oriented methods only focus on the analysis and design phases of the software development life cycle [23], [13]. This paper presents a novel agile tool kit (Java-based) to create, tailor and customize agile agent-oriented or agile service-oriented process fragments to create multiple abstraction paradigm-based (called here “m-abstraction”) agile software development processes by using a method engineering approach [22]. In summary, we are currently developing an agile software solution framework, containing an embedded m-abstraction tool kit, which is being used and tested for the construction of agile processes for agent-oriented and service-oriented applications by using a method engineering approach.

This paper is organized as follows: Section 2 provides an overview of the required abstraction concepts. Section 3 describes agile and related concepts. Section 4 presents the organization of agile toolkit. Section 5 presents the evaluation and application of the agile toolkit with a case study. Finally, Section 6 presents the conclusions.

2 Abstraction: Agent and Service

A software system may be developed or modelled by using a number of abstraction mechanisms such as object, component, agent or service. An abstraction is a logical view of a real world problem or an entity from a specific (software) perspective; such as the representation of real world entities by objects, agents, services, components, features and procedures. These are all the examples of abstraction mechanisms

2.1 Multi-abstraction or M-Abstraction (M-Oriented)

A software development project may combine more than one abstraction mechanism for a specific situation; for example, one project may involve the use of object-oriented and agent-oriented or agent-oriented and service-oriented abstraction together. In order to develop such a project, we need to have a methodology to support both abstractions at the same time. A method that combines practices to support more than one abstraction is called an m-abstraction or m-oriented method, whereas the project is called an m-abstraction or m-oriented project.

2.2 Characteristics of Agent Abstraction

An agent is an autonomous, interactive, communication-focused and flexible complex entity (existing in some environment) that cooperates with other agents to solve a complex software problem [18], [20], [36]. Indeed, the complex nature of agents makes multiagent systems difficult to build in comparison with object-oriented systems. Agents can be categorized as information attitudes and pro-attitudes. Information attitudes are: belief and knowledge. Pro-attitudes are: desire, intention, obligation, commitment and choice [5], [37]. A number of logical frameworks that

combine these agent attitudes have been proposed by different researchers. For example, a popular logical framework that combines belief, desire and intention (BDI) attitudes to characterize agents was proposed by Rao and Georgeff [32].

2.3 Characteristics of Service Abstraction

A software service is a logical view or an abstraction of a business process such as a program or database for carrying out business-level operations [21], [13]. A service is a contractually defined behaviour that is developed and provided by a service provider and is used by other services or service consumers in compliance with a service contract. According to Arsanjani [2], “a service is a software resource (discoverable) with an externalized service description. This service description is available for searching, binding, and invocation by a service consumer. The service provider realizes the service description implementation and also delivers the quality of service requirements to the service consumer. Service should ideally be governed by declarative policies and thus support a dynamically re-configurable architectural style”. A service-oriented architecture (SOA) is a system with a collection of services, interactions and inter-connecting patterns [24]. A service-oriented software system is a platform-independent system that is designed to follow a standard interface and flexible collaboration contract, and can communicate in any mode at any time [34].

2.4 Agent-Oriented Analysis and Design

There are a number of agent-oriented software development methodologies but they mainly focus on the analysis and design phases of the software development life cycle.

Table 1. The key elements of an agent-oriented analysis and design

Agent-Oriented Analysis and Design		
Identification	Specification	Realization
<ul style="list-style-type: none"> • System functionality • Agents, goals/roles • Use case modeling • Agent interactions • Agent flow 	<ul style="list-style-type: none"> • Agent internal • Agent interactions • Multi-agent system environment 	<ul style="list-style-type: none"> • Agent instantiation and deployment

Table 1 presents the key elements of an agent-oriented analysis and design concept that we extracted from existing well-known agent-oriented methodologies such as Tropos [15], Gaia [38], ROADMAP [19], MaSE [9], [36], Prometheus [26] and PASSI [8]. The concepts of agent-oriented analysis and design together with the concepts of agile practices will be used for the construction of agile agent-oriented process fragments or practices.

2.5 Service-Oriented Analysis and Design

There are no standard service-oriented software development methodologies. Table 2 presents the key elements of service-oriented analysis and design that have been

distilled from existing service-oriented analysis and design architectures [2], [12], [34]. The concepts of service-oriented analysis and design together with the concepts of agile practices will be used for the construction of agile service-oriented process fragments or practices.

Table 2. The key elements of a service-oriented analysis and design

Service-Oriented Analysis and Design		
Identification	Specification	Realization
<ul style="list-style-type: none"> • Domain decomposition • Goal-Service modeling • Existing system analysis 	<ul style="list-style-type: none"> • Service specification • Service flow specification • Message and event specification • Component Specification • Component flow specification 	<ul style="list-style-type: none"> • Service allocation to components • Component layer

2.6 Agent Service-Oriented Abstraction

According to agent-oriented and service-oriented analysis concepts (Tables 1 and 2), it is clear that these are two independent abstractions and computational concepts. The autonomous nature and collaborative actions to achieve desired goals are the highlighted features of an agent whereas service does not support autonomy; it is an abstraction of a business process with a service level agreement. Service is accessed via a message-based infrastructure over the network resources. According to Cao *et al.* [6], integration of agent and service is feasible and can be used for the development of agent service-oriented applications. This is one of the motivations for the development of m-orientation concepts. The concepts of m-orientation together with the concepts of agile practices will be used for the construction of agile m-oriented process fragments or practices for m-oriented software development projects.

3 Agile

In our Agile Software Solution Framework, three types of agile practices or agile process fragments have been identified: agile methodology practices, agile governance practices and finally agile knowledge engineering and management practices. Agile process fragments will be used to deliver business value during and after the development of m-oriented projects (Figure 1)

3.1 Agile Practice or Agile Process Fragment Model

We have distilled agile process fragments from various agile methods such as Extreme Programming [4], Dynamic Software Development [10], Feature Driven Development [27], Adaptive Software Development [16], Scrum [33], and the Crystal Family of Methodologies [7], [35]. All the identified and newly created fragments will be stored in an agile fragment repository (agile knowledge-base). Table 3 presents the key elements of an agile process fragment model [31]. Agile process fragments in the agile knowledge-base (see Appendix A) are used for the construction of agile process fragments and processes for various abstractions. The model is used

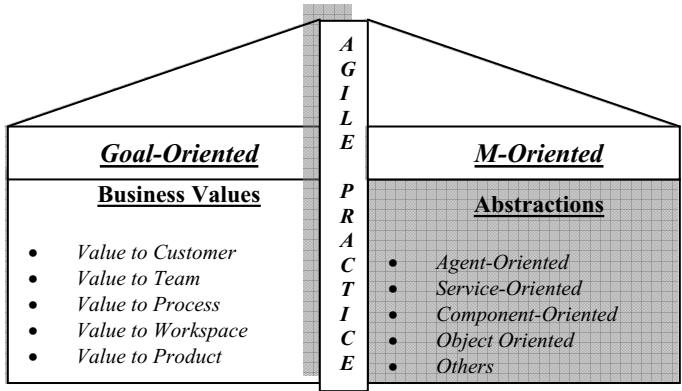


Fig. 1. The two-dimensional view of an agile practice

Table 3. The key elements of an agile process fragment model specifications

Agile Process Fragment	Description
ID & Name	The unique ID and name of the process fragment.
Description & Purpose	The related details and purpose of the process fragment.
Abstraction	Which abstraction mechanism (object, agent, service etc.) does the process fragment support?
Tools & People	Which type of tools and people are required to successfully use the process fragment?
Development Style	Which development style (iterative, rapid) is required to successfully use the process fragment?
Physical Environment	Which physical environment (co-located or distributed) is required by the process fragment?
Pre and Post Conditions	Which pre and post conditions must be true before and after the execution of the process fragment?
Constraints and Risks	What are the possible constraints and risks attached to the process fragment?
Degree of agility	What is the degree of agility (measured in terms of agility attributes) of the process fragment?
Business Value	What business value is added by the process fragment?
Alerts	What are the possible situations when the process fragment should not be applied?

to describe such an agile process fragment is given in Appendix B as a DTD for XML. This helps the developers (directions for self-organizing and empowered teams) to decide on whether to include or exclude a particular agile process fragment in a particular software process for a specific project. In addition, a 4-Dimensional Analytical Tool is embedded to evaluate agile process fragments and the agility measurement mechanism (embedded in 4-DAT) is used to measure the degree of agility of each process fragment [28].

3.2 Agile Business Value

The agile alignment bridge is an issue that has not been investigated and highlighted to any great extent by the agile community. Here, we propose that it should be, because it has an impact on agile governance and on the agile software development approach (both construction and application) in terms of the business value delivered. The Business value to the organization includes: Value to Customer, Value to Team, Value to Process, Value to Workspace, Value to Product

4 Agile Toolkit

Service-oriented architecture (SOA) has been used for the identification and development (together with the case study organisation team) of the essential components or services of the agile toolkit, each service having been implemented and described by using java beans and XML respectively. From this SOA study, we identified five main components (Figure 2) as : (1) agile knowledge-base; (2) agile process fragment and agile process composer, publishers and registry; (3) agility calculator, (4) knowledge-transformer and (5) visualizer. These were selected as providing coverage for i) agility assessment, ii) knowledge (acquisition and storage), iii) process creation and validation and iv) communication. The knowledge-base provides the basic components (agile, abstraction and business value) for the construction of agile process fragments and an agile process. An agile component contains knowledge regarding agility [30], agile values and principles [1] [31] and agile practices (agile software development, agile governance and agile knowledge management). Abstraction contains knowledge regarding various abstraction mechanisms (object, agent, service etc.). The business value component contains knowledge related to possible business values (value to the customer, product, team, workspace etc.) that could be expected from an agile process fragment or an agile process. Agile process fragment represents an individual agile practice (see Appendix A). The agile process contains a collection of agile process fragments (a composition of agile practices by using a software development process meta-model [3], [14], [17]). Composer provides the services for the composition of agile process fragments and agile processes. The agility calculator [28] provided the services for the calculation of degree of agility of a composed agile process fragment or process. Publisher provides the services to transform the composed process fragments and process into an XML [25] format (see Appendix B., for Document Type Definition for XML) and then exports that to the registry. The registry (shared resource) contains agile process fragments and agile processes (with the description of agile process fragments), which are made accessible to developers. The registry exports and imports process fragments to/from the knowledge transformer. The knowledge-transformer transforms the related knowledge (process fragments etc.) to a useable format between the knowledge-base and other agile toolkit components. Visualizer provides an interactive interface to the user of a toolkit.

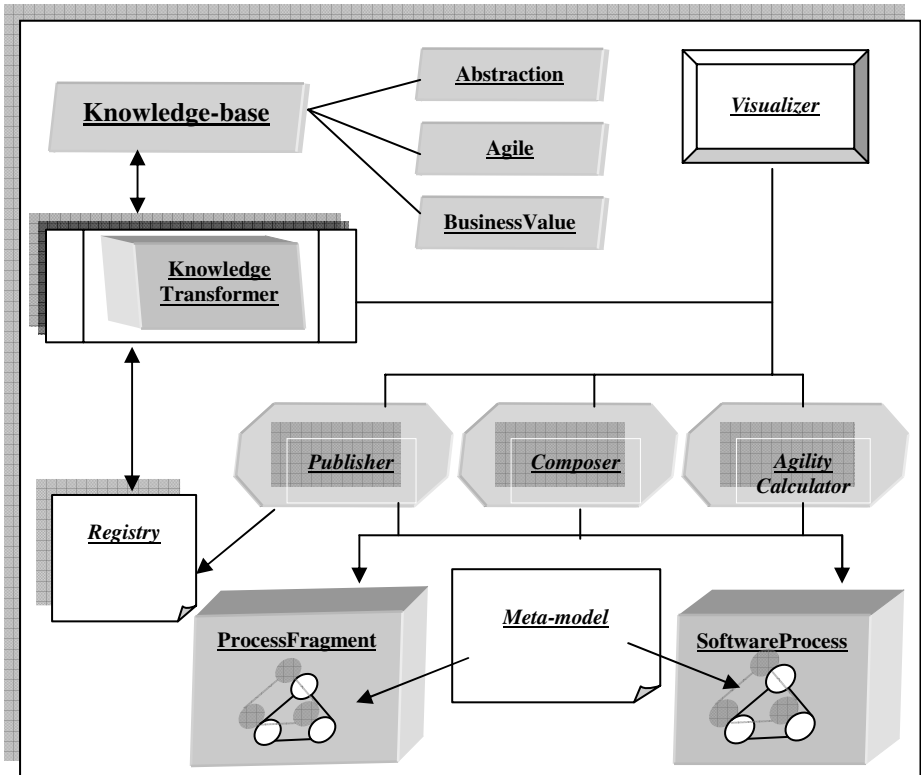


Fig. 2. Components of the agile toolkit

5 Validation and Case Study: Enhanced Pair Programming, Pair Review and On-Site Developer for a Service-Oriented Application

Experiments and tests have been conducted for the assessment and the validation of the agile toolkit. We used this toolkit for the construction of three new agile process fragments: enhanced pair programming (EPP), pair review (PR) and on-site developer for utilization in a service-oriented e-health project case study (empirical testing). These new fragments, as used in the case study, are discussed below.

5.1 The Case Study

The goal of this case study project was to introduce XP [4] into one of the software development organization (our industry partner) for the development of a service-oriented application (e-health).

Table 4. The agile process fragment (Enhanced Pair Programming) specifications

Agile Process Fragment	Description
ID & Name	Enhanced Pair Programming (EPP)
Description & Purpose	A self-organizing pair of two developers will work on more than one independent components of a project but they will exchange the development of the components. The components will be owned by the pair not by the individuals, i.e. pair-ownership for the component.
Abstraction	Service-Oriented (Tested)
Tools & People	Tools should allow for developing and sharing the work. People should be able to communicate and cooperate, self-organize, and have the necessary skills for iterative development with minimal documentation.
Development Style	Iterative and incremental
Physical Environment	Preferably co-located
Pre and Post Conditions	A high-level design and high-level test case design should be available with a high level description of the project components (services). During and after the each iteration of EPP, the design, test cases, requirements and product features will emerge; the product shall be in a stable state with new features or modified features and will be available for testing.
Constraints and Risks	Only two or three co-located people in one set of EPP and one must be senior. Social risks and personality conflict, human resource risks (one person from the EPP may leave or get sick etc. which may affect the development.)
Degree of agility	1.0
Business Value	Reduced Production Cost Reduced Duration Reduced Documentation Improved Product Quality Trained team member Trust but sufficient discipline, control and accountability
Alerts	This approach may not work if all members of the EPP are junior.

In the beginning of this case study, we decided to use only a single XP practice “pair programming” (PP) and analysed it thoroughly before implementation [28], [29], [11], finding several issues that did not allow us to use “pair programming” off the shelf. Therefore, based on this industry experience and evaluation, we decided to customize it by using the agile toolkit to make it useable for the case study e-health development project. We found issues in PP - for example, we cannot hold responsible a single person in a team, the team size grows as the square; there is no option to accommodate an odd number of team members.

The agile toolkit had been used to customize PP by creating three new practices: enhanced pair programming (EPP – see Table 4), pair review (PR – see Table 5) and on-site developer (OSD – see Table 6). In EPP, we decided to use a minimum of two developers (one senior and one junior developer) with an option to add another extra

Table 5. The agile process fragment (Pair Review) specifications

Agile Process Fragment	Description
ID & Name	Pair Review (PR)
Description & Purpose	In the pair review practice, a self-organizing pair of two developers will be used that utilizes the self-testing and exchange-testing techniques for unit testing and integration testing.
Abstraction	Service-Oriented (Tested)
Tools & People	Testing tools should allow for the testing and sharing of the work. People should be able to communicate and cooperate, self-organize, and have the necessary skills for iterative testing with minimal documentation.
Development Style	Iterative and incremental
Physical Environment	Preferably co-located
Pre and Post Conditions	An executable-module with necessary test cases should be available. During and after the each iteration of PR, the design, test cases, requirements and product features will emerge.
Constraints and Risks	Only two or three co-located people in one set of PR and one must be senior. Social risks and personality conflict, human resource risks (one person from the PR may leave or get sick etc. which may affect the development.)
Degree of agility	1.0
Business Value	Reduced Production Cost Reduced Duration Reduced Documentation Improved Product Quality Trained team member Trust but sufficient discipline, control and, accountability
Alerts	This approach may not work if all members of the PR are junior.

developer, if required. EPP developers had to work on individual computers on more than one service (software components) of the e-health project, rather than two on one computer and one component. In the EPP, one programmer should be senior (leader) and with other one or two junior (new to agile or less experience) developers. The senior developer led the development and was responsible for the design and the implementation of the overall service component. The senior developer decided (with the collaboration of other junior developer) which functions of the services would be developed by whom and agreed to help and cooperate with each other for the development of assigned services (software components). Developers in EPP organized themselves by conducting their own small meetings. In-program documentation and face-to-face communication were used to reduce unnecessary documentation and overhead. The senior developer designed the component implementation (code the overall component – skeleton code) and handed it over to the junior for the detailed implementation of one of the specified functions of a service (which they decided mutually). Meanwhile, the senior developer designed the implementation skeleton for the second component; the junior developed a specified

function with the help of a senior and then asked for further directions regarding the development of the component. In this way, by using an exchanged development strategy, they iteratively developed the services (components).

In PR, the developers (two developers with an option to add another extra developer) had to test the services by themselves (self-testing), exchange testing (testing the functions/services of each others), and then finally had to perform the integration testing together. The senior developer led the PR and was responsible for the overall quality of the developed service component. The developers organized themselves for PR by conducting their own small meetings. The developers used a collaborative and communication-oriented (face-to-face) approach to reduce the unnecessary documentation and waste.

The developers found EPP and PR very productive in comparison with a traditional pair programming approach. EPP and PR both helped to improve the quality of the services (components) before user acceptance testing and very few bugs were reported during user acceptance testing. We also trained one junior developer for an agile development environment. The junior developer reported that he was well motivated by the senior colleague and he learned many new programming and testing techniques. It has been observed that the junior developer worked very well with the motivation of the senior as well as being self-motivated. In order to bring sufficient

Table 6. The agile process fragment (On-site Developer) specifications

Agile Process Fragment	Description
ID & Name	On-site Developer (OSD)
Description & Purpose	Developer on the customer site during the start of the project and the prototype development, in order to get quick feedback for clarification of the requirements.
Abstraction	Service-Oriented (Tested)
Tools & People	Tools (portable) should allow rapid requirement gathering, reporting and prototype development. A self-organizing team needs the ability to demonstrate and communicate, and should be able to produce a prototype and work to ensure that the requirements are identified and addressed within the prototype with a minimal documentation.
Development Style	Iterative and incremental
Physical Environment	Off-site (at the customer site)
Pre and Post Conditions	Customer consent and approval for the requirements gathering. During and after the each iteration of OSD, the design, test cases, requirements and product features will emerge.
Constraints and Risks	The workspace at customer site may not be suitable for developers. Customer or customer representative may not be collaborative.
Degree of agility	1.0
Business Value	Reduced Requirements Gathering Cost Reduced Duration Reduced Documentation
Alerts	This approach will not work if the customer does not allow the OSD team to work on their site.

control and discipline to the EPP and PR trusted-team, we embedded the factor of accountability. We allowed and empowered them to take their decisions but they had to justify whatever they decided.

We also developed and used another practice during the case study: the “on-site developer” (developer on the customer site during the start of the project and the prototype development, in order to get the quick feedback for the clarifications of the requirements). In this case study, we managed to develop and test these three new practices and added to the customized version of XP. The results of these practices in terms of their business value are: reduced production cost, reduced duration, reduced documentation, improved product quality, a trained team member and a disciplined team with responsibility and accountability. Tables 4-6 describe the practices based on the process fragment model (Table 3). Initially, the old pair programming practice was assessed [29] and the degree of agility was recorded as 0.60 (the range of value of the degree of agility is [0.0 (Min.) – 1.0 (Max.)]). We also assessed the newly developed practices EPP, PR and OSD and the degree of agility has been calculated by using 4-DAT [28], which is 1.0.

7 Discussion

This paper presents the agile toolkit, which is a part of our newly developed agile software solution framework. This toolkit facilitates the construction, modification or tailoring of situation-specific agile m-oriented (m-abstraction) process fragments and then supports the combination of those fragments into agile software processes (m-abstraction: agent-oriented, aspect-oriented, service-oriented etc.) by using a method engineering approach, feedback and a standard meta-model. The agile toolkit is also a part of our agile workspace in which developers can truly create their own personal software process fragments or processes in an agile manner. The agile toolkit has been tested in one of our pilot projects in industry. The current design of the toolkit has been implemented to fulfill the specific needs of the case study organisation; therefore the implementation cannot be generalised, although the toolkit design could be considered as generic and different organisations may implement this design according to their own specific needs. The results of this case study clearly show that the agile toolkit can be considered productive and can be used by developers or process engineers for the tailoring or construction of situation-specific agile processes for various abstractions. During the case study, it has been found that currently the desktop-based visualizer service could be more effective with a web-based visualizer for an intranet; therefore, the case study organisation has the intention to develop the web-based interface of the agile toolkit in the next increment. We also intend to develop more features for the agile toolkit and undertake further empirical testing ourselves.

Acknowledgment. We wish to thank the Australian Research Council for financial support under the Linkage Grants Scheme. This is contribution number 07/03 of the Centre for Object Technology Applications and Research. We are also grateful to the people from both the research community and the software industry who helped us by providing their valuable feedback and experience in the construction of the agile toolkit.

References

1. Agile Manifesto: Manifesto for Agile Software Development (2006), <http://www.agilemanifesto.org/>
2. Arsanjani, A.: Service-oriented modeling and architecture (2004), <http://www-128.ibm.com/>
3. Australian Standards: Standard metamodel for software development methodologies. AS 4651-2004 (2004), www.standards.com.au/
4. Beck, K.: *Extreme Programming Explained*. Addison-Wesley Pearson Education, Boston (2000)
5. Broersen, J., Dastani, M., Hulstijn, J., Huang, Z., Torre, L.v.d.: The BOID Architecture: Conflicts Between Beliefs, Obligations, Intentions and Desires. In: *Proceedings of the fifth international conference on Autonomous agents*, pp. 9–16. ACM Press, New York, USA (2001)
6. Cao, L., Zhang, C., Ni, J.: Agent Service-Oriented Architectural Design of Open Complex Agent Systems. In: *Proceedings of the 2005 IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, IEEE (2005)
7. Cockburn, A.: *Agile Software Development*. Addison-Wesley, Boston (2002)
8. Cossentino, M.: From Requirements to Code with the PASSI Methodology. In: Henderson-Sellers, B., Giorgini, P. (ed.) *Idea Group Inc, Hershey, PA, USA* (2005)
9. DeLoach, S.A.: Multiagent Systems Engineering: A Methodology and Language for Designing Agent Systems. In: *Proceedings of Agent-Oriented Information Systems* (1999)
10. DSDM: DSDM Consortium, Dynamic Systems Development Method Ltd (2003)
11. Elssamadisy, A., Schalliol, G.: Recognizing and Responding to Bad Smells in Extreme Programming. In: *Proc. ICSE'02, ACM, Orlando, Florida, USA* (2002)
12. Erl, T.: *Service-Oriented Architecture: Concepts, Technology, and Design*. Pearson Education Inc, Upper Saddle River (2005)
13. Feuerlicht, G.: System Development Life-Cycle Support for Service-Oriented Applications. In: *New Trend in Software Methodologies, Tools and Techniques (SoMeT2006)*, IOS Press, Quebec, Canada (2006)
14. Firesmith, D.G., Henderson-Sellers, B.: *The OPEN Process Framework*. Pearson Education, London (2002)
15. Giunchiglia, F., Mylopoulos, J., Perini, A.: The Tropos Software Development Methodology: Processes, Models and Diagrams. In: *Proceedings of the First International Joint Conference on Autonomous Agents and Multi Agent Systems, Bologna, Italy*, ACM Press, New York (2002)
16. Highsmith, J.A.I.: *Adaptive Software Development: A Collaborative Approach To Managing Complex Systems*. Dorset House Publishing, New York (2000)
17. ISO/IEC: *Software Engineering - Metamodel for Development Methodologies*. ISO/IEC Standard 24744 (2007)
18. Jennings, N.R., Sycara, K., Wooldridge, M.J.: A Roadmap of Agent Research and Development. *Autonomous Agents and Multi-Agent Systems* 1, 7–38 (1998)
19. Juan, T., Pearce, A., Sterling, L.: ROADMAP: extending the Gaia methodology for complex open systems. In: *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1, Bologna, Italy*, ACM Press, New York, NY, USA (2002)
20. Knublauch, H.: Extreme programming of multi-agent systems. In: *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 2*, pp. 704–711. ACM Press, NY, USA, Bologna, Italy (2002)

21. Krogdahl, P., Luef, G., Steindl, C.: Service-oriented agility: Methods for successful Service-Oriented Architecture (SOA) development, Part 1. IBM (2005), <http://www-128.ibm.com/>
22. Kumar, K., Welke, R.J.: Method Engineering: A Proposal for Situation-specific Methodology Construction. In: Systems Analysis and Design: A Research Agenda, John Wiley and Sons, New York (1992)
23. Luck, M., Ashri, R., d'Inverno, M.: Agent-Based Software Development. Artech House, Inc, London (2004)
24. Nickull, D.: Service-Oriented Architecture. Adobe Systems, Inc, San Jose, CA (2005)
25. O'Reilly: XML. (2006), <http://www.xml.com/>
26. Padgham, L., Winikoff, M.: Developing Intelligent Agent Systems. John Wiley & Sons, New York (2004)
27. Palmer, S.R., Felsing, J.M.: A Practical Guide to Feature-Driven Development. Prentice-Hall Inc, Upper Saddle River (2002)
28. Qumer, A., Henderson-Sellers, B.: Measuring agility and adoptability of agile methods: A 4-Dimensional Analytical Tool. In: Procs. IADIS International Conference Applied Computing 2006 Guimarães, N., Isaias, P., Goikoetxea, A. (eds.) IADIS Press pp. 503–507 (2006a)
29. Qumer, A., Henderson-Sellers, B.: Comparative evaluation of XP and Scrum using the 4D Analytical Tool (4-DAT). In: Irani, Z., Sarikas, O.D., Llopis, J., Gonzalez, R., Gasco, J. (eds.) Proceedings of the European and Mediterranean Conference on Information Systems 2006 (EMCIS2006) CD, Brunel University, West London (2006b)
30. Qumer, A., Henderson-Sellers, B.: Crystallization of Agility: back to basics. In: Proceedings of the International Conference on Software and Data Technologies (ICSOFT2006), Portugal, INSTICC Press, vol 2, pp. 121–126 (2006c)
31. Qumer, A., Henderson-Sellers, B.: A Framework to Support Non-Fragile Agile Agent-Oriented Software Development. In: Fujita, H., Mejri, M. (eds.) Proceedings of the International Conference on new Software Methodologies, Tools and Techniques (SoMeT2006), Quebec, Canada, pp. 84–100. IOS Press, Amsterdam (2006d)
32. Rao, A.S., Georgeff, M.P.: An Abstract Architecture for Rational Agents. In: Proceedings of the Knowledge Representation and Reasoning, pp. 439–449 (1992)
33. Schwaber, K., Beedle, M.: Agile Software Development with SCRUM. Prentice-Hall, Englewood Cliffs (2002)
34. Tsai, W.T., Malek, M., Chen, Y., Bastani, F.: Perspectives on Service-Oriented Computing and Service-Oriented System Engineering. In: Proceedings of the Second IEEE International Symposium on Service-Oriented System Engineering (SOSE'06), IEEE, NJ (2006)
35. Williams, L., Cockburn, A.: Agile Software Development: It's about Feedback and Change. Computer, vol. 36 (2003)
36. Wood, M.F., DeLoach, S.A.: An overview of the multiagent systems engineering methodology. In: Ciancarini, P., Wooldridge, M.J. (eds.) AOSE 2000. LNCS, vol. 1957, Springer, New York (2000)
37. Wooldridge, M., Jennings, N.R.: Intelligent Agents: Theory and Practice. Knowledge Engineering Review 10, 115–152 (1995)
38. Wooldridge, M., Jennings, N.R., Kinny, D.: The Gaia Methodology for Agent-Oriented Analysis and Design. Autonomous Agents and Multi-Agent Systems, pp. 285–312 (2000)

Appendix A: Agile Practices, Abstractions and Business Value

Table A1. Agile process fragments (practices), Abstractions, Business Value

Agile Practices	<u><i>XP</i></u>	<u><i>FDD</i></u>	<u><i>ASD</i></u>
	<ul style="list-style-type: none"> • The Planning Game • Short Release • Metaphor • Simple Design • Testing • Refactoring • Pair Programming • Enhanced Pair Programming • Pair Review • Collective Ownership • Continuous Integration • 40-Hour Week • On-Site Customer • On-Site Developer • Coding Standards 	<ul style="list-style-type: none"> • Domain Object Modeling • Developing by Feature • Feature Teams • Class Ownership • Inspection • Regular Builds • Configuration Management • Reporting/Visibility of Results 	<ul style="list-style-type: none"> • The Project Mission Development • Adaptive Cycle Planning • Developing By Components • Adaptive Management Model • Collaborative Teams • Joint Application Development by Independent Agents • Customer Focus Group Reviews • Software Inspection • Project Postmortem
	<p style="text-align: center;"><u><i>SCRUM</i></u></p> <ul style="list-style-type: none"> • Product Backlog • Daily Scrum Meeting • Sprint Planning Meeting • Sprint • Sprint Review 	<p style="text-align: center;"><u><i>DSDM</i></u></p> <ul style="list-style-type: none"> • Active User Involvement • Empowered Teams • Iterative and Incremental Development • Frequent Product Delivery • Reversible Changes • Requirements are Baselines at High-level • Integrated Testing • Collaborative and Cooperation Culture 	<p style="text-align: center;"><u><i>CRYSTAL</i></u></p> <ul style="list-style-type: none"> • Staging • Reflection Workshops • Progress Monitoring • Methodology Tuning • Holistic Diversity and Strategy • Parallelism and Flux • User Viewings • Revision and Review
Agent-Oriented	<ul style="list-style-type: none"> • Identification of agents, agent roles/goals design of agent internal and properties, agent interactions and relationships, the multi-agent environment and system. 		
Service-Oriented	<ul style="list-style-type: none"> • Service identification, service categorization, service exposure decisions, choreography, quality of service, component identification, component specification, service realization, service management, standard implementation, service allocation to components, layering the SOA, technical prototyping, product selection, architectural decisions. 		
Business Value	<ul style="list-style-type: none"> • Organization Business Values and Goals: Value to Customer, Value to Team, Value to Process, Value to Workspace, Value to Product. 		

Appendix B: Agile Process Fragment Modelling Using XML (DTD)

```
<!ELEMENT AgileProcessFragment
(id,name,description,purpose,abstraction,tools,people,
developmentstyle,physicalenvironment,precondition,
postcondition,constraints,risks,agility,businessvalue,
alerts)>

  <!ELEMENT id (#PCDATA)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT description (#PCDATA)>
  <!ELEMENT purpose (#PCDATA)>
  <!ELEMENT abstraction (#PCDATA)>
  <!ELEMENT tools (#PCDATA)>
  <!ELEMENT people (#PCDATA)>
  <!ELEMENT developmentstyle (#PCDATA)>
  <!ELEMENT physicalenvironment (#PCDATA)>
  <!ELEMENT precondition (#PCDATA)>
  <!ELEMENT postcondition (#PCDATA)>
  <!ELEMENT constraints (#PCDATA)>
  <!ELEMENT risks (#PCDATA)>
  <!ELEMENT agility (#PCDATA)>
  <!ELEMENT businessvalue (#PCDATA)>
  <!ELEMENT alerts (#PCDATA)>
```


Achieving Success in Supply Chain Management Software by Agility

Deepti Mishra and Alok Mishra

Department of Computer Engineering, Atılım University,
Incek, 06836, Ankara, Turkey
deepti@atilim.edu.tr, alok@atilim.edu.tr

Abstract. Supply chain management is comprehensive software. Due to its scope and unpredictable, complex and unstable requirements, it is not possible to develop it with predictable development process models. Agile methodologies are targeted towards such kind of problems that involves change and uncertainty, and are adaptive rather than predictive. The aim of this paper is to analyze the management and development methodologies used in development of supply chain management software. This paper shows how to overcome risks and handicaps in each development phase of a complex inventive project. It also provides a set of guidelines regarding how the agile methods may be adopted, combined and used in these kinds of projects.

Keywords: Agile methods, DSDM, FDD, Scrum, XP, Adaptive development, SCM.

1 Introduction

Software development is a cooperative game of invention and communication [2]. All agile methods such as Scrum, FDD, DSDM, Adaptive Software Development and especially Extreme Programming recognize this approach. Some of the key practices of agile methods are: scheduling according to feature priorities, incremental delivery of software, feedback from expert users, emphasis on face-to-face communication, pair development, minimalist design combined with refactoring, test-driven development, automated regression testing, daily integration, self organizing teams, and periodic tuning of methods. Working software is the primary measure of success [9]. Agile methods stress early and continuous delivery of software, welcome changing requirements, and value early feedback from customers. Agile methods seek to cut out inefficiency, bureaucracy, and anything that adds no value to a software product [9].

“Manifesto for Agile Software Development” describes the four comparative values underlying the agile position [10]:

- Individuals and interactions over processes and tools,
- Working software over comprehensive documentation,
- Customer collaboration over contract negotiation,
- Responding to change over following a plan.

In particular, agile methodologies are targeted toward problems involving change and uncertainty, and are adaptive rather than predictive [4]. Agile methodologies also emphasize collaboration and team interaction, valuing people over process. Agile methodologies commonly advocate a barely sufficient process [3].

The aim of this paper is to analyze the management and development methodologies used in development of supply chain management software. This paper shows how to overcome risks and handicaps in each development phase of a complex and inventive project. It also provides a set of guidelines regarding how agile methods may be adopted, combined and used in these kinds of projects.

2 Project Background

When the decision was taken to develop supply chain management software, the market analysis was performed within the company. This software was not intended for a specific customer. It was developed to be marketed. A team that includes a marketing expert, a manager and a domain analyst contacted many customers in order to define potential customer sectors and required services and functionalities. This market analysis was actually based on optimization requirements since it is the most important functionality that should be provided. After a couple of weeks, an abstract scope of the product was defined. The base distinction that should be decided is if the project is *predictable* or *inventive* [4] [6]. The development process, management values, planning and estimation models appropriately associated with these two domains are different. So, we analyzed the product and domain characteristics as following:

- Large scale of project
- Project complexity is high
- Acquaintance with the domain was less
- Insufficient requirement specification initially
- Requirement volatility was high
- Variety of customers
- Quick release was important to have an edge in the market
- There were multiple development teams and each team size was small. These teams concurrently developed different parts of SCM (Supply Chain Management) software.
- Getting near the start, reliable estimate of effort and cost was difficult.

Therefore, it would be a wrong decision to choose one of the traditional approaches (predictive methods) (i.e. waterfall methodology etc.) that are used for more predictable kind of projects. As the evidence shows that this project is an inventive project and it should be developed within the motivation of agile and iterative methods [6]. These methods can give you control over unpredictability by benefits of adaptivity. In order to control unpredictability, the key is iterative and incremental development as well as adaptive development. The success of supply chain management software project was based on starting with agile methods and achieving optimal processes by customizing them according to vision and benefiting from adaptivity. Since there is no

systematic way to apply the agile methods, we benefited from that unsystematic approach that involved applying condition specific processes during development. Within that self-adaptivity approach, each team member or sub-team, selected their process according to characteristics of the module that they were developing but tuned their processes to the whole project. Developing with an agile and iterative development process opens the door to the possibility of smaller and more frequent releases. Two primary benefits of this change are increased responsiveness and reduced risk. If responsiveness increases then newly discovered customer needs could be addressed in a much shorter timeframe, most probably with a demonstration since there was a running product although having core functionalities. The primary mechanism that allows a team to steer towards its release goals was demonstrating working software early and often to product owners, customers, and hopefully to end users [7]. Thus, every iteration was an opportunity for the team to get feedback and guidance from customers about how to make the system delivered on the release date the most valuable that it can be to the customer. It was better than presenting the product just by discussions. Adaptive Development is an important concept to reduce the high-risk of a new product, provides ability to customize the product for each customer, and increases the responsiveness of production.

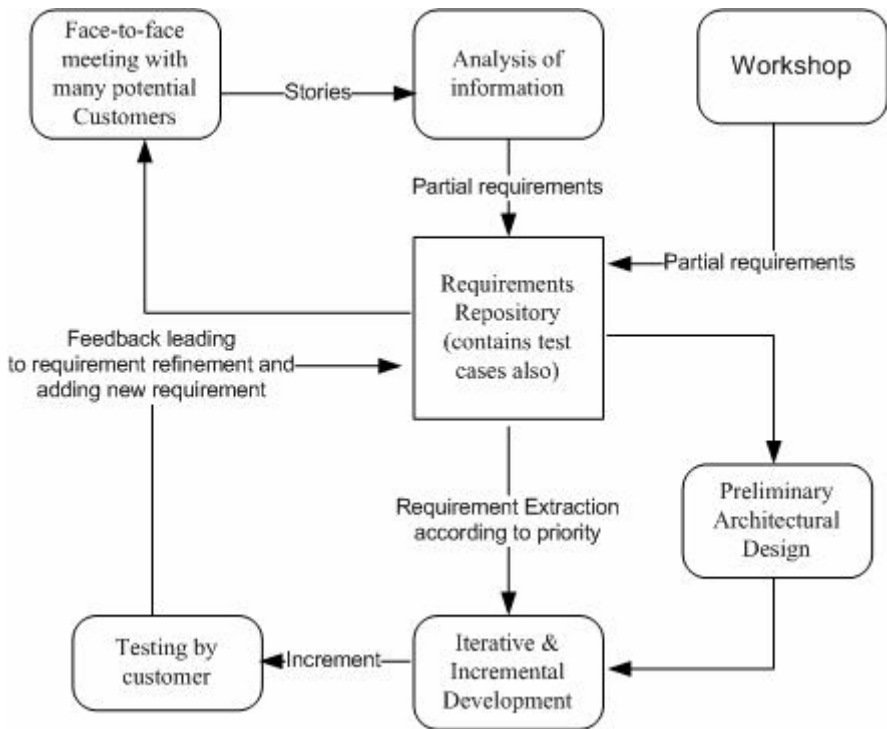


Fig. 1. Development Process for SCM Software

3 Requirement Analysis

Initially, information was collected from many potential customers and then workshops were organized to define a vision and scope, and identify functions and features at a high level (such as just the names of use cases and features). All information defined in those sessions was collected within a requirement repository. At any point in time we have likely collected a large number of “could do”, “should do” and “must do” requirements. These should be aggregated in a centralized repository where they can be viewed, prioritized, and “mined” for future iterations. Requirements should also be readily accessible to all team members, available to be enhanced and revised over time, and remain reasonably current to directly drive testing as they are implemented.

It was observed that the product will consist of:

- a core part (solver and many heuristics for the optimization)
- a support part
 - GIS (Geographical Information System) facility
 - storage of data (should be able to integrate with an ERP system)
 - reports
 - GUIs.

Research for each facility and the domain analysis was performed as a separate, scheduled parallel tasks and each one is assigned to a different team member. A test-driven approach method was used in order to gain knowledge and decide whether it should be developed from scratch or by integrating a pre-developed library or product.

- a) The solver and heuristics was a more domain specific and critical work. We had many constraints such as performance related with the response time and data transition part was most important because we had huge amount of data to be processed within an acceptable time. Other constraints were accuracy, flexibility and to achieve the right solution. The process started with preparing test data, test cases and test environment (hardware and software), in order to evaluate the existing products (open-source as well as commercial) according to functionality and defined constraints. As the team gained more knowledge about domain while testing was in progress, it was realized that some customized solutions should be implemented for some customers. Due to the problems mentioned above, we decided to develop from scratch all the optimization parts including solver.
- b) For the support part, the existing commercial and open-source products were evaluated according to listed functionalities by the business expert. Finally, it was decided that it would be more appropriate to integrate existing open source libraries, not to develop a new one. The mandatory functionalities were provided by these products, and the other functionalities would be developed by team members. Requirements for the additional functionalities which were known were defined and placed into the requirements repository.

The method used in supply chain management software during the study of scope, vision and requirements analysis phase for facility analysis was a less documented version of Dynamic Systems Development Method (DSDM). The fundamental idea behind DSDM is that instead of fixing the amount of functionality in a product, and then

adjusting time and resources to reach that functionality, it is preferred to fix time and resources, and then adjust the amount of functionality accordingly. Also the roles defined in that approach were combined and adapted, according to the team structure of the company. Because of simplicity, more test-driven, close collaboration and communication, XP was also a part of this method. The methods that were used in SCM during this study were hybrid and adapted implementation of the mentioned processes.

4 Project Management

The project management approach used in the supply chain management software project was mainly based on the hybrid usage of Scrum and XP that are two types of agile software development. The framework of Scrum activities, XP's feedback and communication were the concepts that were used for the management processes. The Scrum Backlog and progress tracking approaches are minor variations of XP practices, and they are so simple that they are well within the XP spirit of "do the simplest thing that could possibly work". Instead of using Scrum's 30-day timeboxed iteration length, Scrum's timeboxing concept was used but with XP adaptation, as XP prefers shorter—one or two weeks—iterations. The reasons were inventive type of project, unpredictable requirements, and need of customization for each customer. Also it was important to release the product (may be a core or demo version) as soon as possible for getting an edge in the market. It was not possible to make predictive planning using structural methods. In order to overcome high-risk; plans, estimations, schedules, task assignments and measurements related to the management values should be made within small time intervals (iterations), and at the end of the those intervals feedback should be analyzed in order to plan successive iterations.

The approach used for the supply chain management software project to make plans with development teams is called collaborative planning [2]. The initial planning meeting and each iteration planning meeting was held between project manager, business expert and a representative from each development team. The initial meeting was related to the planning of the scope analysis of GIS, solver and heuristics and ERP inclusion in the system. During each iteration planning meeting, developers estimated the work that they were responsible for. The planning method used in the development of this project was based on the rolling wave planning concept which is a refinement of adaptive planning [2]. It implies that there is no detailed plan for whole project development, unlike for iterations. There is no fixed plan of how many iterations there will be, how long they are, or what will happen in each. But there were milestones with dates defined in the development; the path of iterations to achieve those milestones is left flexible or adaptive. It is better than trying to plan a detailed speculative schedule for an unpredictable project. Such adaptive planning is the key idea in agile methods. According to existing circumstances an abstract plan was done but again not for the whole lifecycle of the project. The requirements, functionalities were defined for the baseline. The primary plan in detail for each iteration was made in order to achieve the baseline. During iteration planning meeting, goals were listed that would be accomplished within a week period. The feedback from previous iterations was used to plan current iteration. Goals were separated into one week accomplishable tasks according to estimations done by team members who were responsible

for the tasks. It means that each iteration time was fixed and tasks were divided into subtasks that could be accomplished within that fixed time period. During those meetings, technology, program structure and overall system functionalities were also discussed according to the information collected from many potential customers. An iteration planning was based on existing emergent requirements, presence of resources, and degree of knowledge of current iteration. Therefore, planning was closer to optimal in terms of working towards milestones; each step can be performing the most skillful thing we know to plan regarding risk, productivity, and effectiveness because each planning step is taken with maximum—and fresh—information. We take a step, and then ask, "Given what we now know, what is the most skillful thing we should do in the next step to work towards our milestone goal?" And repeat. The milestones were used for each internal release due date and there were many iterations planned within that period time.

5 Architectural Design

Preliminary architectural design was also done using the initial requirements. This is supported by Mead [8] that architecture modelling and trade studies are important activities that should take place during requirements engineering activities, not after requirements have been defined. Software architecture must be considered during requirements engineering to ensure that the requirements are valid, consistent, complete, feasible etc. [8]. There were many development teams working concurrently on different parts of SCM software. To avoid any confusion between these teams and also to have a common picture of what they were developing, a team consisting of project manager, development team representatives and a business expert made the core architecture of the system. This was the structure that specifies the whole system as major components, modules; collaborations, interactions and interfaces between them, plus the responsibility of each module. All the defined modules and components were drawn as black boxes and the interactions between them were represented by arrows. Development of each module was assigned to different teams as parallel tasks. The responsibilities and collaborations were defined clearly for each module (i.e. Controller, IO manager) and sites (DBMS, GIS, Application Server). This structure was also allowed to change as a result of customer's feedback from future iterations. Since it was basis structure and there was collective ownership on that part by the team members, it was important to document that structure in order to be accessed easily. The diagrams, responsibilities, functionalities and scenarios were recorded through the documents. Object-Oriented design techniques were used for architectural design so as to increase applicability of the iterative and incremental development process because OO design provides modularity, minimum coupling and maximum cohesion, thus a flexible structure. Another benefit of using OO techniques was to define the tasks parallel, since all modules provide encapsulation and loosely coupled structure, each could be developed independently as a sub-product and then be integrated easily because of well-defined interfaces. Once the core is built, team leaders returned to their respective teams and the development was done in parallel with multiple teams by using short iterations. Each team leader had a clearer picture and common vision due to the architectural design and could better convey and main-

tain that for the rest of the project. Further, each acts as a liaison to the other teams. Also, after spending some close time with the other team leaders, there was improved communication between them.

Each defined module (such as Computational, IO managers, GUIs, GIS, Reporting) can use different development processes but all of them should be synchronized to the development process of that architecture because it defines the overall system. Feature Driven Development (FDD) which is a type of agile software development approach and based on the iterative and incremental development was used during development of that structure. The FDD approach does not cover the entire software development process, but rather focuses on the design and building phases [1]. However, it has been designed to work with other activities of a software development project and does not require any specific process model to be used.

6 Project Development

This was done using Agile development methods (FDD and with Extreme programming XP), their adaptive and hybrid usage. The programming language used during development was pure java based on full object-oriented structure. Most Agile software development literature cites its use for application development projects, often implemented in object-oriented languages [4].

FDD consists of a set of “best practices” and the developers of the method claim that even though the selected practices are not new, the specific blends of these ingredients make the five FDD processes unique for each case. All practices available should be used to get the most benefit of the method as no single practice dominates the whole process.

The XP Values are Communication, Simplicity, Feedback, and Courage. The essence of XP truly is simple. Be together with your customer and fellow programmers, and talk to each other. Use simple design and programming practices, and simple methods of planning, tracking, and reporting. Test your program and your practices, using feedback to steer the project. Working together this way gives the team courage [5]. XP aims at enabling successful software development despite vague or constantly changing requirements in small to medium sized teams. Short iterations with small releases and rapid feedback, customer participation, communication and coordination, continuous integration and testing, collective ownership of the code, limited documentation and pair programming are among the main characteristics of XP. One of the fundamental ideas of XP is that there is no process that fits every project as such, but rather practices should be tailored to suit the needs of individual projects.

For each iteration (new functionality, defect fix, changing) during the development of the core part (solver and many heuristics for the optimization), the requirements were selected from the repository according to their priority and defined functionality for that iteration. Their use-cases and working scenarios were prepared by the domain expert and supplied to the development team. And then within the defined scope and process for the overall structure, the design, planning and implementation was done for each module that would be developed in order to make the increment for the system. When each module in a particular increment was developed and integrated to the

overall system then the testing of the overall system was performed and result was validated. Later, the product was released (end of the release or end of the iteration).

The other parts of the system included in the *support* category were GUIs, GIS, reports, and storage of data. As we mentioned before, these parts of the system were used as a layer to interact with the users of the system. The development of GUIs and their integration to the system was done by a team that included highly-skilled persons of that subject. The requirements and scenarios of usage were defined by the business expert and development team. The required GUIs and their functionalities were defined for each iteration and release in corresponding meetings in an abstract way. According to plan that was defined for each iteration and release of the overall project, the meetings had been performed before each iteration or release to define everything related with GUIs between the team members that were responsible for development of GUIs and the business expert. The iterations of GUIs were planned according to development of selected GUIs which were prioritized and already defined in the iteration plan of the overall project. In order to develop the GUIs, first the business expert drew the GUIs using a design tool, and also documented the usage scenario of those GUIs. Also the information about how and in which part of the system, the data that would be gathered from these GUIs would be used. Later, a development team prepared the empty forms of those GUIs just satisfying visual requirements using all the documentation. After the validation process by unit and integration testing of those empty forms, they were integrated according to usage scenario and their data bindings were done according to defined data flow to already built structure of the overall system. The validation of those parts was done with integration tests by the development team and the business expert.

For the GIS (Geographical Information System) component, an open-source library was chosen to be integrated and used in this product. The main functionalities were provided by the library; all other required functionalities and services that were planned were provided by our product. This part was also done with test-driven and early development approach used in early phases of scope, vision and requirements analysis.

The Report component part was also based on an open-source library. Since most of the reports were already defined within a business domain, the main selection criteria were technology that was used within the project, adaptability, flexibility, portability and usability of the candidate libraries.

Users can export or import their real data from or into this system through storage. These activities may be done offline using corresponding modules and user interfaces, visual services provided by the system itself or may be online through the integration of the storage part of the system to the storage mechanism of the customers that may be a database, an existing system or an ERP system. These considerations were discovered in the early phases of scope, vision and requirements analysis. This part was defined as a black box in the overall system design but for defining interfaces, a test-driven and early development approach was used at early phases of scope, vision and requirements analysis. The ERP products were analyzed and tested, their results and possible requirements of database and availability of existing systems were discussed in meetings that included development team members, a business expert and project manager. The vague or conflicted suggestions were resolved by defining the test cases and test data. Using the early development approach, a prototype or demo version was

developed and tested. In these studies, knowledge was gained about development of storage part, possible integration of that part to other systems or ERP systems with their tested interfaces, usage of the databases, functional and non-functional requirements, constraints and the scope of this part was defined. The actual development and detailed plan of that part was left to the phase of delivering the final product to the customer because as mentioned before, this was a new product and it was not developed for a specific customer. But an initial design and development of the storage part performed with a flexible, adaptable and portable structure in order to reduce the delivery risk. As we have already gained knowledge and experience in the early development and testing of that part and a flexible and adaptable architecture and interfaces were developed using the agile development approach, the delivery and acceptance of customers would be successfully achieved.

The other agile development method used in the development of supply chain management software just for demo versions, had different characteristics. During the walkthroughs with the customers, they requested some demonstration of customized solutions. Whenever such requests existed, all release and iteration plans were updated or new plans were prepared. After it was decided to make such a demonstration, the product backlog of that new release was prepared and backlog items were prioritized by the business expert and product owner. The planning of that emergent release performed within a release meeting that might not be a regular meeting. If the release backlog items fit or are already included in existing release (current release under development), then the existing release plan had been updated for the successive iterations, but if it did not then the existing release plan was changed or redirected to an emergent demonstration release. The method used for the emergent development was different from the existing one, which had shorter timeboxes, more customized solutions and more visible development. During the meeting of this emergent release plan, all tasks defined clearly, listed on a white board, estimation of each task was done by the module developers. A schedule-driven approach was used during the iteration planning because the time was fixed according to demonstration date and all definition of backlog items and their priorities were defined according to that approach. The defined tasks and assigned team member's name were written on cardboards and hung up on walls. Team member responsible for a particular task was required to update those cardboards as the task progresses. So progress of each task can be seen through those cardboards if it was finished, tested, committed or should be validated. If some tasks were dependent on each other, the team member could follow progress and update their individual plans without interrupting other team members. This is a combined and customized approach of agile methods and a very useful approach for such emergent releases. This approach increases the visibility of development and the awareness of how progress is going on. It also increases the morale and focus of the development team members.

7 Conclusion

Agile methodologies are targeted towards problems involving change and uncertainty, and are adaptive rather than predictive. These methods can control unpredictability by using benefits of adaptivity. In order to control unpredictability, the key is iterative

and incremental development as well as adaptive development. As our Supply chain management software project was an innovative project, key practices of agile methods such as scheduling according to feature priorities, incremental delivery of software, feedback from expert users, emphasis on face-to-face communication, pair development, minimalist design combined with refactoring, test-driven development, daily integration, self organizing teams, and periodic tuning of methods helped significantly to achieve its successful implementation. As agile methods provide flexibility, it encourages the development teams and individuals towards creativity which is essential for successful implementation of innovative projects.

As it was innovative, large scale, high risk project, we formally did the architectural design along with documentation. This design documentation played an important role in the successful implementation of this project and it will be helpful in the maintenance phase also. The most important characteristic of development methods used in this project is that they were adapted to circumstances in each phase of the development. Agile development methods were combined so that new approaches are resulted from this self-adaptivity approach. It was not possible to complete and fix all the requirements because of the business domain and product characteristics. The software development team handling such a large project was small. Communication between team members was strong, as they were working in a small office and a business expert already aware of the business domain was in the same office so that they could interact whenever needed. The development approaches used in the supply chain management software project involved less documentation than the process-oriented approaches, usually emphasizing a smaller amount of documentation for a given task or only the critical parts were documented.

References

1. Abrahamsson, P., Salo, O., Ronkainen, J., Warsta, J.: Agile software development methods- Review and analysis. VTT Publications 478, 1–112 (2002)
2. Cockburn, A.: Agile Software Development. Addison-Wesley, London (2000)
3. Cockburn, A.: Agile Software Development. Addison-Wesley, London (2002)
4. Fowler, M.: The New Methodology, (April 2003), <http://www.martinefowler.com/articles/>
5. Jeffries, R., et al.: Extreme Programming Installed, vol. 172. Addison Wesley Longman, Redwood City (2001)
6. Larman, C.: Agile and Iterative Development: A Manager's Guide. Addison-Wesley, London (2003)
7. Leffingwell, D., Muirhead, D.: Tactical Management of Agile Development: Achieving Competitive Advantage, Rally Software Development Corporation, p. 1–23 (2004)
8. Shekaran, C., Garland, D., Jackson, M., Mead, N.R., Potts, C., Reubenstein, H.B.: The role of software architecture in requirements engineering. In: Proceeding of the First International Conference on Requirements Engineering, pp. 239–245 (April 18–22, 1994)
9. Tichy, W.F.: Agile Development: Evaluation and Experience, University of Karlsruhe, tichy@ipd.uka.de
10. www.agilemanifesto.org

Software Measurement Programs in SMEs – Defining Software Indicators: A Methodological Framework

María Díaz-Ley¹, Félix García², and Mario Piattini²

¹ Sistemas Técnicos de Loterías del Estado (STL)
Gaming Systems Development Department 28234 Madrid, Spain

maria.diaz@stl.es

² ALARCOS Research Group

Information Systems and Technologies Department
UCLM-Soluziona Research and Development Institute
University of Castilla-La Mancha,
13071 Ciudad Real, Spain

{Felix.Garcia, Mario.Piattini}@uclm.es

Abstract. Implementing a measurement program is not an easy task. It requires effort, resources, budget, experts in the field, etc. The challenges to successfully implement a measurement program in small settings are considerable and greater than in large companies. Small and medium-sized enterprises (SMEs) have an additional handicap: the existing methods and frameworks that support measurement programs such as Goal Question Metric (GQM), Goal-Driven Software Measurement, GQ(I)M, PSM and ISO/IEC 15939 do not fully satisfy the needs of such companies. We propose MIS-PyME, a methodological framework which supports small and medium enterprises (SMEs) in establishing software measurement programs, especially as regards the definition of software indicators. MIS-PyME is based on GQ(I)M and aims at supporting SMEs measurement activities related to software process improvement tasks. This framework has been applied in STL, where the main benefit derived from the use of MIS-PyME has been an effortless, more accurate program definition integrated into software process improvement practices.

Keywords: Methodological framework, measurement programs definition, GQ(I)M, MIS-PyME.

1 Introduction

Measurement is a key technology for supporting the basic tasks of an improvement program. Collecting and interpreting well-defined data provides organizations with the necessary information to make well-founded decisions about process improvement[1]. Process improvement results in increased productivity, better quality, and reduced cycle time, all of which make a company competitive in the software business[2].

Small organizational units are just as likely to be confronted with demands for credible evidence about their ability to deliver quality products on time and on budget

as large, multinational organizations. Similarly, managers in small settings are equally or even more likely than their counterparts in larger organizational units to have to make well-founded business decisions about process improvement and technology adoption, and must have the wisdom of taking new business opportunities. Therefore, implementing serious measurement programs is even more important in small organizational settings [3].

Although measurement is applied in various areas, it has proved to be a complex and difficult undertaking in the field of software and especially in the context of SMEs [4]. The existing obstacles include limited resources and a limited budget, a need for training, tight schedules, poor software measurement knowledge, a low cash flow, a restricted mentality as regards software measurement, etc. [5]. [6]. [7].

Table 1. Requirements for measurement program models suited to SMEs

SMEs Re- strictions	Measurement program model for SMEs Requirements	Benefits
Limited re- sources, schedule and budget	<ul style="list-style-type: none"> - Easy to Understand and Manage (EUM) - Effortless (EFL): It should guide users and help them define measurement programs in an effortless way - Complete. (COM): It should cover their process improvement needs 	<ul style="list-style-type: none"> - prevents users from spending too much time defining measurement programs. - does away with the need to contract measurement experts to develop measurement programs.
Limited training , poor soft- ware meas- urement knowledge	<ul style="list-style-type: none"> - Informative (INF): It should contain full information about the measurement possibilities, the benefits of using measurement and its potential use in technical and organizational management. 	<ul style="list-style-type: none"> - makes users learn about measurement. - makes users learn about the benefits deriving from its use. - makes the measurement program more accurate.
	<ul style="list-style-type: none"> - Integration into the Processes (INTP): It should contain full information about its practical integration into the organization’s software processes 	<ul style="list-style-type: none"> - help measurements activities implemented for different aims be coherent. - measurement usefulness can be better understood since its potential use is clearly shown when the measurement goal is derived from the software process practices. - measurement activities are better integrated into software processes.
	Measurement Maturity Model (MMM): It integrates a measurement maturity model	<ul style="list-style-type: none"> - make the organization progress across software measurement. - advise the user to implement those measurement goals which suit its measurement maturity and prevent the user from defining measurement goals which cannot be successfully reached.

Our research aims at overcoming some of the obstacles mentioned in order to favor the implementation of measurement programs in small settings. MIS-PyME (Marco metodológico para la definición de Indicadores de Software orientado a PyME) is based on GQ(I)M [8, 9] and its main focus is on defining software indicators, which are the basic instruments for the analysis and interpretation of measurement objectives, and therefore for decision making.

With MIS-PyME we intended to provide a methodological framework that would help implement measurement practices which would support software process improvement activities suited to SMEs.

After listing the restrictions SMEs are confronted with in terms of software measurement, we conclude by showing the requirements which software measurement models should fulfill in order to be suited to SMEs regarding the definition of measurement (Table 1). We will use this table to look at the benefits and the contributions of our work.

This paper is organized as follows: Section 2 brings the MIS-PyME into context. Section 3 provides a more detailed insight into the framework, and briefly lists its specifications. Section 4 provides a case study examining the benefits derived from the implementation of this framework in a medium-sized company and Section 5 outlines the benefits resulting from present and intended research.

2 Related Work

In this section we show the most outstanding models and standards which support the definition of measurement programs, and the deficiencies of such standards as regards their implementation in SMEs. Special attention should be paid to the measurement models for the definition of measurement for SMEs which will be described below. An analysis of software products, process re-use and the integration of these practices into the measurement model will also be provided.

- Goal Question Metric GQM [10]. According to this model, there should exist an independent team which leads measurement program initiatives. This team must possess deep knowledge of measurement issues and have unrestricted access to the leaders of each project. Implementing this model in a company made up of 10 people which desires to perform measurement initiatives by itself will pose a challenge. Besides, there are authors who think that GQM encourages practitioners to define measures that are difficult to analyze and collect [10-12]. This is a drawback, especially for SMEs.
- In 1996, the SEI (Carnegie Mellon Software Engineering Institute) published the Goal-Driven Software Measurement guidebook [8]. This extension to GQM is called Goal Question Indicator Metric, GQ(I)M. Even if the definition of a measurement program is easier with this model than with GQM [10] (since it provides an intermediate layer - the indicator layer - as well as analyses, measurement examples, and supporting templates [9]), the project team still needs to have great knowledge and high insight on the field of measurement, and must make big efforts to define the measurement program.

Moreover, neither GQM [10] or GQ(I)M [8] give any guideline as to how to integrate measurement into software processes (INTP). GQ(I)M[8] contains some information on how to learn about measurement possibilities but GQM [10] does not, and none of them contains a measurement maturity model.

- PSM (Practical Software and Systems Measurement) [13] is a framework created by the Department of Defense in 1994 and its goal is to provide project and technical managers with the Best Practices and guidelines in software measurement. PSM focuses on issues in software projects which typically require management and control. It does not however clearly show the usefulness of measurement programs in supporting process improvement, and it does not provide any guideline on how to help the company improve through measurement maturity.
- ISO/IEC 15939 [14] lists the activities and tasks required in order to successfully identify, define, select, apply, and improve software measurement or the measurement structure in the organization under a generic project. However, it does not give any detailed methodology for defining the measurement program, it is not easy to implement effortlessly, and it does not give any information regarding measurement benefits, software process integration or measurement maturity.

Table 2 sums up how the above measurement models fulfill the requirements needed for a measurement definition model to be adapted to SMEs. The resulting values may be: “No”, “Yes”, “Some” or “-”, which occurs when the concept does not fit into the model.

Table 2. Fulfillment of the measurement models requirements

Requirements	GQM	GQ(I)M	PSM	ISO/IEC
EUM	YES	YES	YES	SOME
EFL	NO	NO	YES	NO
COM	-	-	SOME	-
INF	NO	SOME	YES	NO
INTP	NO	NO	NO	NO
MMM	NO	NO	NO	NO

There are some studies that tailor some of the most widely known software measurement models and standards mentioned to the needs of SMEs. One of these is the work by Gresse et al. [4], who suggest the GQM Lightweight method. This approach consists in integrating the re-use of context-specific quality and resource models into the GQM model. This makes it unnecessary to start the model from scratch. This approach saves some effort and by so doing matches one of the aims of our approach, although the means to achieve this differ.

As far as process and product re-use are concerned, the latter having to do with the re-use of indicators and measures in our methodology, Medonça et al. [15] approach is to be taken into account. It is understood as a measurement model which integrates the previous experience of a company into its methodology. Finally, some studies believe in the benefits of re-using software products, processes, and experiences to achieve higher quality systems at a lower cost [16]. Some enterprises have developed this issue as observed in [17], [18].

3 MIS-PyME Specification

Firstly, we present an overview of MIS-PyME; we go on to look at the main work products of the framework, and provide a more detailed analysis of MIS-PyME. We finish by summing up what the contributions of this measurement framework have been.

3.1 MIS-PyME Framework Overview

MIS-PyME Framework deals with the definition of software indicators for SMEs. It is based on GQ(IM) [8, 9]. However, the steps of the methodology have been modified and tailored to the needs of SMEs. The main adaptations are:

- MIS-PyME supports this definition by providing measurement goals and indicator templates which function as a guide to the definition of measurement programs in the frame of software process improvement.
- MIS-PyME includes a useful database of indicators and measures taken from successfully implemented measurement programs.
- MIS-PyME provides a measurement maturity model which helps the company improve its software measurement in an orderly fashion.

It must be borne in mind that the scope of MIS-PyME in the measurement process only covers the “measurement planning process”.

3.2 MIS-PyME Specifications

This section gives an overview of MIS-PYME main work products and characteristics.

MIS-PyME Work Products. The main work products identified in MIS-PyME are the following:

- MIS-PyME guide: A document which is intended as a guide for MIS-PyME model. It is focused on two working methods. The first one is used when a precise measurement goal is required to be defined for process improvement. The second one guides the organization and helps it progress through measurement maturity in an orderly fashion.
- MIS-PyME measurement goals table: MIS-PyME framework proposes a set of structured measurement goals usually required to implement improvement activities related to software processes. The goals are organized in a structure based on the measurement maturity required to implement each goal defined.
- MIS-PyME indicator templates: An indicator template is defined for each measurement goal. The indicator template will guide users and help them define indicators and measures for a specific measurement goal. An indicator template shows, among other things, the conditions required to successfully implement the indicator regarding previous indicators required, conditions which must be fulfilled in order to successfully implement the indicator and how to integrate this indicator into the software process. These are typical questions which the indicator tries to answer. Typical outcomes and their related analysis may also be described and show the user what the potential of an indicator is, etc.

- MIS-PyME database: Each MIS-PyME indicator template contains a set of examples of real indicators which have been defined in a successfully implemented measurement program. The measures used as input for these MIS-PyME indicator templates are also included in the database.

MIS-PyME Roles. MIS-PyME defines only three roles which have to be performed by different people. The first one is the measurement analyst, who should be familiar with the activities and processes carried out by the software development and maintenance department. It is preferable if his or her usual work relates to the definition of requirements, testing, configuration management or security, rather than design or development tasks. The second one should be a top manager who supports the measurement program initiative and has in-depth knowledge of the working method, software processes and process improvement needs. The third one is the reviewer, who will act at the “Verify the measurement program” status of the methodology. This role will be played by at least two people; one of them will be a project manager and the other one, a developer. The other steps in the methodology are performed by the measurement analyst, who should ask the top manager for all necessary information.

MIS-PyME Methodology Steps. We shall now briefly describe the MIS-PyME methodology and the most important changes made in MIS-PYME methodology as compared with GQ(I)M [8, 9] basic model. Figure 1 succinctly outlines this methodology.

1. Identifying your process improvement goals: MIS-PYME first step refers to GQ(I)M [8] step three, “Identifying your sub-goals”, but our initial goals will derive from the software process model and they will not be business goals, but management process improvement goals.
 - Description: Defining the process improvement goals that you want to carry out aided by software measurement. Identifying the related entities that will help achieve this goal.
 - Input: Needs of the organization in order to establish and improve software processes. Output: List of process improvement goals and related entities.
2. Formalizing measurement goals
 - Description: Measurement goals are specified. After that, the object of study, the purpose, the point of view, the environment and the measurement constraints are defined. The template for the definition of measurement goals has been changed with respect to that in GQ(I)M[8] so that it is easier to use. This has been achieved inasmuch as the purpose of the measurement is restricted to a set of precise purposes, as Briand et al. [5] measurement goal template does.
 - Input: List of process improvement goals and related entities. Output: MIS-PyME measurement goal templates filled out.
3. Identifying if measurement goals have been defined:
 - Description: Once measurement goals have been defined, verification of whether the MIS-PYME measurement goals table already defines the required measurement goals may follow.

- Input: MIS-PyME measurement goal table. Output: Register in MIS-PYME measurement goal table and related MIS-PyME indicator template.
4. Defining Indicators: This is a three step status.
- Description: Indicators required to implement measurement goals are defined.
 - Input: MIS-PYME indicator templates related to each measurement goal. Output: MIS-PYME indicator templates filled out.
- 4.1. Specifying the indicators
- Description: If the measurement goals were in the MIS-PYME measurement goals table, the measurement analyst might take a look at the recommendations, restrictions, preliminary actions, information needs, etc. according to the MIS-PYME indicator template established for that goal. He/she should otherwise be guided by general recommendations provided by the generic MIS-PYME indicator template.
 - Input: MIS-PYME indicator templates related to each measurement goal. Output: MIS-PYME indicator templates filled out.
- 4.2. Searching in MIS-PyME database:
- Description: When defining an indicator, measurement analysts may check for any examples in the database related to the MIS-PYME indicator template required for the desired indicator. If a suitable one is found, they can directly and effortlessly adapt the indicator proposed to the measurement program being defined.
 - Input: MIS-PYME indicator templates related to each measurement goal. Output: MIS-PYME indicator examples.
- 4.3. Identifying sub-goals derived:
- Description: Any of the questions posed or the prerequisites recommended in the MIS-PYME indicator template table may lead to another measurement goal. We call these measurement-derived goals, which may also have their corresponding measurement goal in the table for MIS-PYME measurement goals and their corresponding MIS-PYME indicator templates. Step 3.1 and 3.2 may then be repeated until all measurement-derived goals and their relevant indicators have been defined.
 - Input: MIS-PyME indicator template filled out. Output: list of derived measurement goals.
5. Defining your measures and identifying the actions needed to implement them: Step 7 and 8 of GQ(I)M[8] are joined at one point.
- Description: The measures that have to be collected are identified in detail and defined in the checklists. It is defined which data is to be included/excluded from the measured values, as well as how the data will be collected. The ability of the organization to obtain the measures is analyzed, and the way in which they could be collected is established. If it is not possible to collect the desired data, the indicator specification may be modified based on this information
 - Input: MIS-PyME indicator templates filled out. Output: measure definition checklists and data collection specifications.

6. Integrating measurement.
 - Description: Integrating the measurement activities into previous measurement processes and into other software processes is the aim of this step. MIS-PYME provides guidance as to the structure of the (recommended) html document where all the indicators, measures, and measurement sub-processes of the organization are defined.
 - Input: MIS-PyME indicator templates, measure definition checklists and collection specifications. Output: (updated) measurement process specification and (updated) software process specification.
7. Verifying the measurement process
 - Description: The measurement process resulting from the process is verified by reviewers and modified if required.
 - Input: Measurement process specification (updated) and software processes (updated). Output: verified measurement process specification and verified software processes.

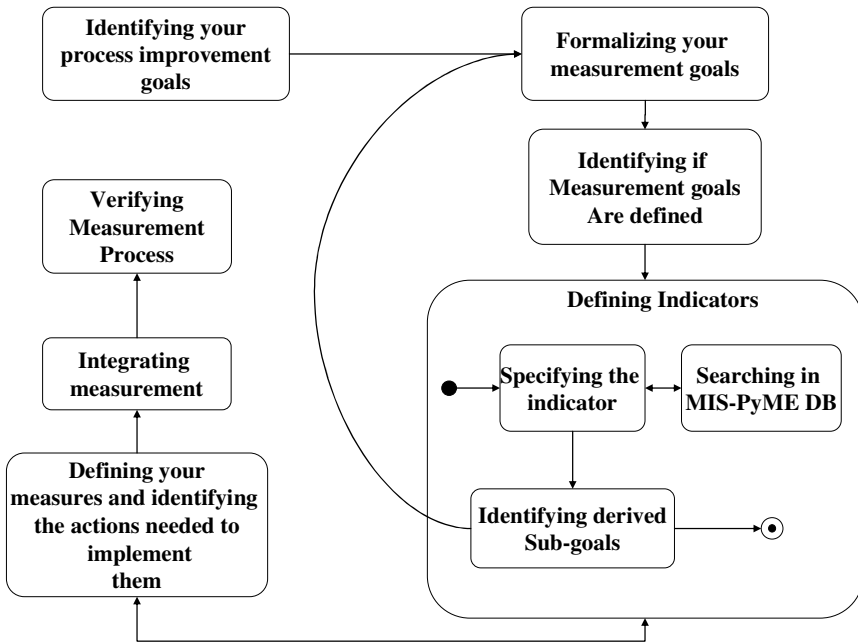


Fig. 1. MIS-PyME methodology steps

3.3 MIS-PyME Contribution

Next table (see Table 3) shows how MIS-PyME fulfills the requirements mentioned in section one for the definition of software measurement programs suited to SMEs. After examining the table we can conclude that none of the measurement models shown in section 2 equals our contribution in terms of requirements met.

Table 3. Contribution of MIS-PyME

REQ	How it is fulfilled	Description
EUM	MIS-PyME Guide provides two working methods	MIS-PyME supports the definition of measurement through any process improvement goal or progressive measurement maturity.
	Linkage between MIS-PyME measurement goals table – indicator templates –examples in DB	Users can easily obtain an overview of the whole definition process of a typical measurement goal by defining the need (improvement process practice) to define measures.
EFL	MIS-PyME measurement goals table	Typical process improvement measurement goals have been written down in the structured table, which should help users to shape their idea of measurement goals.
	MIS-PYME indicator templates	Questions asked to project managers are better focused thanks to the guidelines provided in the section “questions” in the indicator templates.
		Section “analysis and interpretation” in the indicator templates gives guidelines regarding the type of analysis that can be performed on the indicator, as well as its possible outcomes and interpretations.
		Guidelines regarding typical indicator inputs or how the indicator could be graphically displayed make the definition easier, especially if the measurement analyst is not an expert in the measurement area.
MIS-PYME database	These indicator and measure examples related to a measurement goal make it easier to define the measurement program and let the user improve and check that definition.	
COM	MIS-PyME measurement goals table	The measurement goals proposed. The related indicator templates and examples cover the basic needs for typical software process models. This deals with the measurement of product, projects and software processes.
INF	MIS-PYME indicator templates	Section “analysis and interpretation” gives guidelines regarding the type of analysis that can be performed on the indicator, as well as its possible outcomes and interpretations, which also show the potential of the use of measurement.
	MIS-PyME measurement goals table, indicator templates, Data-Base	All the practical and theoretical information contained in these products helps users learn and understand the potential of the use of measurement and facilitates its exploitation.
INTP.	MIS-PyME measurement goals table	The link between process improvement and the measurement program is clearly established.
	indicator templates	Fields regarding “post definition” and “indicator integration” give guidelines about how to integrate the indicator into software processes.
IMM	MIS-PYME indicator templates	“Restrictions of purpose” and “evolution” fields meet the purpose of adjusting the indicator definition to the current measurement maturity.
	MIS-PYME guide	This guides users and helps them define and implement measurement goals according to their maturity level.

4 Applying MIS-PyME Framework in the Context of STL

The first application of MIS-PyME in real small settings is described in this section. The goal of this application was not to formally validate the framework but to obtain some hands-on experience with the framework and see if it was fit to solve the problems of a real organization. This gave us the opportunity to detect the deficiencies existing in its practical application and obtain the first feed-back regarding the acceptance of the framework by the organization and the preliminary benefits brought about by its use.

4.1 Introduction

Software measurement initiatives have been encouraged for many years by the software development and maintenance department in this company, which is formed by 39 people. However, the measurement process defined was not accurate enough and had not been properly established throughout the department. Some deficiencies had been detected, especially regarding those measures dealing with reliability. One of them was for example that the purpose of some of the indicators implemented had not been accurately established. An effort was made to try and evaluate the reliability of the project at a time when no collective and stable model existed for the definition of a fair evaluation threshold. Also, the input data was not enough for the analyses and interpretations that had been performed on projects.

The need of accurately measuring products, projects and processes in the organization increased since the number of projects and their scope was gradually increasing. A commitment to establish a well defined and accepted measurement program started to take shape.

The goals of the measurement program were as follows:

- To improve the definition of the indicators which had been previously established but had not been much accepted. These are related to the following areas: reliability of products under production, reliability of products under development within the scope of a specific project, and precision of the estimates of the duration of projects.
- To define other indicators required for project, product and process management related to estimation, development, service quality, and software process effectiveness. These were divided into two periods:
 - In the first period, only the measurement objectives which required data to be collected just once during the project or those objectives related to the monitoring of a process were defined and implemented
 - In the second phase, those measurement objectives which required data to be collected quite frequently, such as those related to project monitoring, would be defined later once phase 1 had been implemented. We will not deal with this phase in the present paper.
- To establish an easy and well documented methodological framework for the definition of measurement programs.
- To develop easy measurement management tools.

4.2 Development and Implementation of the Measurement Program

The measurement program was carried out by a person from the development department whose measurement knowledge was not bad, but he was not an expert in the area. The director of the department, who supported the initiative and had a good knowledge of the existing measurement needs, was the supervisor.

Initially, the measurement program was developed using GQ(I)M[8] measurement definition model. It took one month to define the first phase of the measurement program. The results were reviewed by some research members and by the director of the development department. Among the weaknesses and potential improvements that were detected are the following :

- In spite of the time the measurement analyst had spent trying to understand the limitations of the purpose of the indicators, he failed to define some of the same.
- The measurement program was not meant to be applied to the development processes existing at the time, and there was not a clear idea as to what these indicators were intended to be used for.
- The measurement process was not well structured. It was not well documented either, and turned out to be hard to follow.

These and other problems detected were not solely caused by the fact that GQ(I)M was used. If a measurement expert had defined the measurement program, he or she might have succeeded, but it became evident that it would be quite easy to fail if GQ(I)M was used even if the scope of the measurement program was restricted and known, and even if the responsible for defining the measurement program was already somehow familiar with software measurement.

Some aspects of GQ(I)M which could have lead to an unsuccessful measurement program were indicated by the measurement analyst:

- Since the measurement analyst did not want to bother project managers, he posed some of the questions to specify the goal himself. He failed in some of them.
- It was difficult to specify the indicators which would fulfill the measurement goal.
- It was difficult to know the purposes of the indicator which it would be possible to implement.
- It was difficult to define how to analyze the indicator.
- It was difficult to implement the measurement goal since many measures could not be collected.
- It was difficult to document the measurement process and integrate it into other measurement processes already established in an easy and understandable fashion.

Due to these problems, the measurement program had to be interrupted. It would be resumed two months later, since at that moment the responsible for the measurement program had to devote his time to some other urgent project.

In the second attempt at using MIS-PyME framework, the participants in the measurement program were the same as in the first attempt. The definition of the measurement program took one month and a half.

As may be noticed, the measurement program took more time using MIS-PyME than GQ(I)M. The reason is that, starting from the first version, the measurement

program was more accurately defined and it was reviewed several times, as the development director trusted the resulting measurement program definition, but did not trust the one defined using GQ(IM).

Part 1 of the measurement program defined 5 measurement goals and 20 software indicators. The data collected for the other indicators that had already been implemented stayed the same, but the way of analyzing the indicators was modified in most of the cases.

The implementation of part 1 of the measurement program required some developments to be made in order to create tools that automated the measurement activities as much as possible and tailor others. These developments were the following:

- Some new reports were implemented in the request for change & incident management system (Remedy).
- Some excel sheets were created in order to manage the measurement program. One excel sheet was developed to manage development processes; the other, for product management issues. The plan was to start using these easy sheets, and a more complex and powerful tool in the future if necessary.

In figure 2 we show one of the implemented goals which consists in evaluating the quality of project development services. This goal was defined by indicator IND-PROC-CALIDADSRV which uses two input indicators. One indicator characterizes the deviation between the first formal duration estimation of the projects and the real duration of the same (IND-PROC-INEXACDURACION), and the second indicator measures the reliability of the software developed by measuring the incidents occurred in the course of production for each project one month after the last installation of software (IND-PROC-FIABIMPL) related to the project.

The first indicator had already been defined and used before this initiative, but this was not the case of the second. The common goal had been well defined by means of IND-PROC- CALIDADSRV indicator definition.

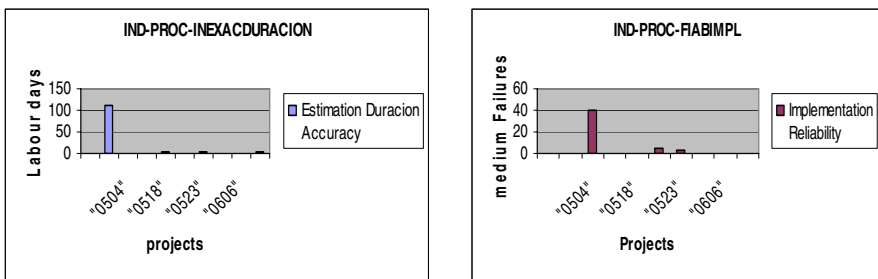


Fig. 2. Service quality indicator (IND-PROC-CALIDADSRV) which contains two input indicators: IND-PROC-INEXACDURACION and IND-PROC-FIABIMPL

Once the measurement program defined was implemented we analyzed this indicator at the time as defined and required. IND-PROC-FIABIMPL had to be retrospectively collected so as to be used with the other indicator IND-PROC-INEXACDURACION. This indicator allows us to understand and evaluate the general quality of a software

project provided to our clients, and to ascertain if the on-time release of the software product had a negative impact on software reliability, etc.

4.3 Lessons Learnt

The lessons learnt were analyzed after the first period of analysis was performed by the responsible for the definition of the measurement program and the director of the development department. We have divided their conclusions in two: The measurement program that resulted from the experience and the methodology (MIS-PyME) used.

As far as the measurement program is concerned, they reported that they trusted this measurement program but did not trust other previous measurement programs. The reasons were:

- This measurement program covers most of their basic needs.
- Its definition is quite complete as regards the questions which should be answered for each indicator and the analysis and interpretation that can be done, which also prevents users from making a free analysis or interpretation since they have to adjust to what is defined.
- The measurement program was well integrated with the software processes.
- The measurement program was documented based on the Web, which made it easier to read, access and use since it contained links to other reports on the software processes, sheets, etc that might be required.
- However, there was still quite a lot of data that had to be manually collected and included in the excel sheet, which they found to be quite bothersome.

Regarding the use of MIS-PyME framework as compared with our previous experience:

- Questions asked to project managers are better focused thanks to the guidelines provided by the MIS-PYME indicator templates.
- Fitness to the purpose is more easily achieved thanks to the guidelines regarding the types of indicators and the restrictions in the implementation of each type of indicator.
- MIS-PYME indicator templates give guidelines regarding the type of analysis that can be performed on the indicator, as well as its possible outcomes and interpretation. This information allows the analyst to learn some information and pass it on to project managers as far as the potential of the results obtained from the indicators analysis is concerned.
- Guidelines regarding typical indicator inputs (how the indicator could be graphically displayed), and indicator examples make the definition easier, especially if the measurement analyst is not an expert in the measurement area.
- The integration of the measurement process into software processes is also easier using MIS-PyME, since it informs about how to do this.
- MIS-PyME documentation is Web-based, which facilitates its use (easy access to the required templates, etc.).
- The guide regarding how to document the measurement process helps to increase the reliability of the resulting measurement process, making it easier to use and more integrated as well.

However, there are still some deficiencies in MIS-PyME model. Among several mistakes found, the most outstanding problem was that MIS-PyME database, containing the indicators and measure examples, is still quite small and does not cover most of the needs.

5 Conclusions and Further Research

In this paper we have proposed a methodological framework which makes it easier to define measurement programs. The framework, which is called in Spanish MIS-PyME (Marco metodológico de definición de Indicadores de Software para PyMEs) is based on GQ(D)M [8, 9] and is designed to be used with software process improvement practices. It provides a full and detailed guide that helps define common required measurement goals based on indicator templates and a database that includes examples of indicators and measures that have been implemented in successful measurement programs. It also integrates a model to make the organization progress through measurement.

We started by showing the characteristics required for a measurement definition model suited to SMEs. We have shown how the most outstanding measurement models do not fulfill the requirements and, after reviewing MIS-PyME framework, we have given a number of reasons why MIS-PyME framework does match these requirements, thus proving the extent of our contribution.

The end of the paper presents the use of MIS-PyME for developing a measurement program in the development department of Sistemas Técnicos de Loterías del Estado (STL) where the results were positive.

In the future, we shall continue monitoring and refining this measurement framework in order to validate the MIS-PyME framework for different SMEs. We have to increase the MIS-PyME database and include other MIS-PyME measurement goals and related indicators to consider other needs. The maturity model integrated in MIS-PyME is the least developed area and we may focus on it in the years to come.

Acknowledgment

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

References

1. Brijjckers, A., Differding, C.: The Role of Software Process Modeling in Planning Industrial Measurement Programs. In: Proceedings of the Third International Symposium on Software Metrics (METRICS'96) pp. 31–40 (1996)
2. Daskalantonakis, M.K.: A Practical View of Software Measurement and Implementation Experiences Within Motorola. *IEEE Transactions on Software Engineering* 18(11), 998–1010 (1992)

3. Goldenson, D., Rout, T., Tuffley, A.: Measuring Performance Results in Small Settings: How do you do it and what matters most? In: Proceedings of the First International Research Workshop for Process Improvement in Small Settings, pp. 41–44 (2005)
4. Gresse, C., Punter, T., Anacleto, A.: Software measurement for small and medium enterprises. In: 7th International Conference on Empirical Assessment in Software Engineering (EASE). Keele, UK (2003)
5. Briand, L.C., Differding, C.M., Rombach, H.D.: Practical Guidelines for Measurement-Based Process Improvement. *Software Process - Improvement and Practice* 2(4), 253–280 (1996)
6. Mondragon, O.A.: Addressing Infrastructure Issues in Very Small Settings. In: Proceedings of the First International Research Workshop for Process Improvement in Small Settings, pp. 23–29 (2005)
7. Emam, K.E.: A Multi-Method Evaluation of the Practices of Small Software Projects. In: Proceedings of the First International Research Workshop for Process Improvement in Small Settings (2005)
8. Park, R.E., Goethert, W.B., Florac, W.A.: *Goal-Driven Software Measurement-A Guidebook*: Carnegie Mellon University Pittsburgh: Software Engineering Institute (1996)
9. Goethert, W., Sivity, J.: Applications of the Indicator Template for Measurement and Analysis, in *Software Engineering Measurement and Analysis Initiative* (September 2004)
10. Solingen, R.v., Berghout, E.: *The Goal/Question/Metric Method - A practical guide for Quality Improvement of Software Development*. Mc Graw Hill, New York (1999)
11. Solingen, R.v., Berghout, E.: Integrating Goal-Oriented Measurement in Industrial Software Engineering: Industrial Experiences with and Additions to the Goal/Question/Metric Method (GQM). In: *Seventh International Software Metrics Symposium*. London, England, pp. 246–258 (2001)
12. Shepperd, M.: *Foundations of Software Measurement*. Prentice Hall, Hemel Hempstead, England (1995)
13. PSM: *Practical Software and Systems Measurement - A Foundation for Objective Project Management Version 4.0c*: Department of Defense and US Army (November 2000)
14. ISO/IEC 15939, in *Software Engineering - Software Measurement Process* (2002)
15. Mendonça, M.G., et al.: An approach to improving existing measurement frameworks. *IBM Systems Journal* 37(4) (1998)
16. Basili, V.R., Caldiera, G., Rombach, H.D.: The experience factory, in *Encyclopedia of Software Engineering*, J.J.M. (ed.) John Wiley & Sons, pp. 469–476
17. Druffel, E., Redwine, S.T., Riddle, W.E.: The STARS Program: Overview and Rationale, pp. 21–29. *IEEE Computer*, Los Alamitos (1983)
18. Guerrieri.: Searching for Reusable Software Components with the RAPID Center Library System. In: *Proceedings of the Sixth National Conference on Ada Technology*, pp. 395–406 (1988)

Smart Technologies in Software Life Cycle

Zane Bičevska and Jānis Bičevskis

Datorikas Instituts DIVI, A.Kalnina str. 2-7, Riga, Latvia
University of Latvia, Raina blvd. 19, Riga, Latvia
Zane.Bicevska@di.lv, Janis.Bicevskis@lu.lv

Abstract. In software life cycle models traditionally the main attention is being paid to the software development, including requirement gathering (specification), design, implementation and testing. Less research is devoted to the system maintenance and operation despite the fact that these aspects take up the main part of the duration of a successful system. In the paper smart technologies are being analysed – architectural designs and software components which using meta information on system and its usage conditions are able to solve efficiently the problems of maintenance and usage: data quality and performance monitoring, software flexibility and testability, context dependant user interface. The advantages of smart technology usage are pointed out helping to improve software maintenance and operation processes.

Keywords: Smart technologies, Testing, Maintenance, Life Cycle models.

1 Introduction

The driver of nowadays IT industry is the speed and search for new technologies. It has a strong impact on the process of application development: on the one hand, the high competition in the market requests a high quality of products, and on the other hand it restricts the time resources and capacity available for quality assurance. Moreover, not only developers of development platforms and standard products are faced with this dilemma but also the big community of individual and specific IT solutions developers.

In particular developers of individual software are in a never ending loop – the rapidly developing and heterogeneous environment (hardware, infrastructure, software, data quality etc.) impacts the demand of individual IT solutions substantially. Therefore the IT solutions are subject for further changes and it leads to non-homogenous software that is hard to manage and to distribute.

The proposed way is to develop a „smart” software that like human beings would be able to deal with unknown environment and adequately react on unexpected events. Though development of smart software can take additional resources, it will pay off in phases of software maintenance and operation. Authors propose such software development that includes both the base functionality of information system

and additional features (services) for a better maintaining of the software. These additional features, like scaffolding built in the construction process of a building, are created in the process of software design and implementing. But unlike the building process the “scaffolding” of an information system is never taken down, it stays in the information system for its whole life time.

Conditionally, features for smart technology compatible software can be divided into two big groups as follows:

1. External stability, when SW avoids performance incidents caused by impacts of other system components, for instance, SW operation environments, checking, security monitoring, availability monitoring and processing of other external events
2. Internal stability, when SW avoids or at least limits impact of internal faults, for instance, control of SW performance correctness in production (self-tests of core functionality), automatic download of new versions and converting of collected data and others.

The targets set in the *self-adaptive software* researches [1, 2] partially overlap with the statements stated by the smart technologies. Self-adaptive SW researchers are focusing on SW ability to adapt to implementation environment and external conditions thus allowing even code translation to the targeted environment and defining rules for SW reaction on external events. This report sets different targets: troubleshooting SW exploitation failures by applying automatic indication of possible failures and reporting them to staff. Implementation of this approach is more beneficial and convenient to use in practice.

Hereinafter the term of software being compatible with principles of smart technology will be denoted as STSW.

2 Software Life Cycle Models and STSW

Since costs of hardware and infrastructure necessary for creating and operation of IT solutions are decreasing rapidly, the main part of producing and maintaining costs are those for software. Unfortunately the producing of high quality software up to users' requirements, secure and easy to maintain is still just a dream of software developers. The real problem is the lack of productive communication between customers and developers. The customers being not well-informed about specifics of IT are not able to formulate the system requirements clearly and precisely enough in order to use them in the development of an application. There is no common communication “language” understandable for both specialists and non-IT specialists that could serve as a precise medium for defining the system requirements. The quite popular among IT specialists Unified Modelling Language UML [3], is only very rarely suitable in an environment being not very familiar with IT terminology.

Nevertheless the main difficulty which software developers are faced to is not a formulating of exact requirements but especially the changing requirements. Software developers have to create software according to inaccurate and changing in the time requirements. There are different life cycle models [4] traditionally applied to solve the problem – linear models demanding a stepping back to the previous lifecycle phases in case of nonconformity between requirements and the IT solution and incremental models requiring the defining of requirements and implementing of them gradually.

More effective than models described before are so-called prototyping models when the development process is divided into numerous steps each of which are finished with an intensive communication with a customer to see whether the results fit to the desirable. The prototype as a communication “language” between customer and developer seems to be very effective in sense of harmonization of requirements and possibilities but it is a quite time-consuming process that is not suitable for development projects with limited budgets and fixed deadlines.

The existing software life cycle models are mainly focused on the creating process of software which is just the first step in the life of every software solution and definitely not the longest one. Even very qualitative software can become out-of-date in a very short period of time because of changing requirements and appearing of new technologies.

This article analyses the STSW ideas that should ensure continued usage of software in spite of changes.

3 The Principles of Smart Technology

The smart technology is based on the idea about “self-managing” software, respectively software able to control of internal and external factors and accordingly reacting on them. This report discusses currently the most popular components of the smart technologies, admitting that their number might significantly increase in the future.

- 1) Automatic updating of versions, ability to provide remote reporting on missing components and actual status
- 2) Analysis of external environment, ability to check external environment and adapt to it
- 3) Self-testing, dynamic tracing of control flow, events and values
- 4) Incorporation of business model into software
- 5) Data quality control
- 6) Performance monitoring
- 7) Security (confidentiality) monitoring
- 8) Availability monitoring

Detailed description of smart technologies features follows.
Smart technology overview is given in the following figure.

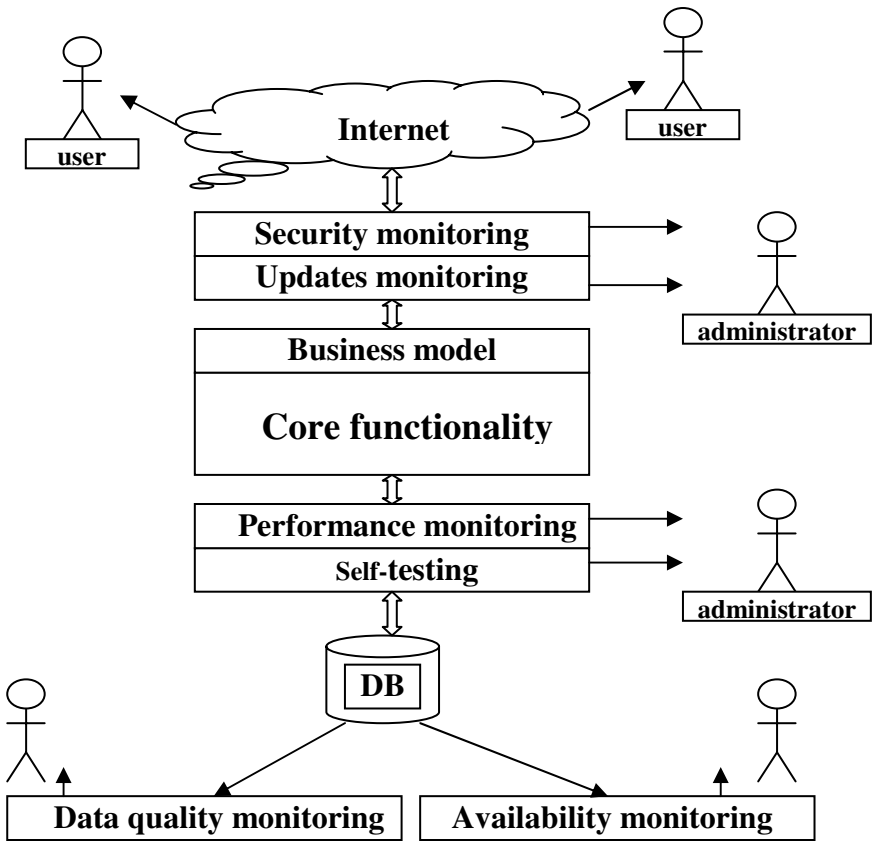


Fig. 1. Fundamental scheme of smart technology

3.1 Automatic Updating of Versions

Problem Identification

The first application of smart technologies is related to the field of delivering and installing (distributing) of software. Information system equipped with smart technology is able to analyse the environment which it is put into from viewpoint of standard and specific parameters. As standard parameters are supposed to be for example the operational system, the data base management system, browsers etc. used on the concrete server or workstation.

Specific parameters are check of evidence of previous (possible damaged) software versions as well as evidence of other specific solutions on the workstation. Typical example of a dangerous software- “neighbour” are antivirus solutions which can classify smart technology software as a virus and even block it up.

Similarly as for installation of a new version SW users should have a reverse link to SW developer. Due to this reverse link developers can receive complete information on

performance, including failure reports and statistics on activities that would allow developers to improve SW quality. As a rule including of the above mentioned features into SW requires components that are functioning throughout the whole life cycle of SW and are considered as extension of core functionality that sometimes remaining unaware to client.

Implementation

Currently all leading SW developers provide components that allow automatic loading of update packages that after authorization by administrator are installed in production. Similarly, special components allow sending reports on incidents that have occurred during system operation. Unfortunately, only a very few systems provide converting of the collected data to a new version and automated installation of completely new versions. Moreover, simultaneous usage of different version can cause hardly predictable consequences.

3.2 Analysis of External Environment

Problem Identification

Software ability to analyze features of the external environment and adapt itself to these features is called external stability. Frequently, these problems are researched separately “*adaptive software*” and “*self-adaptive software*”. There are two approaches used in the praxis. In the first case the supplier of software dictates requirements for external environment should be assured to guarantee the operation of delivered software. This principle is consequently applied in the distribution of software (standard) products when minimal basic requirements for hardware, operating systems etc. are defined. This approach is not very effective in case of collisions with other IT solutions or older versions of the same software, or other factors. Another problem of this approach is limited access to the market if consumers are not ready or able to satisfy the postulated minimal requirements.

The other way is to improve the software assuring the possibility to use the software in very many different environments and their combinations. One of the very popular keywords in this field is a platform independence which assures the usage of software in various operating systems, on different data base management systems, using various browsers etc. In fact there is a choice for a developer of “universal” software between two ways. The first one is to develop a solution for every possible (desired) combination of environments. The other one is to use only those technical methods or development tools that cover a subset of possibilities supported by all chosen target platforms. A typical example of the second way is a usage of a universal subset of SQL requests to ensure the correct functioning of data requests in various data base platforms. Apart from the chosen way this approach is expensive because of disproportion between the potential benefit (satisfied customer) and the necessary investment (very wide technical functionality).

Nevertheless the platform independence does not solve the problem of software installing – there is still a big amount of time resources and efforts spent for analysing of collision causes, for consultations and for improving of software to ensure the adequacy of it to the specific environment.

Implementation

STSW bases on the idea from the real world which shows the ability of living beings to adjust themselves to specific conditions.

In a similar way the software should be able to analyze versions and configurations of the operating systems and the data base management systems in accordance with the requirements of the application. In effect this check is much more complicated, because the version number and some of the configuration only partly determine the ability of the given application to fulfill their functions. We propose the following Smart software development model. A Smart software application requirement passport is created during the software development process in which the requirements against the environment are fixed: for operating system and other of its components, detailed on level of object classes, of DLL's and of others - .ini-files, registry entries, location of files and folders, regional and language settings, workstation settings etc. The creation of the passport is an obligation for the developer; it happens by generation of the passport from the development environment in which the application is created and is able to work. The created passport is integrated into the SW. After implementation of the application in the production environment a module prepared by the developer compares the production environment parameters with the passport parameters. In the case of differences, without starting the production of the application, the user is notified about the differences and, if possible, the correct environment configuration is prepared. Our experience [5, 6] shows, that in the case of distributed systems (also for WEB applications) differences between the requirements of the application and the environment are important and therefore it is necessary to use STSW to eliminate the problem.

3.3 Self-testing

Problem Identification

The next feature of STSW also comes from the organic nature; it is a self-testing – ability to control itself and understand the limits of own possibilities.

The STSW should be able to test itself before running in operation. Hardware tests which are running every time the hardware is switched on are a common thing. Unfortunately it is rather unusual to see this approach in software solutions.

Implementation

The self-testing could be done in the following way. There is set of core requirements identified by customer; these are functions which are essential for using of software. The core functionality can include basis functionality - for instance calculations, workflows – but should not include requirements of lower importance – navigation, comfort etc. developers create multifunctional tests that include the testing of a core functions und incorporate (integrate) them into the software. In the application are included a function of automated running of the incorporated tests and comparing of the results with benchmark values.

The self-testing activities can be run before the real operating of a system. It would guarantee integrity of the system in a real production. There is a difference between testing of a system as a part of quality assurance in the development process and self-testing as a part of real operating of system in a production. The aim of self-tests is to discover possible integrity problems in the production environment, and the self-tests should be run and used all the time system is used in a production. Considering that implementation mechanisms of self-testing provide additional tracing for control flow, events and values. These tracking possibilities are necessary for the programmers during unit testing.

Of course the self-testing feature demands additional efforts in the software development process. Nevertheless it is proven in the praxis [7] that this approach can substantially improve quality of software.

3.4 Incorporation of Business Model into Software

Problem Identification

Information systems are built to support different business processes. It means the primary business process is to perform business processes. Thus the most important domain specific concepts are related to view of users. The IT solution must be able to deliver information in domain specific terms understandable for users. This requirement is often considered in messaging systems of different software solutions. But STSW should enlarge the set of domain specific messages available in a system adding an information about status of the system, offering a context-sensitive help and statistics about events in the production environment still using domain specific terms.

Implementation

The proposed feature can be implemented in STSW by incorporating of business process models into the software, by linking of software processes to business processes. It can be achieved using an instruments (procedure calls) of a system that ensures coincidence of business processes and software objects. In this case the business process model serves as meta-information for description of software functions.

The described approach has been partially implemented in the [8] solution that includes messaging system and user manual according to business processes. It was achieved due to the technical architecture of the solution which based on an interpreter being able to interpret descriptions of business processes and act accordingly to them.

3.5 Control of Data Quality

Problem Identification

A very important component of STSW is delivering of the diagnosed information to developers. The most primitive way to inform users and developers about accidents in the production is a users' activities log which is a part of every modern information system.

STSW requires not only ability to register and collect information (being passive) but also a possibility to compare the information with quality requirements (meta-information), to measure it and distribute it to the appropriate staff (being active).

In the case of very developed systems there should be a feature for trying to solve occurred problems of systems by itself. It relates with activities of assuring of external stability of system that are able to find and install the missing components and drivers.

The special role in the production plays the quality of data in the data base. The data model (ER model) just partially defines limitations and dependencies among data elements. The detailed control of inputs is usually supported by software procedures but they can also be incomplete.

Additional difficulties defy if there are events having time gaps between events and the fixation of the events in the data base. In these cases input of inconsistent and low quality data is possible.

Implementation

The solution of the problem could be an including of a special independent component (smart component) into the IT solution that regularly upon staff request ensures the conformity of data in the data base and consistency rules.

Doubtless, the attributes of data quality and measuring of them is specific for every application [9]. But the general theory [10] discusses us how to define and measure the quality of collected data. The practical experience shows that autonomous data quality monitoring have been implemented only in a few IT solutions.

3.6 Performance Monitoring

Problem Identification

Performance analysis has become especially important since portals and Web applications are widely used. The essence of the problem is the inability of the portal or Web application developers to forecast with the sufficient precision the user request intensity and request types. Thus the developers can not identify the “bottlenecks” during the development. The approach to the problem solving, partly, is to use generators (11) of Web page requests, being able to deliver to the portal a request flow defined in advance. Measuring the performance parameters at different loads the developers are trying to forecast the performance of the Web application in real production environment. Unfortunately the measurements are taken in a test environment not in a real one and the real flow of requests is substituted with a simplified one.

Implementation

STSW offers such solution to the problem: developers in due time insert the means of measurement of load and time parameters into the application. During everyday usage these means are in an inactive state. The means can be activated by the system maintenance staff in appropriate situations thus getting precise information about “bottlenecks” of the system. As an additional feature STSW performance monitoring can contain the notification of maintenance staff on reaching the critical values of system reaction time.

3.7 Security Monitoring

Problem Identification

Security – possibility to use a system only by destined users – is one of the most critical IT requirements. Usually the ensuring of IT solution's security is limited by usage of users' enrolment and authorising mechanisms. But it is quite simplified approach as the responsibility for the security is delegated to users of the system. It is well-known that security is impacted not only by users but also by other factors – incompleteness of operating systems and programming environments, errors of infrastructure and inadequate operating of a system – most of which are unknown for users

The idea of smart technologies is to perform, at least partially, additional activities for ensuring of security. STSW can support the management of passwords and user information by accepting only very secure (not guessable) passwords and regular warning about necessity to change password and to control the uniqueness of it. Additionally STSW could inform the appropriate employee and users about external attempts to access a system or to read sensitive data. These features can be relatively easy implemented due to the agents technologies.

Implementation

All features mentioned in this chapter are successfully implemented in the [7] solution that lets ascertain for any mobile phone number which mobile phone operator it belongs to. Everybody can use the service and have the information about one number per request but nobody can retrieve the whole list of mobile phone numbers belonging to one operator. STSW can detect the attempts to create complete lists of numbers belonging to one operator and send warning messages to responsible persons. STSW should be also protected against automated “scanning” of passwords.

3.8 Availability Monitoring

Problem Identification

Similarly as in previous case access to the system, respectively use of the system when it is necessary, is the most crucial criteria (request) for the IS. These requests are met by establishing appropriate infrastructure usually using double power supply, Internet access and double servers. However, this is an expensive way to achieve the accessibility and, moreover, it does not provide the aimed result. The remaining problem is that working online the integrity of database should be ensured in case one of the servers is down.

Implementation

More up-to-date solutions are provided by load-balancing methods [12]. According to this solution remote servers and power supply clusters are established in order to ensure auto switching to other servers in case one of the servers is down. However, mechanic switching among servers and notification of personnel on failure via short messaging still does not resolve more complicated problem – data synchronization on simultaneously operating servers in case the servers has run down.

Thus, more additional efforts and customized approach are requested in each particular situation. Doubtless, such possibilities should be foreseen during designing

of the system architecture and should be included into STSW, and can be carried out via agent technologies.

Similarly as in previous case monitoring of availability has been carried out [7].

4 Conclusions

Software conforming to the principles of Smart technology offers a number of advantages compared to the currently used platform independent configuration-fixed solutions:

1. Smart technologies have a number of advantages for information system maintenance and development. They have an important impact on software life cycle model. They are usable in incremental models, less useful in linear models.
2. Adding of smart technologies to the software after the implementation is useful for the external stability support. Internal stability support is achieved including smart technology already in the software architecture design phase – later inclusion is expensive and time-consuming.
3. Implementation of smart technology principle in software takes fewer resources than full-range configuration support. In the same time smart technology places fewer constraints on the acceptable means of expression.
4. Data quality control mechanisms have to be created at the metalevel and separated from the other business logic. In such a manner it is possible to achieve high reusability and openness to the changes of these mechanisms.
5. Smart technologies allow reducing the efforts for software testing and setting up, thus increasing the client service level.
6. Smart technologies assist to provide software performance in an environment containing heterogeneous platforms and infrastructure.

References

- [1] Laddaga, R., Robertson, P.: Self Adaptive Software: A Position Paper. In: International workshop on Self Adaptive Software Properties in Complex Information systems, 2004, Bertinoro, Italy (2004)
- [2] Wang, Q.: Towards aRule Model for Self-adaptive Software ACM SIGSOFT Software Engineering Notes, vol. 30(1) (January 2005)
- [3] Arlow, J., Neustadt, I.: UML and the Unified Process. Addison-Wesley, London (2002)
- [4] Pressman, R.S.: Software Engineering. A Practitioner's Approach, 6th edn. McGrawHill, New York (2005)
- [5] Andzans, A., Mikelsons, J., Medvedis, I., et al.: ICT in Latvian Educational System - LIIS Approach. In: Proceedings of The 3rd International Conference on Education and Information Systems: Technologies and Applications, July 14-17, 2005 Orlando, Florida, USA (2005)
- [6] Latvian Education Informatization System – LIIS [on-line]. Available on internet: <http://www.liis.lv>
- [7] Available on internet: <http://www.numuri.lv>
- [8] Available on internet: <http://www.liaa.gov.lv>

- [9] Loshin, D.: Enterprise Knowledge Management: The Data Quality Approach. Morgan Kaufman, Seattle (2001)
- [10] Wang, Richard, Y., Ziad, Mostapha, Lee, Y.W.: Data Quality. Kluwer Academic Publishers, Boston (2000)
- [11] Available on internet: <http://www.mercury.com/us/products/performance-center/loadrunner/>
- [12] Shimonski, R.J.: Windows Server 2003 Clustering & Load Balancing, Osborne McGraw-Hill, ISBN 0-07-222622-6

Convertibility Between IFPUG and COSMIC Functional Size Measurements

J.J. Cuadrado-Gallego¹, D.Rodríguez¹, F. Machado², and A. Abran³

¹ Department of Computer Science
University of Alcalá
28805 Alcalá de Henares Madrid, Spain
{jjcg,daniel.rodriguez}@uah.es

² Universidad Católica del Uruguay
Av. 8 de Octubre 2738
11600 Montevideo, Uruguay
fmachado@ucu.edu.uy

³ École de Technologie Supérieure
1100, rue Notre-Dame Ouest
Montréal, Québec
Canada H3C 1K3
alain.abran@ele.etsmtl.ca

Abstract. Since 1984 the International Function Point Users Group (IFPUG) has produced and maintained a set of standards and technical documents about a functional size measurement methods, known as IFPUG, based on Albrecht Function Points. On the other hand, in 1998, the Common Software Measurement International Consortium (COSMIC) proposed an improved measurement method known as Full Function Points (FFP). Both the IFPUG and the COSMIC methods both measure functional size of software, but produce different results. In this paper, we propose a model to convert functional size measures obtained with the IFPUG method to the corresponding COSMIC measures. We also present the validation of the model using 33 software projects measured with both methods. This approach may be beneficial to companies using both methods or migrating to COSMIC such that past data in IFPUG can be considered for future estimates using COSMIC and as a validation procedure.

Keywords: Functional Size measurement, IFPUG, COSMIC, Software Estimation.

1 Introduction

Function Point Analysis or FPA is one the oldest and most widely used software functional size measurement method. It was proposed by Albrecht and his colleagues at IBM in 1979. Since 1984 this method is promoted by the International Function Point Users Group (IFPUG) [7]. In 1994, the International Organization for Standardization (ISO) set up a working group to establish an

international standard for functional size measurement. This group did not produce a measurement standard, but a set of standards and technical documents about functional size measurement methods, known as the ISO/IEC 14143 series [1,2,3,4,5]. The FPA method became the standard ISO/IEC 20926 [11] in 2003, its unadjusted portion being compliant with the ISO/IEC 14143 [1]. Starting in 1998, a set of experts in software measurement created the Common Software Measurement International Consortium or COSMIC, and proposed an improved measurement method known as Full Function Points (COSMIC FFP) [6]. This method became the standard ISO/IEC 19761 in 2003 and is also ISO/IEC 14143 compliant. Both IFPUG and the COSMIC-FPP methods measure functional size of software, but produce different results. For this work, we briefly compare IFPUG and COSMIC definitions and propose a model to convert functional size measures obtained with the IFPUG method to the corresponding COSMIC FFP measures. To do so, we have used a repository of 33 projects measured using both methods.

The organisation of the paper is as follows. Section 2 provides a high level view of the mapping between both methods. Section 3 presents and analyses our approach and its empirical validation. Finally, Section 4 concludes the paper and future work is outlined.

2 Analysis of Correspondence Between Definitions

This section presents a very high level view of the components and relationships for IFPUG and COSMIC measurement methods needed to obtain correspondences between the concepts defined by such components and relationships to determine under which conditions it would make sense to compare the measurements obtained with both methods.

There are three initial concepts in the measurement of software functionality size shared for both methods: the purpose of a measurement, the scope of a measurement and the application boundary. Such concepts define what is measured and what it is measured for. It is possible to have a mapping between both methods for the key terms: (i) the *purpose of a measurement*; (ii) the *scope of a measurement* and (iii) the definition of *boundary*. The same happens with other key concepts in the software functional size measurement that must be considered; three are related to data (the *object of interest* or *entity*, the *data group* or *file* and the *data attribute* or *data elements*) and two to its transformation processes (the *functional process* or *transactional function*). Table 1 summarizes the correspondence of concepts between COSMIC and IFPUG.

After analyzing both methods, it can be concluded that: (i) the software functional size measures obtained shall be comparable when the purpose and the scope of the measurement coincide, as well as the application boundary; obviously, the application to be measured also has to be the same. These concepts are practically identical in both methods; (ii) both methods coincide when they divide the user data processing requirements into units, using practically the

Table 1. Correspondence of concepts between COSMIC and IFPUG

<i>COSMIC</i>	<i>IFPUG</i>
Purpose of a measurement	Purpose of the count
Scope of a measurement	Scope of the count
Boundary	Application boundary
User	User
Object of interest	Entity
Data group	File
Data attribute	Data elements
Functional process	Transactional function

same criterion. Consequently, functional processes in COSMIC will be transactional functions in IFPUG and vice versa; and (iii) both methods also coincide in grouping data sets using practically the same criterion. Consequently, data groups in COSMIC will correspond to files in IFPUG and vice versa.

3 Conversion Rule Proposed

There are several situations in which it is possible to know reasonably the resulting data movements for each object of interest. Some of those objects of interest will correspond to Internal Logical Files (ILF) or External Interface Files (EIF), according to the equivalences established so that it is possible to express the number of data movements according to the number of File Types Referenced (FTR) between IFPUG and COSMIC. There is a data movement for each FTR in the External Outputs where an object of interest is deleted: the application writes when deleting the corresponding data group. In this case, there is usually an error or confirmation message. Preliminarily, we could generalize that the minimum number of data movements in an elementary process is equal to the number of FTRs adding one:

$$CFSU_{MIN} = FTR + 1 \quad (1)$$

where $CFSU_{MIN}$ (COSMIC Function Size Unit) is the minimum size of the functional process measured in COSMIC and FTR is the number of File Type Referenced in IFPUG. However, in COSMIC, the minimum number of data movements in a functional process is 2 CFSU. When the number of FTR is zero, Eq. (1) only returns 1 data movement. We need to reformulate Eq. (1) to consider this case:

$$CFSU_{MIN} = Max(2, FTR + 1) \quad (2)$$

The theoretical maximum of data movements cannot be determined from the number of FTRs. Even if there are four data movements at the most for each file type referenced, there could be other data movements that will not imply persistent data groups; for example, commands, parameters, etc. However,

from the above analysis it is possible to assume the maximum number of data movements according to the elementary process type:

- In external inputs (EI), there are usually no more than two data movements in the same functional process for each file type referenced: there are two data movements, one for the input and the other for the writing of the data of an object of interest.
- In external outputs (EO) and External Queries (EQ), where data of an object of interest are read and shown, neither is there usually more than two data movements in the same functional process for each file type referenced, one for reading and one for the output of the object of interest.
- In all elementary processes there is usually an error or confirmation message.
- In EO, generally, there is also an output of data created during the elementary process, or an entry command or parameter.

We could generalize that the maximum number of data movements in an elementary process is equal to the double number of file types referenced plus one, for external entries and inquiries, and plus one for external outputs. Considering that the number of file types referenced could be zero, and that the size measured in COSMIC cannot be lower than 2, the above is expressed as follows:

$$CFSU_{MAXEI/EQ} = \text{Max}(2, 2 \cdot FTR + 1) \quad (3)$$

$$CFSU_{MAXEO} = 2 \cdot FTR + 2 \quad (4)$$

where $CFSU_{MAXEI/EQ}$ is the maximum size of the external input or external queries functional process measured in COSMIC, $CFSU_{MAXEO}$ is the maximum size of the external output functional process measured in COSMIC and as before FTR is the number of file types referenced.

In short, given the measurement of an application with IFPUG, of which the number of transactional functions and the number of FTRs in such functions are known, we propose as a hypothesis that such application will have a COSMIC size within the interval given by the following equation:

$$\begin{aligned} & \sum_{i=1}^{EI} \text{Max}(2, FTR_i + 1) + \sum_{i=1}^{EO} \text{Max}(2, FTR_i + 1) + \sum_{i=1}^{EQ} \text{Max}(2, FTR_i + 1) \\ & \leq CFSU \leq \\ & \sum_{i=1}^{EI} \text{Max}(2, 2 \cdot FTR + 1) + \sum_{i=1}^{EO} \text{Max}(2, 2 \cdot FTR + 1) + \sum_{i=1}^{EQ} \text{Max}(2, 2 \cdot FTR + 1) \end{aligned} \quad (5)$$

4 Experimental Validation of the Conversion Rules

The data used in this qualitative analysis come from 33 software applications, measured with IFPUG version 4.1 and COSMIC version 2.2. Out of these 33 software applications, one is a case study documented by IFPUG [7]. Data from IFPUG were taken as such and only the measurement with COSMIC was carried out. Another application is a case study provided by IBM Rational as example RUP [8]; this application was already measured with COSMIC and only the

measurement with IFPUG was performed. Another application is a case study described in Fetcke [9]; the data from IFPUG and COSMIC were obtained as such from a case study used to compare different software measurement methods. The remaining 30 applications were final projects of students attending the Software Engineering course at the University of Alcalá, Madrid, Spain. These software development projects included the description of the application and the measurements with both methods. These measures were obtained by a team of three junior measurers, which later were verified by another senior measurer and finally by the authors. Some projects from the Software Engineering courses at the University of Alcalá were discarded when the description of the application did not enable the validation of the measures obtained. The differences in the measures were generally due to different interpretations of user requirements and furthermore, rules of IFPUG and COSMIC methods are stated in a natural language and thus, subject to ambiguity and interpretation. However, all differences were exhaustively revised and reconciled.

The intervals in our set of measures vary between 78 and 462 function points, with a mean of 291.2 function points, and a standard deviation of 98.6 function points. The summary of the results of the measurement appears on Table 2. IFPUG is the size measured in function points without adjustments with IFPUG 4.1; ILF+EIF is the number of data functions, internal logical files plus external interface files in each project; EI+EO+EQ are the number of transactional functions in each project, external inputs plus external outputs plus external inquiries; FTR is the total number of file types referenced in all functional processes of each project. Lastly, COSMIC is the functional size measured in COSMIC units.

We used two complementary techniques for our experimental research (i) the direct verification on a relatively large number of cases, where we evaluated our hypothesis; and (ii) statistical analysis to generalize the findings. The first technique consists of evaluating a model in a relatively large set of cases and confirming that the expression in Eq. (5) is always verified. In each case, the same software application is measured both with the IFPUG method and with the COSMIC method. If in any of such cases, the expression in Eq. (5) is not verified, we would be able to affirm that the corresponding model does not enable any conclusion regarding the size of an application measured with COSMIC from the intermediate measures resulting from the measurement of such application with COSMIC. On the contrary, if the expression in Eq. (5) is verified in all cases, we will be able to state that the corresponding model adequately describes the cases considered, but we will not be able to make general statements for other applications not included in the cases considered.

We now describe the statistical analysis. To do so, we defined two random variables: one as the difference of the value measured with COSMIC and the minimum value given by Eq. (5), and the other as the difference between the maximum given by Eq. (5) and the value measured with COSMIC. The former represents the distance between the lower extreme of the range and the value measured with COSMIC, while the latter represents the distance between the

Table 2. Project Measurement Results

<i>Proj ID</i>	<i>IFPUG</i>	<i>ILF+EIF</i>	<i>EI+EO+EQ</i>	<i>FTR</i>	<i>COSMIC</i>
1	95	5	16	27	68
2	126	10	14	37	80
3	78	3	16	27	72
4	329	25	44	71	177
5	340	14	72	108	195
6	324	6	82	87	267
7	177	9	33	33	108
8	381	12	65	163	278
9	360	12	62	139	210
10	286	14	46	58	191
11	462	14	65	169	286
12	283	7	53	122	263
13	109	5	21	21	65
14	432	19	79	149	294
15	326	12	74	91	200
16	331	13	62	84	234
17	236	9	42	88	158
18	324	10	62	132	297
19	311	6	63	126	310
20	346	14	63	91	263
21	410	19	88	88	215
22	395	14	84	97	279
23	279	14	52	65	166
24	324	13	61	91	224
25	412	19	64	163	248
26	315	11	66	123	313
27	157	9	20	107	215
28	307	14	45	155	264
29	167	8	22	89	125
30	299	11	54	111	267
31	269	19	39	66	144
32	299	12	57	114	277
33	320	15	47	103	155

value measured with COSMIC and the upper extreme of the range. To accept statistically that the value measured with COSMIC is always within the interval, both variables must have a known distribution with positive mean. The second and fourth columns in Table 3 show the maximum and the minimum given by Eq. (5) in relation to the COSMIC measure.

The significance level chosen for these statistical tests is 98%, corresponding to $\alpha = 0.02$, because, as the affirmations about the variables are independent among them, the significance level resulting from the combination of both will be equal to $98\%^2 \geq 95\%$, corresponding to $\alpha = 0.05$.

The first step in the statistical analysis is to characterize these random variables, calculating some of their descriptive statistics and determining their

Table 3. Measurements Minimum and Maximum calculated according to the Model

<i>Proj ID</i>	<i>Minimum CFSU</i>	<i>Maximun</i>	<i>D ↓</i>	<i>D ↑</i>
1	43	68	73	25 5
2	51	80	88	29 8
3	43	72	73	29 1
4	115	177	198	62 21
5	180	195	301	15 106
6	169	267	278	98 11
7	66	108	114	42 6
8	228	278	403	50 125
9	201	210	352	9 142
10	112	191	200	79 9
11	208	286	357	78 71
12	175	263	312	88 49
13	42	65	68	23 3
14	228	294	392	66 98
15	165	200	436	35 236
16	146	234	244	88 10
17	130	158	236	28 78
18	194	297	342	103 45
19	189	310	329	121 19
20	156	263	268	107 5
21	178	215	278	37 63
22	181	279	292	98 13
23	117	166	199	49 33
24	152	224	260	72 36
25	227	248	400	21 152
26	189	313	324	124 11
27	129	215	249	86 34
28	200	264	375	64 111
29	111	125	208	14 83
30	165	267	295	102 28
31	105	144	180	39 36
32	171	277	285	106 8
33	150	155	269	5 114

distributions with the respective distribution parameters. Table 4 shows descriptive statistics for variables $D \downarrow$ and $D \uparrow$.

As we can see in Table 4, the mean between both variables is positive. After several tests with different distributions, we found that both variables follow an exponential distribution, the first with $\lambda = 0.017$ and the second with $\lambda = 0.019$. The data adjustment with exponential distribution was performed using the Kolmogorov-Smirnov test [10]. In this test, the null hypothesis H_0 is that variables follow an exponential distribution; the alternative hypothesis H_A is that they do not follow an exponential distribution. The test results for both variables appear in Table 5.

Table 4. Statistics for the random variables $D \downarrow$ and $D \uparrow$

<i>Statistic</i>	$D \downarrow$	$D \uparrow$
<i>Mean</i>	60.36	53.64
\sum	35.68	55.75
σ^2	1272.99	3108.05
<i>Median</i>	62	34
<i>Min</i>	5	1
<i>Max</i>	124	236

Table 5. Kolmogorov-Smirnov test for variables $D \downarrow$ and $D \uparrow$

	$D \downarrow$	$D \uparrow$
D	0.173	0.148
p -value	0.254	0.431
α	0.02	0.02

We accept the null hypothesis H_0 that samples follow an exponential distribution, as the p -value calculated is higher than the significance level $\alpha = 0.02$ in both cases. The risk of rejecting the null hypothesis H_0 when it is true is 25.36% and 43.09% for the below and above differences, respectively. Figures 1 and 3 show the histograms for the variables $D \downarrow$ and $D \uparrow$ respectively. Figures 2 and 4 show the distributions accumulated for both variables. In these graphs it is possible to visually check the test results, in the sense that the adjustment in both cases is good, but it is better for variable $D \uparrow$.

The fact that both variables $D \downarrow$ and $D \uparrow$ have exponential distribution not only confirms our hypothesis, but further corroborates our hypothesis. On the one hand, it means that the probability of obtaining smaller differences

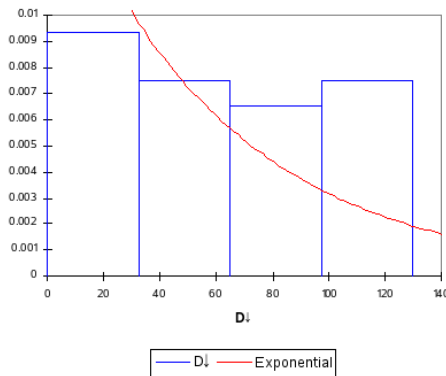


Fig. 1. Histogram for $D \downarrow$

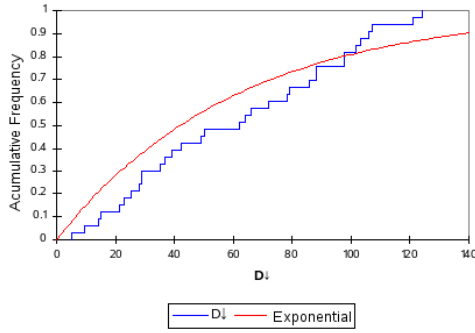


Fig. 2. Accumulative distribution for D_{\downarrow}

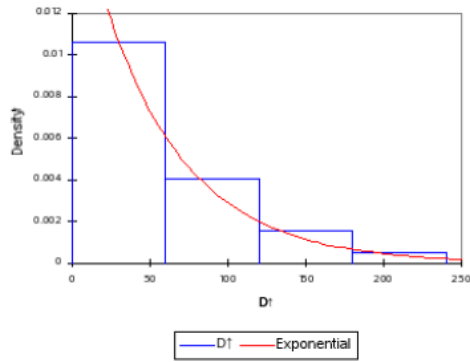


Fig. 3. Histogram for D_{\uparrow}

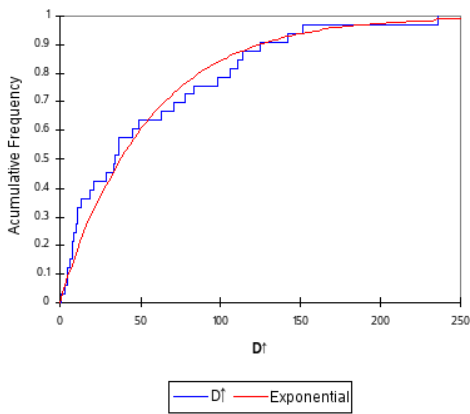


Fig. 4. Accumulative distribution for D_{\uparrow}

between measures and extremes is higher, and the probability of obtaining larger differences between measures and extremes is lower. Also, on the other hand, it also means that the distances are always positive, i.e., measures using COSMIC will never be outside the interval calculated according to our model.

5 Conclusions and Future Work

In this paper, we proposed a method to convert from IFPUG Function Points defined by the International Function Point Users Group (IFPUG) to COSMIC Full Function Points (COSMIC FFP) defined by the Common Software Measurement International Consortium (COSMIC). Although both methods produce different results, we have empirically shown an equation that limits interval of the conversion to be within a range. Such approach can be beneficial to companies using both methods or in the process of migrating to COSMIC such that past data measured using IFPUG can be considered for future estimates using COSMIC. Also, when organizations used both methods to improve their estimates, the approach of this paper can be used as an additional validation procedure.

Future work will consist in performing further case studies and validations within academia and industrial organizations.

Acknowledgements

We would like to thank the Spanish Ministry of Science and Technology for supporting this research (Project CICYT TIN2004-06689-C03).

References

1. ISO/IEC: Iso/iec 14143-1:1998 information technology – software measurement – functional size measurement — part 1: Definition of concepts. Technical report, International Standards Organization & International Electrotechnical Commission (1998)
2. ISO/IEC: Iso/iec 14143-2:2002 information technology — software measurement— functional size measurement — part 2: Conformity evaluation of software size measurement methods to iso/iec 14143-1:1998. Technical report, International Standards Organization & International Electrotechnical Commission (2002)
3. ISO/IEC: Iso/iec tr 14143-3:2003 information technology — software measurement — functional size measurement — part 3: Verification of functional size measurement methods. Technical report, International Standards Organization & International Electrotechnical Commission (2003)
4. ISO/IEC: Iso/iec tr 14143-4:2002 information technology — software measurement — functional size measurement — part 4: Reference model. Technical report, International Standards Organization & International Electrotechnical Commission (2002)
5. ISO/IEC: Iso/iec tr 14143-5:2004 information technology — software measurement — functional size measurement — part 5: Determination of functional domains for use with functional size measurement. Technical report, International Standards Organization & International Electrotechnical Commission (2004)

6. COSMIC: Cosmic measurement manual ver. 2.2. Technical report, Common Software Measurement International Consortium (2003)
7. IFPUG: Ifpug: Case study 1 release 3.0. Technical report, International Function Point Users Group (2005)
8. IBM: Course registration system. Technical report, IBM Rational (2004)
9. Fetcke, T.: The warehouse software portfolio: A case study in functional size measurement. Technical Report 1999-20, Technische Universitaet Berlin, Fachbereich Informatik (1999)
10. Montgomery, D., Ruger, G.: Applied Statistics and Probability for Engineers. John Wiley Sons, Inc, New York, USA (2003)

A Framework for Measuring and Evaluating Program Source Code Quality

Hironori Washizaki¹, Rieko Namiki², Tomoyuki Fukuoka², Yoko Harada²,
and Hiroyuki Watanabe²

¹ National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo, Japan
washizaki@nii.ac.jp

² Ogis-RI Co., Ltd., MS-Shibaura Bldg., 13-23, Shibaura 4, Minato-ku, Tokyo, Japan
{Namiki_Rieko,fukuoka_tomoyuki,Harada_Yoko,Watanabe}@ogis-ri.co.jp

Abstract. The effect of the quality of program source code on the cost of development and maintenance as well as on final system performance has resulted in a demand for technology that can measure and evaluate the quality with high precision. Many metrics have been proposed for measuring quality, but none have been able to provide a comprehensive evaluation, nor have they been used widely. We propose a practical framework which achieves effective measurement and evaluation of source code quality, solves many of the problems of earlier frameworks, and applies to programs in the C programming language. The framework consists of a comprehensive quality metrics suite, a technique for normalization of measured values, an aggregation tool which allows evaluation in arbitrary module units from the component level up to whole systems, a visualization tool for the evaluation of results, a tool for deriving rating levels, and a set of derived standard rating levels. By applying this framework to a collection of embedded programs experimentally, we verified that the framework can be used effectively to give quantitative evaluations of reliability, maintainability, reusability and portability of source code.

1 Introduction

In today's world, where value is controlled in every corner of society by software systems from the embedded to enterprise level, demand is increasing for a system of technology to measure and evaluate system quality characteristics (e.g. reliability) to use the evaluation results to maintain and improve the system. In this paper we propose a quality evaluation framework based on quantitative quality measures, for software engineers involved in development, maintenance or procurement of software, or others involved in improvement of development processes. We deal with quantitative measures of quality that take measurements of program source code written in the C programming language.

There is great demand for practical technologies which can measure and evaluate quality with high precision and identify quality characteristics that will cause problems or will need improvement, because the quality of the source code has a significant effect on the overall system performance and cost of development and maintenance. In the past, various techniques for measuring quality

have been proposed, but they generally have not covered quality characteristics comprehensively and the metrics or measured results have not been widely used [1].

In response, we propose a framework which applies to source code written in the C programming language and implements quality measurements and evaluation effectively. The framework is independent of any person/evaluator properties, and resolves the problems of conventional approaches.

2 Problems with Conventional Quality Measurements

From a quality point of view, measurement methods can be classified into four types based on amount of information. A *Metric* contains the least amount of information, and simply measures a particular property without relating it to quality. A *Quality Metric* measures a property and includes a way to interpret the measurement result in terms of a quality characteristics. *Quality Metrics* is used to refer to multiple such metrics for a single quality characteristics and a *Quality Metrics Suite* treats several quality metrics and systematically summarizes the results of each.

Though many metrics have been proposed, it is generally difficult to select an appropriate one from among them or to interpret the measurement results [2]. Further, they and measured values have not been broadly useful [1]. For quality measures which apply to source code in particular, the main problems are summarized below.

(P_1) Non-comprehensive suites: In order to take into account tradeoffs between different quality characteristics (e.g. time-behaviour vs. analysability), it is desirable to be able to measure and evaluate all quality characteristics, which effect the final system's quality in use, at the same time. However, most of the existing metrics which apply to source code do not relate measurement values to quality, or provide a quality metric which measures quality based on only a single characteristic. There are a few suites which handle source code, including REBOOT [3], QMOOD [4], SPC suite [5], the suite from Ortega [6], the EASE project result [7] and the ISO/IEC TR 9126-3 reference implementation [8]; however, they all require additional input besides the source code (e.g. a design model) and/or they lack comprehensive coverage of the measurable and assessable source code characteristics specified by the ISO9126-1 [9] (or equivalent) quality model.

(P_2) Lacking in ability to break-down or overall evaluation: Generally, source code written in a high-level programming language has a layered structure, with inclusion relationships between multiple logical and physical modules. For example, in the C programming language, generally functions are included in files, files in directories, and directories in the system, giving a four-layer structure. In this case, it is desirable to measure and evaluate the quality of individual module units according to various objectives, such as comparing the entire integrated system quality with another system and evaluating the quality

of individual small modules in order to identify problematic parts. However, no quality metrics suite has been proposed which can measure source code quality for arbitrary units from component up to the overall system.

(P_3) Difficulty in deriving standard rating levels: In order to evaluate quality from measurement results, rating levels which determine the allowable range of values for each type of measurement (i.e. thresholds), and assessment criteria which evaluate the results in units of quality characteristic and can combine them into an integrated result is required[10]. Generally, to derive rating levels, a set of samples are used, which are then divided into superior and inferior groups based on some criteria (e.g. a particular usage scenario[5] or qualitative evaluations[11,12]). The distributions of measured values in both groups are compared, and the upper and lower bounds of the range which statistically contains most of the measurements from the superior group are used as the thresholds. However, traditional techniques have required additional information, such as usage scenarios or qualitative evaluations, in addition to the source code and it has not always been easy to derive rating levels.

3 Proposed Framework for Quality Evaluation

We propose a practical framework which effectively measures and evaluates the quality of programs from source code written in the C programming language. The overall scheme, the solutions to the problems described above, and details of each of the elements of the framework are described below.

3.1 Overall Approach and Solutions to Problems

The structure of the framework is shown in Figure 1. It is made up of five elements: the quality metrics suite, an aggregation tool, a visualization tool, a rating levels derivation tool, and actual derived rating levels. Note that the framework does not include actual measurement tools. Rather, we make use of existing tools (e.g. QAC[13] and Logiscope[14]) which both apply to C source code and cover the metrics specified in the suite.

- Quality metrics suite: We extended the ISO9126-1 quality model to create a more comprehensive model, and built a collection of metrics together with associated quality characteristics based on this quality model by defining the interpretation of measurements from a quality standpoint. This resolved the problem P_1 of conventional approaches.
- Aggregation and visualization tools: We resolved the problem P_2 by implementing a technique for normalizing the results of each measurement in the suite to a value from 0 to 100 based on rating levels, and a mechanism for aggregating and summarizing module scores in step-wise gradations. The visualization tool transforms the collection of totaled scores into an evaluation report that is easier to understand intuitively.

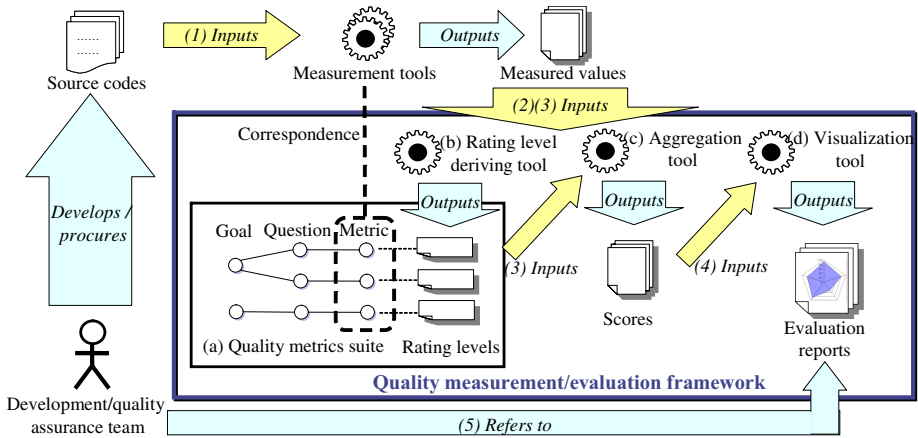


Fig. 1. Structure of the quality measurement/evaluation framework

- Rating level derivation tool and reference values: We resolved the problem P_3 by implementing a mechanism to derive rating levels statistically. The mechanism requires a set of source codes that are acceptable from a quality point of view. Then, by applying our metrics suite and the above three tools to several existing embedded software programs, we derived some actual rating levels and included them in the framework as reference values for software in the same domain.

The process flow for using the framework is shown below. Step (2) is not always required after rating levels have been derived for the target problem domain; however the rating levels should be continuously improved by iterating the process and accumulating measurement results, because the rating levels highly depend on the set of source codes used for the level derivation.

- (1) Measurements (measured values) are obtained by applying measurement tools that handle the metrics specified in the quality metrics suite to the source code being measured.
- (2) If rating levels are to be derived, this is done by applying the rating level derivation tool to measurements of source code that are acceptable from a quality point of view. If such codes are not available, the framework uses source codes which has been improved from a quality point of view, without any significant changes in functionality.
- (3) The measurements are entered into the aggregation tool to get the aggregate result of all of the scores. Internally, the aggregation tool uses the quality metrics suite as well as the derived rating levels.
- (4) An evaluation report is created by entering the aggregate results into the visualization tool.
- (5) The report is used to identify problematic parts or quality characteristics that need improvement and can be useful in resolving them.

3.2 Details of Structural Elements

The details of the elements and techniques that compose the framework are described below.

(a) Quality metrics suite

Using the ISO9126-1 general quality model as a starting point, we repeatedly interviewed several software professionals to narrow-down to the internal quality characteristics (static, not dynamic, qualities that can be measured and evaluated) of program source code that are not dependent on a particular programming language. The resulting quality model is shown in Figure 2. Note that functionality and usability have been excluded because they are difficult to measure using the source code only.

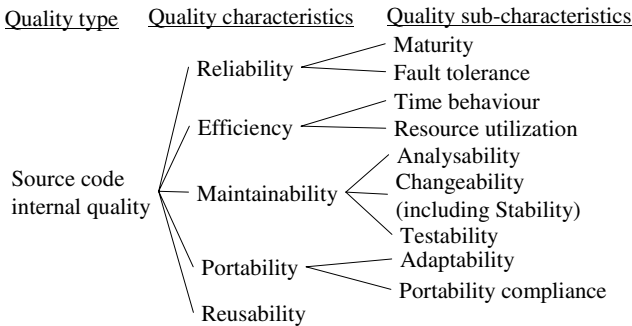


Fig. 2. Quality model

- Reliability: The ability to maintain specified performance levels when used under the specified conditions [9]. We take maturity and fault tolerance as sub-characteristics, while recoverability and reliability compliance (not the "reliability" itself) are excluded because they are difficult to measure and evaluate from the source code only.
- Efficiency: The ability to provide appropriate performance relative to the amount of resources consumed when used under clearly specified conditions [9]. Sub-characteristics are time behaviour and resource behaviour, while efficiency compliance has been excluded.
- Maintainability: The ease to which modifications can be made [9]. Sub-characteristics are analysability, changeability and testability, while maintainability compliance has been excluded. Also, to avoid duplication, the stability characteristic in ISO9126-1 is included under changeability.
- Portability: The ability to be transferred from one environment to another [9]. Sub-characteristics are adaptability and portability compliance, while installability, co-existence and replaceability have been excluded because they are difficult to measure and evaluate from the source code alone.

- Reusability: The extent to which a system or module-unit parts can be re-used in a different environment. This is not regulated in ISO9126-1, but considering its importance, particularly with respect to development efficiency within the same problem domain, we have added it as another quality characteristic separate from portability.

Next, we applied the Goal-Question-Metric (GQM) method [15], and assigned a metric to each quality sub-characteristic in the quality model. The GQM method is a goal-oriented method for mapping a goal to a metric by using a question which must be evaluated in order to determine whether the goal has been achieved or not. It is used within the framework to assign the metrics to the quality characteristics being evaluated.

We posed questions so that the evaluation could be made independently of the programming language of the source code being evaluated, and with the goals being to measure and evaluate each of the quality sub-characteristics. Finally, we narrowed down possible (programming-language dependent) metrics to those which could provide answers to the questions by measured values. If a given question was at a relatively high abstraction level, or was more removed from the available programming-language-dependent metrics, we handled it by dividing into sub-questions. In this way, the suite is structured in four layers, with the goals and questions being independent of language, and the sub-questions and metrics being basically language-dependent. This raises the reusability of the framework by clearly separating the fixed part of the framework (i.e. commonality) from the part which may require modification (i.e. variability).

An excerpt from the suite is shown in Figure 34. The suite is made up of 47 questions, 101 sub-questions and 236 metrics. As metrics, we used those which are supported by existing tools for the C programming language (such as QAC and Logiscope), the degree of conformance to existing coding style guides for the C language (such as MISRA-C [17] and IPA/SEC’s guide [18]), and other metrics which seemed necessary for particular questions or sub-questions where currently available measurement tools did not apply. 19% of the metrics could not be measured using currently available tools. One such example is the very specialized measurement, ”number of branches due to macros.” Further, 29% of the questions (either directly or via sub-questions) could not be assigned a metric at all. In the future, we will reduce or eliminate the proportion of metrics and questions that are not covered by developing new measurement tools. A selection of metrics from the full list is shown in Table 11. The table includes the following details to help the evaluator understand each metric.

- Type of measurement scope: System (ID: MSyXXX), Directory (MMdXXX), File (MF1XXX) or Function (MFnXXX).
- Type of rating level [19]: Threshold (quality is interpreted to be best when the measurement value is a particular value, or within a particular range), Minimal (the smaller the better) or Maximal (the larger the better).

¹ The entire suite is published in [16].

- Scale type [20]: Nominal, Ordinal, Interval, or Ratio.
- Programming language dependency type: Not (not dependent on language), Not-OO (non-object-oriented language dependent), OO (dependent on object-oriented language), C (C programming language), C++ (C++), or C&C++ (C or C++).

Table 1. List of metrics used (excerpt)

ID	Metric name	Rating	Scale	Dependency
MSy021	Number of recursive passes	Minimal	Ratio	Not
MmD027	Number of elements located directly below the directory	Threshold	Ratio	Not
MF1003	ELOC	Threshold	Ratio	Not
MFn072	Cyclomatic number	Minimal	Ratio	Not

Characteristic	Sub-characteristic	Goal	Question	Sub-question	Metric
Reliability	Maturity	Purpose : Evaluate Issue : the frequency of faults Object: source code Viewpoint: end-user	Q0100: Is the code not prone to faults?	Q0101 Has memory been initialized properly?	MF134: Number of un-initialized const objects. MF1107: Number of arrays with fewer initialization values than elements. MF1133: Number of strings which do not maintain null termination. MF1169: Number of enumerations not adequately initialized.
			
			Q0200: Is the scope not too large?	Q0201: Is the number of partition elements appropriate?	MMd027: Number of sub elements MMd008: Number of functions MF1003: Effective number of lines.
	Q0400: Is it possible to estimate the size of resources to be used?	Q0401: Is there not any recursive call?	Msy021: Number of recursive paths.		
	Fault tolerance
Maintainability	Analysability	Purpose: Evaluate Issue : the easiness of identifying styles, structure, behaviour and parts for maintenance Object: source code Viewpoint: developer	Q3700 Are the functions not too complicated?	Q3701 Is the function-call nesting not deep?	MFn095 Depth of layers in call graph
				Q3702 Is the logic not too complex?	MFn066 Max. nesting depth in control structure. MFn072 Cyclomatic number. MFn069 Estimated no. of static paths.

Fig. 3. Quality metrics suite (excerpt)

For example, Figure 3 gives several language-independent questions (e.g. Q3700) which help to evaluate how easy the source code is to analyze. Q3700 is quite abstract and difficult to measure directly, so it is broken-down into several sub-questions, including Q3701 and Q3702. Finally, metrics are assigned to each sub-question to make it possible to obtain data to answer them. The single metric, MFn095, is assigned to Q3701, and three metrics, MFn066, MFn072 and MFn069, are assigned to Q3702. By making these assignments, source code quality can be evaluated in quality-sub-characteristic units from the measurement

values. For example, it is clear in Table 1, that the measured value for the cyclomatic number of a function [21] can be used to evaluate the analysability of the source code. Also, since the type of rating level for cyclomatic number is "Minimal", the smaller the measurement value is, the better the analysability is for that part of the code.

(b) Technique and tool for deriving rating levels

In order to evaluate the permissible range of values for a particular quality characteristic from the measurements obtained using the suite, a rating level for each metric is required. Within the framework, such a rating level is derived using a collection of existing program source codes (denoted as the "acceptable set") that are acceptable from a quality point of view.

If such codes are not available, the framework uses source codes (denoted as the "after-improvement set") to which some quality improvements have been made while not altering the functionality, as an alternative to the acceptable set. Due to tradeoffs between different quality characteristics, there might be quality characteristics that have got worse compared to the before-improvement set, among all characteristics. However, we think the after-improvement set can be regarded approximately as an acceptable one, because some developers or clients accepted the set instead of the corresponding before-improvement in fact.

Measurements were made on the acceptable set or the after-improvement set, and rating levels of three different types were derived using the upper and lower hinges (i.e. 75th and 25th percentiles) of the statistical distributions of measurements from each metric as described below:

- Minimal: The rating level is below the upper hinge.
- Maximal: The rating level is above the lower hinge.
- Threshold: The rating level lies between the upper and lower hinges.

This derivation technique was implemented using spreadsheet software functions on the Excel worksheet. We applied this technique to three quality-improved, industrial embedded software S_1, S_2, S_3 (automobile or internal printer software) that we were able to obtain. The measurement values for program scale, before and after quality improvements, for the total of six programs are shown in Table 2. Comparing the programs before and after improvements, the improved versions appear to be implemented with a larger number of functions but the number of lines of code in each function is smaller.

As an example of another metric, the distribution of results from metric MFn072, "Cyclomatic number," before and after improvement, are shown in

Table 2. Scale totals for samples used

Type	Number of files	Number of functions	ELOC
Before improvement	603	3,269	174,650
After improvement	842	4,873	116,015
Total	1,445	8,142	290,665

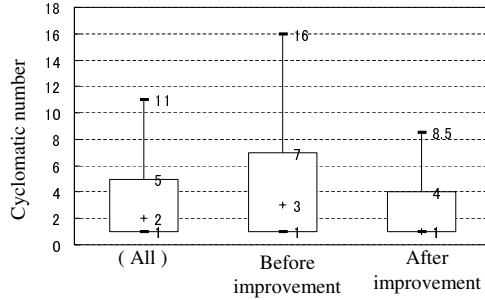


Fig. 4. MFn072 – Distribution of cyclomatic numbers (box-plot diagram)

Figure 4 as a box-plot diagram. In the box plot, the upper and lower edges of the rectangle indicate the upper and lower hinges, the value in the box is the median value, and the lines above and below the box give upper and lower adjacents. Figure 4 shows that compared to the before-improvement set, the after-improvement set tends to yield smaller values. Since the rating level type for the cyclomatic number values is "Minimal" (as shown in Table 1), the rating level allows values below the upper hinge of 4.

(c) Normalization/aggregation tool

To achieve an overall quality evaluation, we aggregate the measurements from the various metrics in the suite into quality-characteristic and module units by normalizing each measurement value, using the rating level, to a value from 0 to 100. More specifically, if a measurement value falls within the rating level as described above, it receives a score of 100. If it falls above the upper outer fence (upper hinge + 3·H-Spread) derived from the improved code set or below the lower outer fence (lower hinge - 3·H-Spread), it receives a score of zero. "H-Spread" (i.e. interquartile range) means the value of "upper hinge - lower hinge". A linear graph between these outer fences for each metric is created, and scores are interpolated linearly from the graph.

As an example, normalized values for measurements of the cyclomatic number are shown in Figure 5. According to Figure 4, the upper hinge for the improved code set is 4, and H-Spread is 3 (= 4-1), so a straight line from a measurement value of 4 to the value $4 + 3 \cdot 3 = 13$ is drawn in the score graph in Figure 5. Then, if the cyclomatic number is 2, the score taken from the graph is 100.

By transforming measurement values to normalized scores using a continuous, linear score graph in this way, small changes are reflected intuitively in the score, and values from different metrics can be compared with each other. It is also conceivable to construct the measurement normalization graphs using other forms such as non-linear curves or discontinuous step functions [22], but for the purpose of understanding overall trends in how small measurement values affect quality characteristics, these other graph types do not improve the results significantly, so linear graphs were selected as most appropriate.

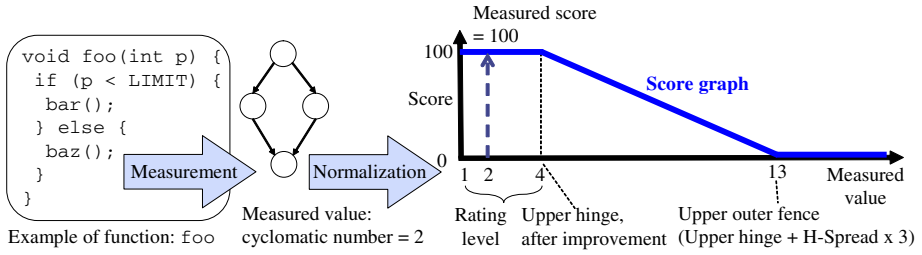


Fig. 5. Calculating the score for cyclomatic number

Next, a weighting is applied to each of the normalized scores, and they are aggregated in quality-characteristic, sub-characteristic and module units. The weighting mechanism allows for the influence of each of the questions or metrics to be adjusted but the standard settings simply use an even distribution (i.e. simply take the average). The aggregation model which forms the basis for aggregating the scores is shown as a UML class diagram in Figure 6. Also, the constraints related to scores in Figure 6 are given below in OCL [23].

```
context CharacteristicResult
-- Score is a weighted sum of the scores from each of the sub-characteristics
inv: score = SubCharacteristicResult->iterate( c:SubCharacteristicResult;
    result:Real=0 | result+c.score*c.SubCharacteristic.weight )
-- The total of all sub-characteristic weights is 1
inv: SubCharacteristicResult->SubCharacteristic->collect(weight)->sum() = 1
context SubCharacteristicResult
inv: Score=QuestionResult->iterate( q:QuestionResult;
    result:Real=0 | result+q.score*q.Question.weight )
inv: QuestionResult->Question->collect(weight)->sum() = 1
context QuestionResult
inv: Score=MeasurementResult->iterate( m:MeasurementResult;
    result:Real=0 | result+m.score*m.Metric.weight )
inv: MeasurementResult->Metric->collect(weight)->sum()=1
context MeasurementResult inv:
if underMeasurement.target <> Metric.target
-- Score is the average of the total of the same metric's measurement
-- result scores of all of the target program module unit's childs
score=underMeasurement->child->collect(MeasurementResult)->select(Metric.
id=self.Metric.id)->collect(score)->sum()/underMeasurement->child->size()
else
-- Score is equal to the measurement value normalized by using rating level
endif
```

An example of score calculation and aggregation based on the aggregation model is shown as a UML object diagram in Figure 7. For simplicity, quality characteristics and sub-characteristics, rating levels, directories and functions have been omitted, and only evaluation of the reliability of the whole system,

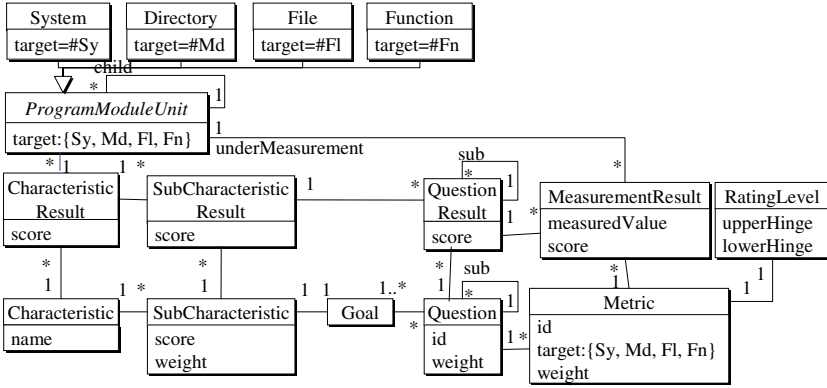


Fig. 6. Characteristic/module unit score calculation/aggregation model

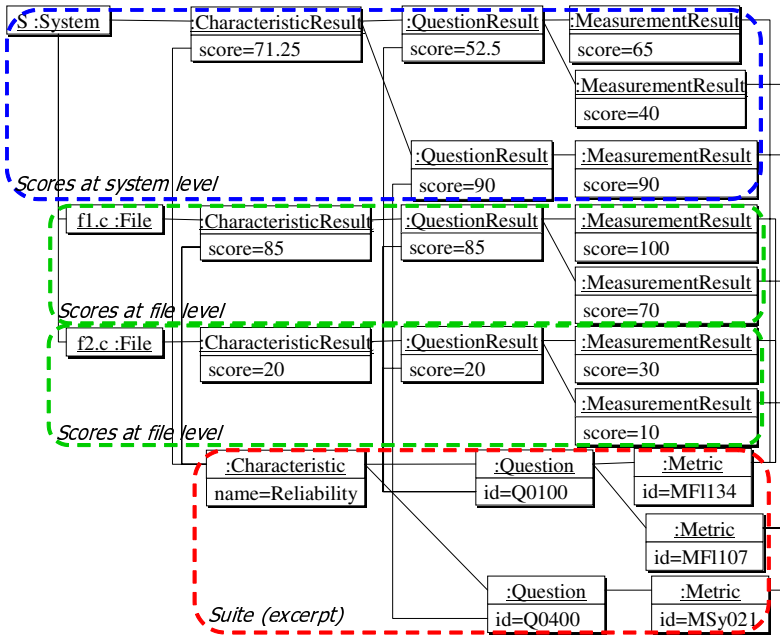


Fig. 7. Example of calculating the score using the aggregation model

S, is shown. Further, the evaluation is made based on only two files, f1.c and f2.c, and not on all directories. In Figure 7 the reliability score for S (71.25) is calculated for two questions from the scores from three metrics. Since the metrics MF1134 and MF1107 apply directly to the files, the average of the scores from measurement values from f1.c and f2.c was used. Figure 7 shows that detailed scores for each quality-characteristic or question can be obtained for the whole

system or for individual module (file) units (e.g. the reliability score for `f1.c` is 85). This normalization/aggregation technique has been implemented using Excel macros.

(d) Visualization tool and example of use

We implemented a visualization tool which displays the scores obtained from normalization and aggregation in module units for each quality characteristic, and allows detailed inspection based on module-inclusion relationships. The tool is implemented in Ruby and generates an evaluation report consisting of a collection of linked HTML pages using the scores calculated in Excel. An example is shown in Figure 8. The person evaluating the code can use the generated pages to get a comprehensive understanding of quality trends from the module level up to the overall system.

The figure shows two browser windows. The left window, titled 'Project result: flac - Microsoft Internet Explorer', displays a report with two tables. The first table, 'System scores', has columns for Reliability (88.8), Efficiency (91.8), and Maintainability (75.4). The second table, 'Directories (toplevel) scores', has columns for Directory, Reliability, and Efficiency, with rows for 'flac' (85.9, 85.7) and 'libFLAC' (87.6, 93.8). The right window, titled 'Module result: flac - Microsoft Internet Explorer', displays a 'Changeability details' table with columns for Question, Sub Question, Metric, and Score. The table contains three rows of data with scores 53.3, 75.9, and 45.0, and a final row with a score of 100.0.

Question	Sub Question	Metric	Score
Q3500 Are elements divided properly?	Q3501 Is the number of divided elements appropriate?	MF1003 ELOC	53.3
		MF002 ELOC	75.9
Q3502 Is the coupling among elements low?		MF1032 Estimated function coupling	45.0
Q3600 Are abstractions done properly?	Q3603 Are the roles appropriately divided?	MF1190 Ratio of functions without any parameters	100.0

Fig. 8. Report examples (left: system/directory, right: characteristic in detail)

3.3 Applicable Scope of the Framework

Because the framework covers quality from the overall system down to a detailed level, it can be used to evaluate quality over a wide range, from management level down to the individual module developer. Specifically, the scores can be used to identify and prioritize problematic characteristics or parts that need quality improvements. Also, if a range of allowable scores (e.g. 75 to 100 points) is set as an assessment criterion for an organization or project, scores can be used as a non-functional requirement during the development or procurement process.

The framework can be used in the following situations:

- Implementing or procuring C programs in the embedded software domain: Entire framework can be reused.
- Implementing or procuring C programs in the non-embedded software domain: All of the framework except for the derived rating level described in this paper can be reused if a collection of acceptable source code (or source code to which some quality improvements have been made) is available in the problem domain. If such a sample is not available, all of the framework

except the rating level and technique for deriving a rating level can be reused, and another technique for deriving a rating level can be incorporated in the framework.

- Implementing or procuring programs in other languages: The goals and questions within the suite which are language independent can be reused.

4 Experimental Evaluation

In the following, we evaluate the validity and usefulness (especially quality improvement reflection capability) of the framework by using several real programs.

4.1 Validity of the Framework in Quality Evaluation

We evaluate the validity of the framework by comparing two evaluation results for the same set of source codes: a qualitative evaluation by using a questionnaire, and a quantitative evaluation by using the framework.

First, we created a table of questions to evaluate each quality sub-characteristic with a four-level score (0, 50, 75 or 100 points), and applied it to the three embedded software programs (S_1, S_2, S_3) that were used to derive the rating level in section 3.2. The programmer in charge of each program before improvements or the person accepting the program after improvements was asked to perform this qualitative evaluation by answering the questions. Table 3 shows the average results of this evaluation in quality-characteristic units². "Before" and "After" in the table show the results for the code before and after quality improvements were made. For two of the program, S_1 and S_2 , the qualitative evaluation showed improvement for almost all quality characteristics.

Table 3. Qualitative quality results using the query table

	Reliability		Efficiency		Maintainability		Portability		Reusability	
	Before	After	Before	After	Before	After	Before	After	Before	After
S_1	92	92	80	83	75	95	69	100	92	100
S_2	59	79	67	71	54	78	76	88	60	83
S_3	–	92	–	78	–	75	–	88	–	83

Next, we compared the results of the qualitative evaluation described above with the quantitative results from the framework in order to verify the validity of the framework. The quantitative evaluation results for each of the programs, before and after improvement, are shown in Table 4. We examine the validity of the framework for each quality characteristic below:

- Reliability, maintainability and reusability: As with the qualitative evaluation results, the quantitative evaluation results for each of these characteristics

² Due to some reasons, the before-improvement qualitative results for S_3 were not available.

showed improvement, suggesting that the framework is valid for them. However, one program (S_3) did not show improvement in reusability afterwards, so it may be necessary to add additional metrics and corresponding quality mappings.

- Portability: The quantitative result for programs S_2 and S_3 showed improvement afterwards, so the framework may be useful for this evaluating this quality characteristic. However, the improvement seen in the qualitative evaluation of S_1 was not reflected in the quantitative evaluation, so it may be necessary to adjust or add to the metrics or quality mappings used.
- Efficiency: For all programs, the quantitative evaluation results showed different tendencies than the qualitative evaluation results for before and after quality improvements, suggesting that the metrics used were not appropriate. One reason for this may be that it is fundamentally difficult to estimate the final system’s efficiency by only using the source code [24]. We will need to make revisions to the metrics used here.

From the above-mentioned results, it is found that the framework can be used effectively to give quantitative evaluations of reliability, maintainability, reusability and portability of source code.

Table 4. Quantitative quality evaluation results using the framework

	Reliability		Efficiency		Maintainability		Portability		Reusability	
	Before	After	Before	After	Before	After	Before	After	Before	After
S_1	79	83	96	92	80	88	87	86	80	92
S_2	88	99	99	96	74	89	94	96	0	95
S_3	85	90	96	86	67	75	77	82	0	0

4.2 Quality Improvement Reflection Capability of the Framework

We used another embedded program which controls a Japanese shelf of gods for verifying the quality improvement reflection capability of the framework. In an earlier version, the program had maintenance problems such as the heavy use of global variables and the very long `main()` function. To solve these problems, we restructured the program by shifting global variables to non-global variables (such as local variables) and by dividing long functions into small ones. The excerpts of the programs before and after improvements are the following.

```

/***** shrine.c, before improvements *****/
extern int mic_threshold; extern int show_mic_value; ...
void main(void) {
    MY_ADCSR.BYTE = 0x31; while(!MY_ADCSR.BIT.ADF);
    PADDR = 0x7F; PADDR.BIT.B2 = 0;
    PADDR.BIT.B3 = 0; PADDR = 0x0C | PADDR; ...
/***** shrine.c, after improvements *****/
int main() {
    start_microphone(); init_switch(); init_motor(); ...

```

Figure 9 shows the quantitative evaluation results using the framework for each of the programs. The result of the after-improvement reflects a significant improvement in maintainability since several metrics related to maintainability provide different values. Regarding this example, it is found that the framework has the quality improvement reflection capability. Note that efficiency has been slightly decreased in the after-improvement due to the division of functions; this is a typical example of tradeoff between maintainability and efficiency.

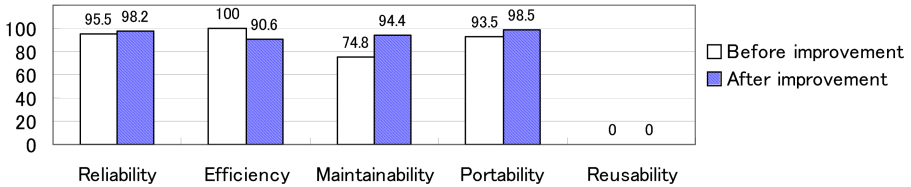


Fig. 9. Quantitative results using the framework for `shrine.c`

5 Conclusion and Future Work

In this paper, we propose a framework for evaluating the quality of program source code in order to resolve several problems faced by existing techniques. The framework focuses mainly on the C programming language and incorporates a quality metrics suite, a normalization and aggregation tool, a rating level derivation tool, and a set of actual rating levels. The framework is useful mainly for evaluating the quality of C language source code in module units from a detailed level up to whole systems and from individual quality sub-characteristics up to overall system quality. It would also be possible to apply the framework to source code in other languages by changing the sub-questions and metrics in the measurement suite. We verified that the framework can be used effectively to evaluate programs for reliability, maintainability, reusability and portability by applying it to several embedded software programs.

In further research, we plan to re-examine the metrics for some of the quality characteristics (particularly efficiency) to improve the accuracy of our quality evaluation by investigating the relation between the internal measured values (obtained by the framework) and possible external measurement values. Also, by applying the framework to many more programs, we will investigate how effective it is, and how it depends on the problem domain.

References

1. Ogasawara, H., et al.: Evaluating Effectiveness of Software Metrics, Union of Japanese Scientists and Engineers, 20SPC Research subcommittee report (2004)
2. Chaudron, M.: Evaluating Software Architectures, <http://www.win.tue.nl/mchaudro/swads/>

3. Sindre, G., et al.: The REBOOT Approach to Software Reuse, *Journal of Systems and Software*, 30(3) (1995)
4. Bansiya, J., Davis, C.G.: A Hierarchical Model for Object-Oriented Design Quality Assessment, *IEEE Transactions on Software Engineering*, 28(1) (2002)
5. Supervised by Kanno, A., et. al.: Software quality maintenance technology for the 21st Century, Union of Japanese Scientists and Engineers (1994)
6. Ortega, M., et al.: Construction of A Systematic Quality Model for Evaluating A Software Product, *Software Quality Journal*, 11(3) (2003)
7. Monden, A.: A Study of Data Collection using EPM and Analysis using GQM. In: 4th Empirical Software Engineering Workshop (2005)
8. ISO/IEC TR 9126-3: Software engineering – Product quality – Part 3: Internal metrics (2003)
9. ISO/IEC 9126-1: Information technology – Software product evaluation: Quality Characteristics and Guidelines for their use (2001)
10. ISO/IEC 14598-1: Information technology – Software product evaluation: Part 1: General overview (1998)
11. Washizaki, H., et al.: A Metrics Suite for Measuring Reusability of Software Components. In: Proc. 9th IEEE International Software Metrics Symposium (2003)
12. Hirayama, M., et al.: Evaluating Usability of Software Components, *Information Processing Society of Japan Journal*, 45(6) (2004)
13. Programming Research Ltd.: QAC, <http://www.programmingresearch.com/>
14. Telelogic: Logiscope, <http://www.telelogic.com/corp/products/logiscope/>
15. Basili, V.R., Weiss, D.M.: A Methodology for Collecting Valid Software Engineering Data, *IEEE Transactions on Software Engineering*, 10(6) (1984)
16. <http://www.ogis-ri.jp/solution/QAFramework.html>
17. The Motor Industry Software Reliability Association: MISRA-C: 2004 – Guidelines for the use of the C language in critical systems (2004), <http://www.misra-c2.com/>
18. IPA/SEC: C-Language Coding best practices for Embedded software Guide, Shoeisha Inc. (2006)
19. Emi, K., Lewerentz, C.: Applying Design-Metrics to Object-Oriented Frameworks. In: Proc. 3rd IEEE International Software Metrics Symposium (1996)
20. ISO/IEC 15939:2002, Software engineering – Software measurement process (2002)
21. McCabe, T.J., Watson, A.H.: Software Complexity, Crosstalk, *Journal of Defense Software Engineering*, 7(12) (1994)
22. Kazman, R., et al.: Making Architecture Design Decisions: An Economic Approach, CMU/SEI-2002-TR-035 (2002)
23. OMG: UML 2.0 OCL Specification, <http://www.omg.org/docs/ptc/05-06-06.pdf>
24. Washizaki, H., et al.: Experiments on Quality Evaluation of Embedded Software in Japan Robot Software Design Contest. In: Proc. 28th International Conference on Software Engineering (ICSE 2006), pp.551–560 (2006)

Software Fault Prediction with Object-Oriented Metrics Based Artificial Immune Recognition System

Cagatay Catal¹ and Banu Diri²

¹ TUBITAK-Marmara Research Center, Information Tech. Ins., 41470 Turkey
cagatay.catal@bte.mam.gov.tr

² Yildiz Technical University, Computer Engineering Dept., 34349 Turkey
banu@ce.yildiz.edu.tr

Abstract. Software testing is a time-consuming and expensive process. Software fault prediction models are used to identify fault-prone classes automatically before system testing. These models can reduce the testing duration, project risks, resource and infrastructure costs. In this study, we propose a novel fault prediction model to improve the testing process. Chidamber-Kemerer Object-Oriented metrics and method-level metrics such as Halstead and McCabe are used as independent metrics in our Artificial Immune Recognition System based model. According to this study, class-level metrics based model which applies AIRS algorithm can be used successfully for fault prediction and its performance is higher than J48 based approach. A fault prediction tool which uses this model can be easily integrated into the testing process.

1 Introduction

Software systems are becoming more and more complex and people's quality expectations are increasing. Therefore, it is necessary to manage these expectations as an engineering discipline called Software Quality Engineering [1]. Software Quality Engineering consists of many Quality Assurance activities and an important subset of them is testing. Other subsets are fault prevention, inspection, fault tolerance, formal verification and fault prediction.

These subsets can detect many software problems and even improve software testing process. Improvements in the testing process will reduce development life-cycle, project risks, resource and infrastructure costs. Because testing process is time-consuming and expensive, we may anticipate this problem with fault prediction models. These models provide a test strategy by focussing on fault-prone modules and testing duration decreases with this approach. In this study, we propose a novel fault prediction model to improve the testing process. Our goal is to predict the classes that will contain faults at the next release of an Object-Oriented System.

Current software metrics and defect data are used to construct the prediction model for the next release of software. Most of the datasets which locate in

PROMISE repository [2] have been collected at NASA as a part of Software Metrics Data Program for development projects. In this study, we have used datasets from this repository to construct our fault prediction model.

Metrics are independent variables and the fault-proneness of module is the dependent variable. Process or product metrics can be used for independent variables but mostly product metrics are used. Method-level and class-level metrics are two different metrics groups inside product metrics. Actually, Object-Oriented programming and procedural programming can benefit from method-level metrics because these programming paradigms have methods. In this study, class-level and method-level metrics have been used.

We use Object-Oriented metrics from Chidamber-Kemerer (CK) metrics suite [3] and we desire to enhance the performance of our prediction model with these metrics. Genetic Programming, Decision Trees, Neural Networks, Case-based Reasoning, Fuzzy Logic, Logistic Regression and Discriminant Analysis have been applied effectively for software fault prediction. As method-level metrics, Object-Oriented metrics are widely used for fault prediction and recent studies focused on these metrics [4, 5, 6, 7, 8, 9, 10, 11, 12, 13], and [14].

In this paper, we applied class-level metrics and method-level metrics for AIRS based fault prediction model. We also investigated each of CK metrics within our proposed model. Our aim is not validating individual metrics, but applying AIRS to build a better fault prediction model. Furthermore, we used different statistical techniques such as correlation-based feature selection technique to select relevant features and investigated the power of the model. In addition, this study indicates that OO metrics are more useful than traditional metrics for our proposed model. Performance criteria shows that our model which uses CK metrics and lines of code provides better prediction capability than other models given in literature. According to experimental results, the performance of our prediction model is remarkable. This study is a part of our on-going research on software quality modeling and we aim to reach our vision by using Artificial Immune Systems paradigm.

This paper is organized as follows: the following section presents datasets and OO metrics. Section 3 describes evaluation criteria. Section 4 introduces natural and artificial immune systems. Section 5 provides experimental results. Section 6 presents conclusions and future works.

2 Metrics and Dataset

Chidamber-Kemerer OO metrics have been proposed in 1994 [3]. They have been used by many tool vendors and researchers. This study uses six Object-Oriented metrics of CK metrics. Also, we tried to get benefit from four more class level metrics too. Koru et al. [15] identified these four metrics for KC1 project, *Percent_Pub_Data*, *Access_To_Pub_Data*, *Dep_On_Child*, and *Fan_In*. We have also attempted to take advantage from traditional method-level metrics which are based on Halstead and McCabe metrics. McCabe is interested in the complex pathways and Halstead focuses on the readiness of the source code.

KC1 dataset has method-level metrics and Koru et al. [15] converted them into class-level ones using minimum, maximum, average and sum operations. Therefore, 21 method-level metrics were converted into 84 class-level metrics. We had 84 metrics from method-level metrics transformation and 10 metrics from class-level metrics. Transformed metrics are shown in Table 1. Object-Oriented metrics are *Percent_Pub_Data*, *Access_To_Pub_Data*, *Dep_On_Child*, *Fan_In*, *Coupling_Between_Obj*, *Depth_Of_Inheritance_Tree*, *Lack_Of_Cohes_Of_Methods*, *Num_Of_Children*, *Response_For_Class*, and *Weighted_Method_Per_Class*.

Object-Oriented metrics which have been used in this study are described below [4], [2]:

- *WMC (Weighted Methods per Class)* is # of methods which locate in each class.
- *DIT (Depth of Inheritance Tree)* is the distance of the longest path from a class to the root in the inheritance tree.
- *RFC (Response for a Class)* is # of methods which can be executed to respond a message.
- *NOC (Number of Children)* is # of classes which are direct descendants for each class.
- *CBO (Coupling Between Object Classes)* is # of different non-inheritance-related classes to which a class is coupled.
- *LCOM (Lack of Cohesion in Methods)* is related to the access ratio of attributes.
- *Percent_Pub_Data* is the percentage of data which is public or protected.
- *Access_To_Pub_Data* is the amount of public or protected data access for a class.
- *Dep_On_Child* shows whether a class is dependent on a descendant or not.
- *Fan_In* is the number of calls by higher modules.

WMC, DIT, RFC, NOC, CBO, and LCOM belong to Chidamber and Kemerer [3] suite. In 1996, Basili et al. [10] studied these metrics. Rosenberg et al. [16] suggested six CK metrics plus cyclomatic complexity, lines of code, and comment percentage for reliability assessment. In 1999, Briand et al. [17] observed that CBO, RFC and LCOM are considerably correlated for fault-proneness of classes by using logistic regression. In 2000, Briand et al. [12] explored the impact of lines of code metric (SLOC) and concluded that CBO, RFC, DIT, SLOC, NOC are important metrics for fault prediction. In 2001, Briand et al. [18] reported that NOC is not significant and this study is contradictory with his previous research. Tang et al. [19] reported that DIT, NOC, CBO are not significant and only WMC, RFC are significant. Yu et al. [13] showed that WMC, SLOC, CBO, RFC, LCOM, NOC are significant but DIT is not important for fault prediction. Gyimothy et al. [5] resulted that NOC is not significant but SLOC and other CK metrics are significant for fault prediction. Zhou et al. [4] showed that except DIT all the other CK metrics and SLOC are significant. According to previous studies, we can say that most of these studies used logistic regression, and WMC, RFC, CBO are almost found to be significant for fault prediction [4].

Table 1. Transformed Metrics

Attributes	Information
loc	McCabe's line count of code
v(g)	McCabe's cyclomatic complexity
ev(g)	McCabe's essential complexity
iv(g)	McCabe's design complexity
n	Halstead total operators + operands
v	Halstead volume
l	Halstead program length
d	Halstead difficulty
i	Halstead intelligence
e	Halstead effort
b	Halstead
t	Halstead's time estimator
IOCode	Halstead's line count
IOComment	Halstead's count of lines of comments
IOBlank	Halstead's count of blank lines
IOCodeAndComment	Lines of comment and code
uniq_Op	Unique operators
uniq_Opnd	Unique operands
total_Op	Total operators
total_Opnd	Total operands
branchCount	Branch count of the flow graph

3 Performance Measurement Criteria

Faults in software systems exist in small portions of components and this information has been confirmed in many researches [20], [21]. Therefore, datasets which are studied for software fault prediction model are skewed. PROMISE repository has some fault datasets from NASA projects and the largest one, JM1, has 19% faulty modules. 15% of KC1, 7% of PC1, 21% of KC2 and 10% of CM1 have faults. ROC (receiver operating characteristics) analysis can be used for these datasets but it has some limitations. Researchers still study on different measurement techniques for skewed datasets and F-measure is the most popular one. Different criteria and non-public datasets prevent software engineering community to find the best prediction model and make it difficult to compare the models. Recently Ma et al. [22] used G-mean1, G-mean2 and F-measure in order to benchmark different machine learning algorithms. In this study, we use these performance indicators to evaluate our model. Some researchers use sensitivity and specificity indicators to assess their models [23]. El-Emam et al. [24] suggested using J coefficient of Youden [25] for binary classifiers in software engineering. El-Emam et al. [23] calculated the AUC (area under curve) from ROC curve. If AUC is about 1.0, that model is said be perfect for classification. Recently, many researchers started to apply F-measure in order to benchmark and assess software fault prediction models [15], [22]. We constructed our model using past project metrics from KC1 project and tested using cross-validation.

After cross-validation, we built the confusion matrix given in Table 2 and then computed the performance indicators (G-mean1, G-mean2, F-measure) using following formulas. ROC analysis uses confusion matrix and formulas use values from this matrix.

Table 2. Confusion Matrix

	NO(Predicted)	YES(Predicted)
NO(actual)	True Negative (TN)	False Positive (FP)
YES(actual)	False Negative (FN)	True Positive (TP)

Faulty modules are regarded as positive (YES) and faulty-free modules are regarded as negative (NO). In this study, we computed following performance indicators and their formulas are given below [22]:

- Recall (PD): Recall is the percentage of fault-prone modules that are classified correctly and it is computed using Equation 1.

$$Recall = \frac{TP}{TP + FN} \quad (1)$$

- Accuracy: Accuracy is the likelihood of correctly predicted number of modules and it is computed using Equation 2.

$$Accuracy = \frac{TP + TN}{TP + FN + FP + TN} \quad (2)$$

- TNR: TNR is the ratio of correctly predicted faulty-free modules and it is computed using Equation 3.

$$TNR = \frac{TN}{TN + FP} \quad (3)$$

- Precision: TNR is the fraction of faulty modules that are actually faulty and it is computed using Equation 4.

$$Precision = \frac{TP}{TP + FP} \quad (4)$$

- G-mean1 and G-mean2: These indicators are recently used for benchmarking and they are computed using Equation 5 and Equation 6.

$$G - mean1 = \text{sqrt}(PD * Precision) \quad (5)$$

$$G - mean2 = \text{sqrt}(PD * TNR) \quad (6)$$

- F-measure: This indicator is the harmonic average of Precision and PD.

Ma et al. [22] investigated various machine learning algorithms for fault prediction and they used G-mean1, G-mean2 and F-measure in order to benchmark. In this study, we calculate them after 10-fold cross-validation.

4 Immune Systems and Artificial Immune Recognition Systems

4.1 Natural Immune Systems

Immune system has two mechanisms that interact with each other. The first one is the innate immune system and there is no need to interact with the relevant pathogen in the past for this type of mechanism [26]. The second one is the adaptive immune system which enhances its ability in order to detect more pathogens after the interaction. Lymphocytes are classified as B-cells and T-cells. Each lymphocyte can bind to a particular invader which is known as antigens. *B-cells are antibody-secreting cells and T-cells kill antigens* [27]. The similarity degree between B-cell and antigen is called affinity. If the antigen is detected by B-cell, B-cell is cloned with a process which is called clonal expansion. These clones are mutated using somatic hypermutation process according to the affinity level. Some of these clones are differentiated into B memory cells in order to respond rapidly for the next attack from same invader and therefore immune system is said to have memory feature. Furthermore, best fitting clones with antigens survive and this process is known as clonal selection. Distributed control, learning by experience, adaptation and parallel processing are main features of immune systems.

4.2 Artificial Immune Systems

Neural Networks, Evolutionary Computation, Genetic Algorithms, Swarm Intelligence, Ant Optimization and Artificial Immune Systems are examples of biologically inspired computing paradigms. Artificial Immune Systems (AIS) are machine learning algorithms which inspire from vertebrate immune systems to solve complex problems. They have been used for pattern recognition, computer security, function optimization, robotics, scheduling, aircraft control, data mining and anomaly detection problems [28]. Some researchers in AIS community studied on immune network theory [29, 30] which has been proposed by Jerne [31] even though immunologists refuted this theory [32]. Timmis et al. [30] developed a resource limited artificial immune systems and used the term *artificial recognition ball (ARB)* to represent the collection of similar B cells. When each antigen is presented to the ARB, cells acquire resources based on the stimulation value but resources are limited. Therefore, least stimulated cells are removed until the numbers of distributed resources are smaller than the allowable number. This approach has also been used by Watkins [28] for Artificial Immune Recognition System (AIRS) algorithm but Watkins [28] did not use network representation. Another interesting mechanism in immune system is negative selection. Natural immune systems destroy antibodies that bind to self cells using negative selection. Otherwise, some proteins of human body can be detected as invader by some antibodies and autoimmunity problem can occur. Forest et al. [33] imitated this mechanism and proposed an algorithm which has three steps: describe self, produce detector and observe anomalies [27]. This algorithm

has been used to detect computer viruses. The other mechanism is clonal selection and De Castro et al. used this process to propose an algorithm which is called CLONALG [34]. Clonal selection process includes recognition of antigen, cloning, mutation and differentiation to memory cells. Even though CLONALG looks like evolutionary algorithms, very distinct features exist such as working with binary representation.

4.3 Artificial Immune Recognition Systems (AIRS) Algorithm

Most of studies in AIS have focused on unsupervised learning algorithms until 2001 and Watkins [28] decided to show that artificial immune systems could be used for classification problems. The only exception was Carter's study [35] that proposed a classification system based on AIS in 2000 and his approach was complex. Watkins [28] demonstrated that AIS which uses supervised learning could be used for classification problems [36]. This algorithm uses resource-limited approach of Timmis et al. study [30] and clonal selection principles of De Castro et al. [34] study. After first AIRS algorithm, some researchers proposed a modified AIRS which has a better complexity and a few decreasing in accuracy [37]. Watkins [38] also developed a parallel AIRS. AIRS algorithm has powerful features which are listed below [39]:

- *Generalization*: It does not need all the dataset for generalization and it has data reduction capability.
- *Parameter Stability*: Even though user-defined parameters are not optimized for the problem, reduction of its performance is very small.
- *Performance*: It has been demonstrated that its performance is best for some datasets and remarkable.
- *Self-regulatory*: There is no need to choose a topology before training.

AIRS algorithm has 5 steps: Initialization, antigen training, competition for limited resource, memory cell selection and classification [39]. First step and last step is applied just one time but step 2, 3, 4 are used for each sample in dataset. Sample is called *antigen* for AIRS algorithm. Brownlee [39] coded this algorithm for Java language and now it is accessible from weka.classalgos.sourceforge.net website. Activity diagram of this algorithm is given in Figure II. Details of the algorithm are given below but more detail can be accessed through source code which is distributed with GPL (General Public License) license.

- *Initialization*: Dataset is prepared for training step. Affinity Threshold variable is constructed.
- *Antigen Training*: Antigens (training data) are presented to the memory pool one by one. Recognition cells in memory pool are stimulated and given a stimulation value. The recognition cell which has a maximum stimulation value is selected as the best match memory cell and used for affinity maturation process. This cell is cloned and mutated. Clones are added to the Artificial Recognition Ball (ARB) pool. The number of clones is calculated

using Equation 8. Equation 7 shows how to calculate stim value. ClonalRate and hyperMutationRate are user-defined parameters.

$$stim = 1 - affinity \quad (7)$$

$$numClones = stim * clonalRate * hypermutationRate \quad (8)$$

- *Competition for limited resource*: Competition begins when mutated clones are added to the ARB pool. ARB pool is stimulated with antigen and limited resource is assigned according to stimulation values. ARBs which do not have enough resources are thrown from pool. If stopping criteria is satisfied, process stops. Otherwise, mutated clones of ARBs are generated and recursion goes on until stopping criteria is met.
- *Memory cell selection*: In previous step, when stopping criteria is met, ARB which has a maximum stimulation score is chosen as candidate memory cell. If ARB's stimulation value is better than the original best matching memory, ARB is copied to the memory cell pool.
- *Classification*: Learning process is finished before this step starts. Final memory cell pool is used for cross-validation or testing new data. K-nearest neighbor (k-nn) approach is applied in this step.

5 Experimental Results

AIRS has been used for fault prediction on NASA public datasets by Catal et al. [40]. AIRS algorithm along with the Correlation Based Feature Selection technique provides high performance for large scale projects. Catal et al. [40] did not use Object-Oriented (OO) Metrics but applied method-level metrics. This study uses OO metrics for AIRS based prediction model. Significant OO metrics and best combination of OO metrics have been identified in this study. The significance level of each OO metric is based on the performance of AIRS based model and performance indicators are G-mean1, G-mean2 and F-measure. Results are shown in Table 3. According to the Table 3, AIRS based fault prediction model which applies the combination of CK metrics with lines of code metric has better prediction performance than other AIRS based models which use other independent variables.

In Table 3, we can see that DIT is the only metric which is not significant for fault prediction when threshold level for performance indicators is chosen 0.5. The indicators of other CK metrics are higher than 0.5 when AIRS based model uses k as 1 or 3. This empirical result is consistent with Zhou et al.'s study [4] which applied Univariate Logistic Regression in order to identify significant metrics for the same dataset. They stated that DIT is the only metric which is not significant according to their ungraded severity faults analysis. Even though our approach is based on the performance indicators of our prediction model, results are same. Tang et al. [19], Yu et al. [13] and El Emam et al. [8] showed that DIT is not significant using logistic regression, ordinary least square linear regression and logistic regression respectively.

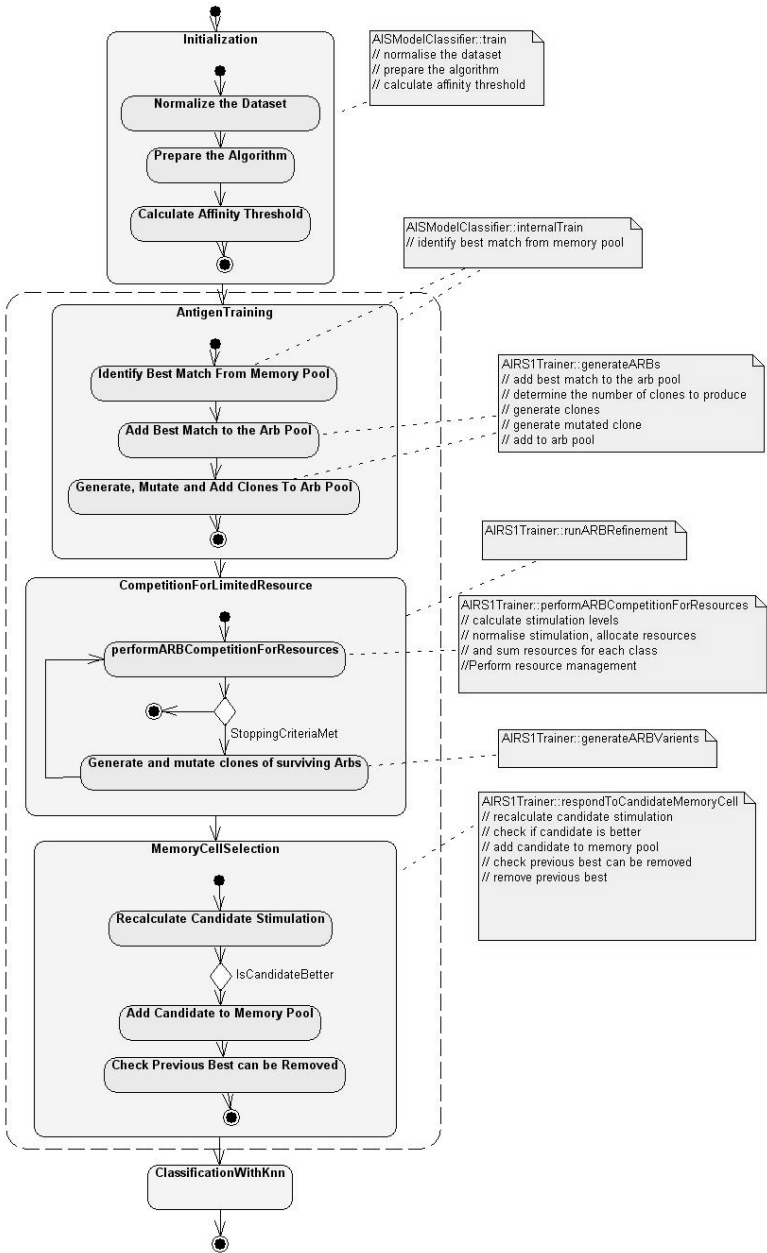


Fig. 1. This diagram shows the activity diagram of AIRS algorithm

According to Table 3, we can conclude that CBO metrics is the most significant CK metrics because of its highest performance indicators. Again this important result is consistent with Zhou et al. study [4] which states that CBO

Table 3. Experimental results with AIRS for kc1-class-level file

Inputs	k	PD	ACC	Prec	G-1	G-2	F
WMC	1	0.548	0.538	0.506	0.527	0.545	0.526
WMC	3	0.504	0.517	0.576	0.539	0.505	0.538
DIT	1	0.470	0.469	0.459	0.465	0.471	0.465
DIT	3	0.508	0.496	0.447	0.476	0.507	0.475
RFC	1	0.472	0.476	0.506	0.488	0.469	0.488
RFC	3	0.570	0.565	0.553	0.561	0.568	0.561
NOC	1	0.517	0.503	0.447	0.481	0.515	0.480
NOC	3	0.510	0.552	0.729	0.610	0.518	0.600
CBO	1	0.639	0.683	0.765	0.690	0.672	0.696
CBO	3	0.594	0.627	0.706	0.647	0.615	0.645
LCOM	1	0.520	0.524	0.541	0.530	0.520	0.530
LCOM	3	0.541	0.524	0.471	0.505	0.536	0.503
SLOC	1	0.558	0.545	0.506	0.532	0.553	0.531
SLOC	3	0.579	0.552	0.482	0.528	0.568	0.526
6 CK metrics	1	0.633	0.662	0.718	0.674	0.653	0.673
6 CK metrics	3	0.646	0.676	0.729	0.686	0.667	0.685
SLOC + 6 CK metrics	1	0.653	0.655	0.659	0.656	0.654	0.656
SLOC + 6 CK metrics	3	0.712	0.724	0.741	0.726	0.721	0.726
94 metrics	1	0.613	0.662	0.765	0.685	0.649	0.681
94 metrics	3	0.631	0.669	0.741	0.684	0.658	0.682
10 OO metrics	1	0.590	0.614	0.671	0.629	0.604	0.628
10 OO metrics	3	0.621	0.641	0.682	0.651	0.634	0.650
SLOC + 10 OO metrics	1	0.650	0.662	0.682	0.666	0.658	0.666
SLOC + 10 OO metrics	3	0.676	0.710	0.765	0.719	0.702	0.718
cfsSubsetEvaluated metrics	1	0.690	0.710	0.741	0.715	0.705	0.715
cfsSubsetEvaluated metrics	3	0.669	0.696	0.741	0.704	0.689	0.703
consistencySubsetEvaluated metrics	1	0.647	0.690	0.765	0.704	0.679	0.701
consistencySubsetEvaluated metrics	3	0.640	0.676	0.741	0.689	0.666	0.687
SLOC, WMC, RFC, NOC, CBO, LOCM	1	0.633	0.662	0.718	0.674	0.653	0.673
SLOC, WMC, RFC, NOC, CBO, LOCM	3	0.659	0.690	0.741	0.699	0.681	0.698

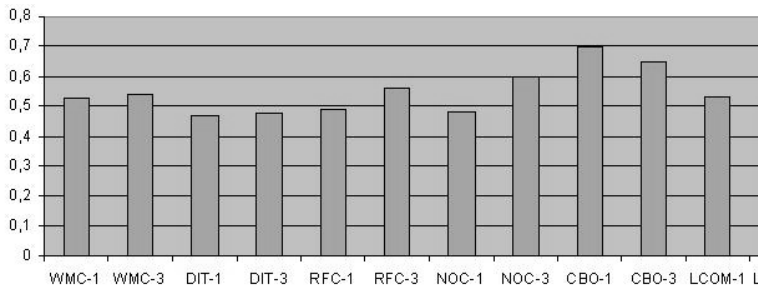
**Fig. 2.** Performance effect of each CK metrics for AIRS based model

Table 4. Experimental results for kc1 file with AIRS and other techniques

Methods	PD	ACC	Prec	G-1	G-2	F
Logistic	0.752	0.711	0.317	0.488	0.727	0.446
Discriminant	0.638	0.790	0.390	0.499	0.722	0.484
Tree	0.193	0.848	0.523	0.318	0.432	0.282
RuleSet	0.187	0.852	0.564	0.325	0.427	0.281
Boosting	0.169	0.862	0.732	0.352	0.409	0.275
KStar	0.509	0.855	0.532	0.521	0.684	0.520
VF1	0.957	0.195	0.156	0.387	0.231	0.269
RF (0.9,0.1)	0.844	0.711	0.330	0.528	0.761	0.475
RF (0.8,0.2)	0.700	0.782	0.387	0.520	0.747	0.498
RF (0.7,0.3)	0.561	0.825	0.447	0.501	0.700	0.498
AIRS1 (k=3)	0.634	0.746	0.298	0.435	0.586	0.405
cfs-AIRS1(k=3)	0.692	0.763	0.368	0.504	0.628	0.480
cfs-AIRS1(k=1)	0.589	0.660	0.426	0.501	0.569	0.494
pca-AIRS1(k=3)	0.563	0.653	0.383	0.464	0.548	0.456
pca-AIRS1(k=1)	0.574	0.634	0.448	0.507	0.561	0.503
cse-AIRS1(k=3)	0.708	0.773	0.374	0.515	0.638	0.489
cse-AIRS1(k=1)	0.647	0.715	0.426	0.525	0.609	0.514

Table 5. Benchmark of our model with Koru et al. [15] for kc1-class-level-data

Dataset	Algorithms and Inputs	Class Type	Prec	Recall	F
kc1-class-level-data	J48, 94 metrics	HR	0.62	0.68	0.65
	J48, 94 metrics	LMR	0.76	0.71	0.73
kc1-class-level-data	k=3 for AIRS and SLOC, WMC, DIT, RFC, NOC, CBO, LCOM	HR	0.66	0.70	0.68
	k=3 for AIRS and SLOC, WMC, DIT, RFC, NOC, CBO, LCOM	LMR	0.78	0.74	0.76
kc1-class-level-data	J48 and SLOC, WMC, DIT, RFC, NOC, CBO, LCOM	HR	0.63	0.63	0.63
	J48 and SLOC, WMC, DIT, RFC, NOC, CBO, LCOM	LMR	0.74	0.74	0.74

is the most important CK metric according to the R^2 value. R^2 value shows the effect of the independent variables in Univariate Analyses. However, they identified that SLOC has higher R^2 value than CBO. Therefore, SLOC is more important metric than CBO according to their analyses but this is not true for our study. Even though there is a difference at this point, two studies result that CBO is the most significant metric in CK metrics suite. In Figure 2, performance effects of them are shown.

According to Koru et al. [15], class-level data provide better prediction performance than method-level data. We also compared some prediction models which use method-level data with our new model which uses class-level data. *KC1* defect dataset which has only method-level metrics were used by Catal et al. [40] and results are shown in Table 4. This table has also experimental result

from Ma et al.'s study [22] which proposes Random Forests for software defect prediction. Even though best prediction model, RF, with method-level data is used, its performance indicators are lower than our new AIRS based model for class-level data. We can conclude that the usage of class-level data rather than method-level one improves the prediction performance for our model. Koru et al. [15] obtained better prediction performances with J48 and KStar for *kc1-class-level* dataset. Furthermore, they stated that comparing various algorithms is not their main purpose.

Our next step for our experiments is to compare our new AIRS based model with results of Koru et al. [15] study because we used the same dataset with their study. Their performance indicators are Precision, Recall and F-measure and therefore we computed these values for our experiments. The comparative results are shown in Table 5. Our AIRS based prediction model provides higher F-measure values, 0.68 and 0.76, than J48 based model for both HR and LMR classes. Because metrics were different for two approaches, we computed the performance of J48 technique with same metrics we used in our model. The last two lines in Table 5 show these results. According to the F-measure values in Table 5, we can conclude that AIRS based prediction model with six CK metrics and lines of code provide better prediction performance than J48 based prediction approach.

6 Conclusions and Future Work

Software testing process is an expensive one and it needs to be improved. Fault prediction models can reduce the testing duration, resource and infrastructure costs. Previous studies used different techniques to construct such a model but our aim is to use the Artificial Immune Systems paradigm which provides remarkable results in many complex problems such as intrusion detection.

In this study, we created our model using Artificial Immune Recognition System (AIRS) algorithm which is a supervised learning one. First, we tried to apply individual Object-Oriented metrics as independent variables for AIRS. Then, we tried several combinations of Object-Oriented (OO) metrics with method-level metrics. OO metrics with lines of code metric provided a better prediction performance than other combinations. The acquired results are remarkable for our Fault Prediction Research Program. We believe that the experimental results will give an impetus to us for our on-going research on software quality prediction.

This paper makes a number of contributions: First, we show new evidence which points out the association between six CK metrics and fault-proneness for AIRS based fault prediction models. Furthermore, we identified the significance level of CK metrics and results were consistent with other studies in literature. DIT was not significant and CBO was very significant for our model as in literature. Second, as a new model, we propose an AIRS based fault-prone module classification system with Object-Oriented metrics. Third, we show new evidence that class-level data is better than method-level data. Last, we present the effectiveness of our model and compare it with J48.

One drawback of our study as Zhou et al. [4] study is the use of a single project dataset because there are no more public NASA datasets in PROMISE repository as kc1 which has OO metrics. For the future, we will try to develop a prediction model that can be used when there is no previous fault or metrics data for a software system. When there is no previous data about metrics or fault, this classification problem (faulty or not-faulty) can be thought as a clustering problem. After constructing clusters, an expert is needed to give a label to each cluster. Clustering approach with expert opinion will not be a fully automated process but it will facilitate the testing process when there is no previous fault data for a software system.

References

1. Tian, J.: *Software Quality Engineering: Testing, Quality Assurance, and Quantifiable Improvement*. John Wiley and Sons Inc, Hoboken (2005)
2. Sayyad, S.J., Menzies, T.J.: *The PROMISE Repository of Software Engineering Databases*, University of Ottawa, Canada (2005), <http://promise.site.uottawa.ca/SERepository>
3. Chidamber, S.R., Kemerer, C.F.: A Metrics Suite for Object-Oriented Design. *IEEE Trans. on Software Eng* 20(6), 476–493 (1994)
4. Zhou, Y., Leung, H.: Empirical Analysis of Object-Oriented Design Metrics for Predicting High and Low Severity Faults. *IEEE Trans. on Software Eng* 32(10), 771–789 (2006)
5. Gyimothy, T., Ferenc, R., Siket, L.: Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction. *IEEE Trans. on Software Eng.* 31(10), 897–910 (2005)
6. Subramanyan, R., Krisnan, M.S.: Empirical Analysis of CK Metrics for Object-Oriented Design Complexity. *IEEE Trans. on Software Eng.* 29(4), 297–310 (2003)
7. Alshayeb, M., Wei, L.: An Empirical Validation of Object-Oriented Metrics in Two Different Iterative Software Processes. *IEEE Trans. on Software Eng.* 29(11), 1043–1049 (2003)
8. El Emam, K., Benlarbi, S., Goel, N., Rai, S.N.: The Confounding Effect of Class Size on the Validity of OO Metrics. *IEEE Trans. on Software Eng.* 27(7), 630–650 (2001)
9. Chidamber, S.R., Darcy, D.P., Kemerer, C.F.: Managerial Use of Metrics for Object-Oriented Software: An Exploratory Analysis. *IEEE Trans. on Software Eng.* 24(8), 629–639 (1998)
10. Basili, V.R., Briand, L.C., Melo, W.L.: A Validation of Object-Oriented Design Metrics as Quality Indicators. *IEEE Trans. on Software Eng.* 22(10), 751–761 (1996)
11. Succi, G., Pedrycz, W., Stefanovic, M., Miller, J.: Practical Assessment of the Models for Identification of Defect-Prone Classes in Object-Oriented Commercial Systems Using Design Metrics. *J. Systems and Software* 65(1), 1–12 (2003)
12. Briand, L.C., Wust, J., Daly, J.W., Porter, D.V.: Exploring the Relationships between Design Measures and Software Quality in OO Systems. *J. Systems and Software* 51(3), 245–273 (2000)
13. Yu, P., Systa, T., Muller, H.: Predicting Fault-Proneness Using OO Metrics. In: *Proc. Sixth European Conf. Software Maintenance and Reeng.* pp. 99–107 (2002)

14. Briand, L.C., Melo, W.L., Wust, J.: Assessing the Application of Fault-Proneness Models Across OO Software Projects. *IEEE Trans. on Software Eng.* 28(7), 706–720 (2002)
15. Koru, A.G., Liu, H.: An Investigation of the Effect of Module Size on Defect Prediction Using Static Measures. In: *Int'l Workshop on Predictor Models in Software Engineering*, Missouri, USA, pp. 1–5 (2005)
16. Rosenberg, L., Stapko, R., Gallo, A.: OO Metrics for Reliability. In: *IEEE Int'l. Symposium on Software Metrics* (1999)
17. Briand, L.C., Wust, J., Ikononovski, S.V., Lounis, H.: Investigating Quality Factors in Object-Oriented Designs. In: *21st Int'l Conf. Software Eng.* pp. 345–354 (1999)
18. Briand, L.C., Wust, J., Lounis, H.: Replicated Case Studies for Investigating Quality Factors in Object-Oriented Designs. *Empirical Software Eng.* 6(1), 11–58 (2001)
19. Tang, M.H., Kao, M.H., Chen, M.H.: An Empirical Study on Object-Oriented Metrics. In: *Sixth Int'l Software Metrics Symposium*, pp. 242–249 (1999)
20. Moller, K.H., Paulish, D.: An Empirical Investigation of Software Fault Distribution. In: *First International Software Metrics Symposium*, pp. 82–90 (1993)
21. Ohlsson, N., Alberg, H.: Predicting Fault-prone Software Modules in Telephone Switches. *IEEE Trans. on Software Eng.* 22(12), 886–894 (1996)
22. Ma, Y., Guo, L., Cukic, B.: A Statistical Framework for the Prediction of Fault-Proneness, *Advances in Machine Learning Application in Software Eng.* Idea Group Inc. (2006)
23. El Emam, K., Melo, W., Machado, J.: The Prediction of Faulty Classes Using OO Design Metrics. *J. Systems and Software* 56(1), 63–75 (2001)
24. El Emam, K., Benlarbi, S., Goel, N., Rai, S.N.: Comparing Case-based Reasoning Classifiers for Predicting High Risk Software Components. *J. Systems and Software* 55(3), 301–320 (2001)
25. Youden, W.: Index for Rating Diagnostic Tests Cancer, vol. 3(1), pp. 32–35 (1950)
26. De Castro, L.N., Timmis, J.: *Artificial Immune Systems: A New Computational Intelligence Approach*. Springer, Heidelberg (2002)
27. Kim, J.W.: *Integration Artificial Immune Algorithms for Intrusion Detection*, PhD Thesis, University College London, Dept. of Computer Science (2002)
28. Watkins, A.: *AIRS: A Resource Limited Artificial Immune Classifier*, Master Thesis, Mississippi State University (2001)
29. Timmis, J.: *Artificial Immune Systems: A Novel Data Analysis Technique Inspired by the Immune Network Theory*, PhD Thesis, University of Wales, Aberystwyth (2001)
30. Timmis, J., Neal, M.: Investigating the Evolution and Stability of a Resource Limited Artificial Immune Systems. In: *Genetic and Evolutionary Computation Conference*, Nevada, USA, pp. 40–41 (2000)
31. Jerne, N.K.: Towards a Network Theory of the Immune System, *Ann Immunol.* 125C, pp. 373–389
32. Langman, R.E., Cohn, M.: The Complete Idiotypic Network is an Absurd Immune System. *Imm. Today* 7(4), 100–101 (1986)
33. Forest, S., Hofmeyr, S., Somayaji, A.: *Computer Immunology*. *Comm. of the ACM* 40(10), 88–96 (1997)
34. De Castro, L.N., Von Zuben, F.J.: The Clonal Selection Algorithm with Engineering Applications. In: *Genetic and Evolutionary Computation Conference*, pp. 36–37 (2000)
35. Carter, J.H.: The Immune System as a Model for Pattern Recognition and Classification, *Journal of American Medical Informatics Association*, 7(1) (2000)

36. Hamaker, J.S., Boggess, L.: Non-Euclidean Distance Measures in AIRS, an Artificial Immune Classification System, Congress of Evolutionary Computation (2004)
37. Watkins, A., Timmis, J.: Artificial Immune Recognition System (AIRS): Revisions and Refinements, ICARIS 2002, University of Kent, Canterbury, pp. 173–181 (2002)
38. Watkins, A.: Exploiting Immunological Metaphors in the Development of Serial, Parallel, and Distributed Learning Algorithms, PhD Thesis, Mississippi State University (2005)
39. Brownlee, J.: Artificial Immune Recognition System: A Review and Analysis, Technical Report. No 1-02, Swinburne University of Technology (2005)
40. Catal, C., Diri, B.: Software Defect Prediction Using Artificial Immune Recognition System. In: The IASTED Int'l Conference on Software Eng, Austria. pp. 285–290 (2007)

Operational Planning, Re-planning and Risk Analysis for Software Releases

Ahmed Al-Emran^{1,2,3} and Dietmar Pfahl^{1,3}

¹ Schulich School of Engineering, University of Calgary, Canada

² Software Engineering Decision Support Laboratory, University of Calgary, Canada

³ Centre for Simulation-based Software Engineering Research (CeSSER),

University of Calgary, Canada

{aalemran, dpfahl}@ucalgary.ca

Abstract. Software release planning takes place on strategic and operational levels. Strategic release planning aims at assigning features to subsequent releases such that technical, resource, risk and budget constraints are met. Operational release planning focuses on the realization of a single software release. Its purpose is to assign resources to feature development tasks such that total release duration is minimized under given process and project constraints. Re-planning becomes necessary on operational level due to addition or deletion of features during release development, or due to changes in the workforce. The allocation of resources to feature development tasks may depend on the accurate estimation of planning parameters such as feature size, developer productivity or development task dependencies. Risk analysis can help assess the vulnerability of a chosen release plan due to these dependencies. This paper presents a simulation-based approach to planning, re-planning and risk analysis of software releases on operational level. The core element of the approach is the process simulation model REPSIM-2 (Release Plan Simulator, Version 2). We describe the functionality of REPSIM-2 and illustrate its usefulness for planning, re-planning and risk analysis through application scenarios.

Keywords: software release planning, operational planning, process simulation, risk analysis.

1 Introduction

One of the key questions of incremental software development is to decide which features can be offered at which release. This decision depends on the customer needs, technological constraints, and the resources and time frame available to implement the features.

Software release planning takes place on strategic and operational levels. Strategic release planning aims at assigning features to subsequent releases such that technical, resource, risk and budget constraints are met. Once a strategic release plan has been generated, i.e., a decision has been made on which features are to be developed in which release, operational release planning focuses on the development of the identified features in a single software release. The purpose of operational release

planning is to assign resources to feature development tasks (e.g., design, implementation, and test) such that total release duration is minimized under given process and project constraints. Re-planning may become necessary on operational level for many reasons, for example, due to addition or deletion of features during release development, or due to changes in the workforce. The allocation of resources to feature development tasks may depend on the accurate estimation of planning parameters such as feature size, developer productivity or development task dependencies. Risk analysis can help assess the vulnerability of a chosen release plan due to these dependencies.

Good software release planning on both strategic and operational levels is extremely important [7]. A bad release plan may cause late delivery of high-value features, unsatisfied customers, budget overrun, and thus decreased competitiveness. Since many planning parameters and the features themselves are under continuous change [12], software release planning becomes a very dynamic task, and related decision-making problems are difficult to solve [6]. While satisfactory solutions for software release planning (and re-planning) on strategic level have recently been published in [3], [5], [10], more research is still needed to develop efficient and effective methods and tools in support of operational release planning.

The research presented in this paper focuses on a simulation model – REPSIM-2 (Release Plan Simulator, Version 2) – emphasizing on planning, re-planning, and risk analyses associated with operational release development plans of single releases. The presented simulation-based approach is applicable to any given solution to a specific strategic release planning problem. A solution to the strategic release planning problem, i.e., the assignment of features to subsequent releases, provides the following pieces of information for each individual release: number of features to be developed, number of task types needed to develop these features, number of developers available, estimates of the efforts needed per task type to develop a feature, estimates of the task type specific productivities of the available developers, and technical dependencies between subsequent task types. These pieces of information are needed to generate solutions to operational release planning and are referred to as problem parameters in the remainder of this paper. An introduction into existing methods supporting strategic release planning can be found in [11].

The remainder of this paper is structured as follows. Section 2 provides the motivation behind this research based on existing work performed in the area of software release planning. Section 3 describes the simulation model REPSIM-2. Section 4 illustrates the applicability and usefulness of REPSIM-2 with the help of a case example. Section 5 discusses achievements and limitations of REPSIM-2 and suggests directions for future research.

2 Related Work

Both optimization and simulation approaches have been proposed in the context of operational software release planning.

For example, OPTIMIZE_{RASORP} (Optimize Resource Allocation for Software Release Planning) is an optimization approach that generates simultaneously feature allocation plans for subsequent releases and operational feature implementation plans

for individual releases [5]. Thus it combines strategic and operational release planning. $\text{OPTIMIZE}_{\text{RASORP}}$ considers tasks associated with features, a pool of developers to carry out these tasks, the productivity of developers to perform these tasks, and mappings between tasks and developers for realization of features within releases and maximizing release value. While $\text{OPTIMIZE}_{\text{RASORP}}$ offers a guaranteed level of optimality regarding the allocation of resources in the context of operational release planning, it does not support automatic re-planning, e.g., re-allocation of developers in the middle of a release implementation due to changes in planning parameters.

$\text{OPTIMIZE}_{\text{RASORP}}$ is an example of a method that simultaneously formulates several release planning decision problems as one complex mathematical optimization problem. While the solution to this optimization problem has been shown to be optimal with regards to some objective function (cf. [5] for details), the formulation of the optimization problem is static in the sense that it does not allow for dynamic re-planning as easily as simulation-based approaches potentially do. For example, the scenario for re-planning discussed in Section 4 of this paper would be either impossible to solve or would require considerable effort to model them as part of the existing optimization-based approach. In addition, none of the existing mathematical optimization approaches can take into account non-deterministic situations, i.e., situations where values of model parameters are randomly sampled from empirically derived or assumed probability distributions. As a consequence, existing models applied to solve release planning problems via mathematical optimization cannot be used for risk analyses.

The first simulation model proposed to tackle issues in the context of strategic and operational software release planning was presented in [4]. Assuming a continuous stream of new incoming requirements, the model is used to decide how many requirements can be handled in a software release, and to investigate potential bottlenecks within subsequent releases. Bottlenecks are associated with task overload situations, i.e., situations in which the level of available resources assigned to specific tasks is too small to process incoming new (or from previous releases postponed) requirements. The model is also used to evaluate resource allocation changes that supposedly avoid previously identified overload situations. The problem dealt with in [4] is different from the problem focused on in this paper for two reasons, because it does not facilitate the analysis of specific developer allocations to feature/task-combinations within individual releases.

REPSIM-1 (Release Plan Simulator, Version 1)¹ [9] is a System Dynamics simulation model that can be used to perform risk analyses on existing operational release plans. Combined process and Monte-Carlo simulation as suggested in [8] is used to evaluate the sensitivity of existing plans to possible planning errors. Planning errors can relate to alterations in expected developer productivity, feature and task specific work volume (effort), and degree of task dependency. Risk analyses allow decision-makers to perform “what-if” analyses on a proposed operational release plan, helping them prepare for potentially required manual re-planning in the case of

¹ Note that the simulation model presented in [PAR06] is actually called REPSIM. In order to stress the fact that the simulation model presented in this paper is an enhancement of REPSIM, a version number has been added.

unplanned changes. However, automatic re-planning (and initial planning) is not supported.

DynaReP (Dynamic Re-Planner) is a discrete-event process simulation model developed using the tool EXTENDTM (<http://www.imaginetthatinc.com>). The model supports both initial generation and re-planning of operational release plans [2]. While initial plans generated by DynaReP are typically require about 5-10% more calendar time for the development of a single release of identical size and with the same available work force than plans generated with OPTIMIZE_{RASORP}, DynaReP can easily be used for automatic re-planning each time a change in planning parameters is identified during the development of a release.

REPSIM-2, which is described in more detail below, combines the planning and re-planning capabilities of DynaReP and the risk analysis capabilities of REPSIM-1. Given the possibility of REPSIM-2 to generate initial operational release plans (and doing dynamic re-planning), the risk analysis capabilities are used differently than in REPSIM-1, where release duration and developer allocation to feature/task-combinations are fixed. Instead of expressing risk in terms of work backlog, i.e., the amount of work that cannot be finished within the given constraints, REPSIM-2 expresses risk in terms of extended release duration.

3 The REPSIM-2 Model

REPSIM-2 is a process simulation model developed using the System Dynamics (SD) modeling and simulation tool VENSIMTM (<http://www.ventana.com>). SD is a simulation modeling technique originally developed in the late 1950s [For61]. It has been applied to the domain of software engineering since the late 1980s [AbM91]. REPSIM-2 supports the following planning, re-planning, and risk analysis tasks:

- Planning: Generating initial operational development plans for single software releases. Initial planning involves the assignment of developers to feature-specific development tasks such that the overall development time is as short as possible under the applied heuristic. Input data for initial planning are estimates of (1) required nominal efforts per feature and task, (2) productivity levels per developer and task, and (3) levels of dependencies between subsequent tasks per feature.
- Re-planning: Revising existing operational development plans for a single software release due to one or more of the following events:
 - A new feature needs to be included in a release.
 - A planned feature is removed from the release.
 - A developer becomes unavailable.
 - A developer is added to the development team.
 - The estimated task dependency is bigger/smaller than expected.
 - The work volumes of features were under-estimated/over-estimated.
 - The productivities of developers were over-estimated/under-estimated.
- Risk analysis: Analyzing initial (or revised) operational development plans for a single software release with regards to the sensitivity of duration and developer

allocation structure in response to variation of feature effort, developer productivity, and task dependency estimates.

The following sub-sections describe the heuristic used for assigning developers to feature-specific development tasks, the set of problem parameters and variables that characterize an operational release plan, and some implementation details of the simulation model REPSIM-2.

3.1 Model Heuristic

The heuristic used for assigning developers to feature/task-pairs essentially consists in matching the next available developer with the highest task-specific productivity to the next waiting feature with the largest effort (for a specific task). If only one developer with very low productivity is currently idle, then this mapping procedure can result in assigning a developer with low productivity to a large feature. To avoid such a worst case situation, a set of threshold variables are defined which exclude developers with productivity below a certain value to be assigned to feature/task-pairs. Details on the implementation of the model heuristic are given in Section 3.3.

3.2 Problem Parameters and Their Representation in the Model

Applying the heuristic described above, REPSIM-2 generates operational release development plans of a single software release by calculating a mapping from the 6-tuple $(F, T, D, eff, prod, dep)$ to the three dimensional matrix $F-T-D-alloc$, with the goal to minimize release duration. The problem parameters $F, T, D, eff, prod, dep$, and $F-T-D-alloc$ are defined as follows:

- $F = \{F[i] \mid 0 < i \leq f, \text{ with } i, f \in N\}$ is a set of f features, with N denoting the set of natural numbers.
- $T = \{T[j] \mid 0 < j \leq t, \text{ with } j, t \in N \wedge T[j-1] \ll T[j] \text{ for } j > 1\}$ is a set of t completely ordered development task types (in the following denoted as “tasks”). The order “ \ll ” defined on T specifies a start-to-start and end-to-end dependency between two subsequent tasks $T[j-1]$ and $T[j]$, i.e., task $T[j]$ cannot start before task $T[j-1]$ has started, and task $T[j]$ cannot end before task $T[j-1]$ has been completed. Typical examples of subsequent tasks are design, implementation, and test.
- $D = \{D[k] \mid 0 < k \leq d, \text{ with } k, d \in N\}$ is a set of d developers.
- $eff: (F, T) \rightarrow R_0^+$, with R_0^+ denoting the set of non-negative real numbers, is a function that assigns an estimated work load (volume) to each feature/task-combination. The volume may be specified, for example, in terms of person-weeks.
- $prod: (D, T) \rightarrow R_0^+$ is a function that assigns an estimated relative productivity factor to each developer/task-combination. For example, if the productivity factor of a specific developer/task-combination $(D[k], T[j])$ equals y and the effort for a specific feature/task-combination $(F[i], T[j])$ equals x person-weeks, then developer $D[k]$ is expected to be able to perform the task $T[j]$ of feature $F[i]$ in x/y weeks. A productivity factor that equals 0 for a specific developer/task-combination $(D[k],$

$T[j]$) implies that developer $D[k]$ is not able to perform task $T[j]$, no matter which feature is affected.

- $dep: DEP \subseteq T \times T \rightarrow [0, 1]$, where $DEP = \{(T[p], T[q]) \mid T[p], T[q] \in T \wedge T[p] \ll T[q]\}$ is a function that further refines the order “ \ll ” defined on T . REPSIM-1 can be adjusted to the following three different versions of dep :
- $dep_1(T[j-1], T[j]) = x \in [0, 1]$ specifies that work on task $T[j]$ can only start, if at least x times $eff(T[j-1])$ has been completed.
- $dep_2(T[j-1], T[j]) = x \in [0, 1]$ specifies that for task $T[j]$ the work on x times $eff(T[j])$ can only start, if task $T[j-1]$ has been completed. For task $T[j]$ the work on $(1-x)$ times $eff(T[j])$ can start as soon as work on task $T[j-1]$ has started.
- $dep_3(T[j-1], T[j]) = x \in [0, 1]$ specifies that for task $T[j]$ the work on x times $eff(T[j])$ can only start, if at least x times $eff(T[j-1])$ has been completed. For task $T[j]$ the work on $(1-x)$ times $eff(T[j])$ can start as soon as work on task $T[j-1]$ has started.
- For $0 < x < 1$: dep_3 is less restrictive than dep_1 and dep_2 . For $x = 0$ and $x = 1$: $dep_1 = dep_2 = dep_3$. In the following, only the implementation of REPSIM-2 using task dependency type dep_1 will be presented. An older implementation of REPSIM-2 using task dependency dep_3 has been described in [9].
- $F-T-D-alloc$ is a function that at each point in time assigns a specific developer to a specific feature/task-combination. An important constraint of developer allocation requires that one developer can only be allocated to one feature/task-combination at a time.

The above listed parameters, characterizing the operational release planning problem, are represented in REPSIM-2 through the following model subscripts², parameters, and result variable:

- Feature: F_1, F_2, \dots is a subscript representing the set of features F .
- Task: T_1, T_2, \dots is a subscript representing the set of tasks T .
- Developer: D_1, D_2, \dots is a subscript representing the set of developers D .
- $Eff-F-T[Feature, Task]$ is a 2-dimensional matrix constant representing the values of function eff for each feature/task-combination.
- $Prod-T-D[Task, Developer]$ is a 2-dimensional matrix constant representing the values of function $prod$ for each task/developer-combination.
- $Task-Dependency[Task]$ is a vector representing the degree of dependency between subsequent tasks as defined by the function dep . This parameter could be defined to be also feature/task-combination specific if a second subscript (i.e., Feature) was added.
- $Alloc-F-T-D[Feature, Task, Developer]$ is a 3-dimensional matrix representing the values of function $F-T-D-alloc$ for each feature/task/developer-combination. The values in each cell of this matrix can either be “0” or “1”, where “1”

² In order to facilitate the individual representation of multiple features, development tasks, and resources (developers), the implementation of REPSIM-2 made extensive use of subscripting offered by VENSIMTM. The use of subscripts keeps the model compact while making it scalable.

indicates that a developer has been assigned to a feature/task-combination during specified time steps.

An example of an operational release plan with 8 features, 3 tasks, and 6 developers is shown Figure 1. It uses the estimated task-specific efforts per feature and assumed task-specific productivities per developer shown in Tables 1(a) and 1(b), respectively.

Assignment of developers to feature/task-combinations:

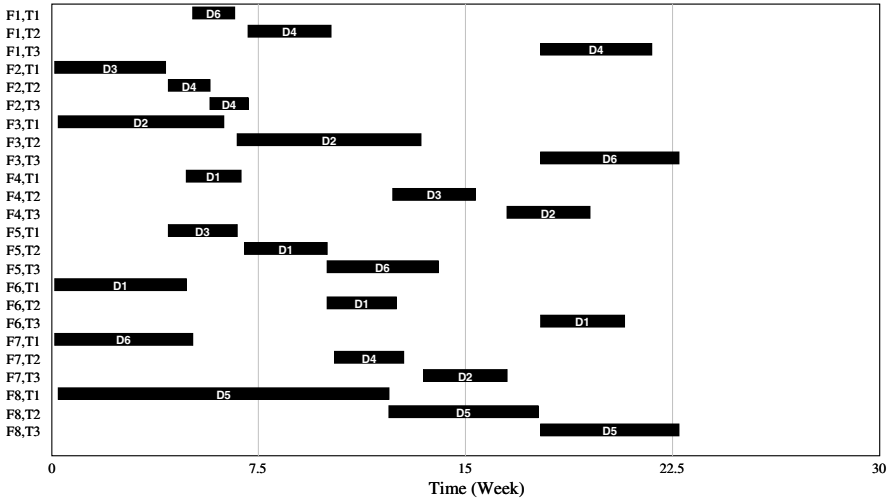


Fig. 1. Example of an operational release plan of a single release

Table 1. (a) Task-specific efforts per feature; (b) Task-specific productivities per developer

		Effort Estimates [person-week]								Productivity Estimates [dimensionless]					
		F1	F2	F3	F4	F5	F6	F7	F8	D1	D2	D3	D4	D5	D6
Task Type	T1: Design	3	8	6	3	5	7	10	6	1.5	1	2	0	0.5	2
	T2: Implementation	6	3	10	3	6	5	5	8	2	1.5	1	2	1.5	1
	T3: Test	6	2	5	6	4	3	6	10	1	2	0	1.5	2	1

3.3 Model Structure

Figures 2 and 3 present the core structure of the REPSIM-2 model using the graphical modeling language provided by VENSIM™, hiding most of the auxiliary variables used for intermediate calculations and calculations of specific output values.

Figure 2 shows the view of the model structure that generates the work-flow dynamics of an operational release development plan in REPSIM-2.

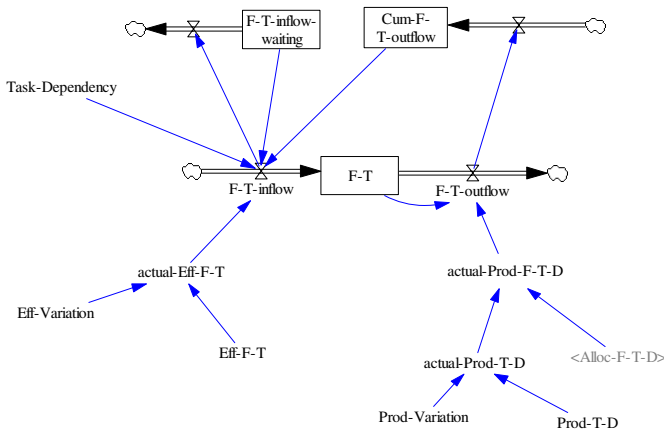


Fig. 2. Flow-graph representation of the release plan work-flow in REPSIM-2

The variable $F-T[Feature, Task]$ is at the heart of the model. It represents the volume (in terms of effort) of a feature/task-combination currently being worked on. Its quantity is controlled by the inflow and outflow rates $F-T-inflow[Feature, Task]$ and $F-T-outflow[Feature, Task]$, respectively, using the following integral equation:

$$F_T[i, j](t) = \int_0^t (F_T_inflow[i, j](u) - F_T_outflow[i, j](u)) du$$

The following model variables are shown in Figure 2:

- $F-T[Feature, Task]$: a 2-dimensional matrix of accumulation variables (often called “levels” or “stocks” in the SD literature) representing for each feature/task-combination the work load that is ready to be worked on.
- $F-T-inflow[Feature, Task]$: a 2-dimensional matrix of adjustment variables (often called “rates” or “valves” in the SD literature) that control for each feature/task-combination by how much new volume the corresponding cell of variable $F-T[Feature, Task]$ increases per time step.
- $F-T-outflow[Feature, Task]$: a 2-dimensional matrix of adjustment variables that control for each feature/task-combination by how much volume the corresponding cell of variable $F-T[Feature, Task]$ decreases per time step. If $F-T-outflow[Feature, Task]$ is greater than 0 for a specific feature/task-combination, actual work is done in relation to this feature/task-combination.
- $Cum-F-T-outflow[Feature, Task]$: a 2-dimensional matrix of accumulation variables representing the volumes of a feature/task-combination that have been completed.
- $F-T-inflow-waiting[Feature, Task]$: a 2-dimensional matrix of accumulation variables representing the volumes of feature/task-combinations that are waiting to be ready such that they can be worked on.

At the beginning of a simulation, the variable $F-T-inflow-waiting[Feature, Task]$ is initialized by assigning the total work loads of each feature/task-combination. When

and how much of the waiting volumes are ready, i.e., moved into the corresponding cells of variable F-T-[Feature, Task], depends mainly on Cum-F-T-outflow[Feature, Task] and the type of task dependency. For example, if task dependency dep_1 has been selected with a value of 0.3, quantities stored in variable F-T-inflow-waiting[Feature, Task] will be moved into variable F-T[Feature, Task], only if the values stored in variable Cum-F-T-outflow[Feature, Task] that correspond to predecessor tasks of respective features have achieved a value of at least 30% of their estimated volume.

In order to be able to perform various kinds of stability analyses on an existing release plan, REPSIM-2 offers the following parameters to model users:

- Eff-Variation[Feature] is a vector of parameters (i.e., model constants) representing an adjustment factor that, if different from 1, modifies the original effort estimates related to each individual feature. This factor could be defined to be also task-specific if a second subscript (i.e., Task) was added.
- Prod-Variation[Developer] is a vector of parameters representing an adjustment factor that, if different from 1, modifies the original productivity estimates related to each individual developer. Again, this factor could be defined to be also task-specific by adding a second subscript (i.e., Task).

The auxiliary variables actual-Eff-F-T, actual-Prod-T-D, and actual-Prod-F-T-D are needed to calculate modifications of the initial estimates for feature/task-combination specific effort estimates and task/developer-combination specific productivity levels.

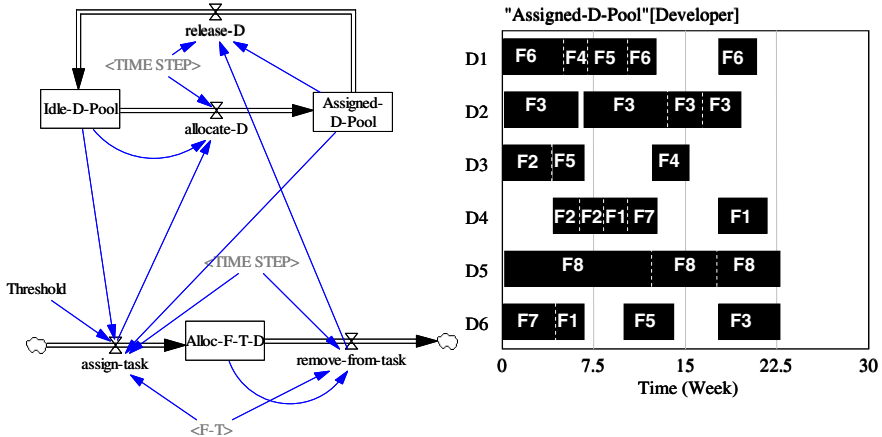


Fig. 3. Flow-graph representation of the heuristic used to assign developers to feature-specific tasks in REPSIM-2

Figure 3 shows the implementation of the underlying model heuristic according to which developers are assigned to feature/task-combinations.

Accumulation variables Idle-D-Pool[Developer] and Assigned-D-Pool[Developer] are vectors representing the set of developers that are idle or assigned to a feature/task-combination. A developer can only be assigned to a new task, when the

respective value of Idle-D-Pool[Developer] equals “1”. As soon as a developer is assigned, the value of the corresponding cell in Idle-D-Pool switches from “1” to “0”, while the respective value of Assigned-D-Pool[Developer] switches from “0” to “1”. As soon as an assigned developer has finished a task, the respective values are toggled once more. The Gantt chart on the right hand side of Figure 3 shows an example allocation of developers to features (without explicitly specifying the task type).

Level variable Alloc-F-T-D[Feature, Task, Developer] equals “1” as long as a developer is assigned to a specific feature/task-combination, otherwise “0”. The following conditions have to be fulfilled, before a specific cell of the three-dimensional matrix Alloc-F-T-D can switch from “0” to “1” at a point in time:

- There is work waiting to be done for any features/task-combination ("F-T"[Feature,Task]>0)
- The work volume (effort) of the waiting feature/task-combination is the maximum of all waiting feature/task-combinations ("Eff-Rank-F-T2"[Feature,Task]=VMAX("Eff-Rank-F-T2-Unalloc"[Feature!,Task]))
- The productivity of a candidate developer is greater than the threshold productivity ("actual-Prod-T-D"[Task, Developer]>Threshold[Task])
- A candidate developer with sufficient productivity is actually idle ("Idle-D-Pool"[Developer]>0:AND: "Assigned-D-Pool"[Developer]<1)
- If several equally large feature/task-combinations for the same feature but different tasks are subject to be assigned the same developer at the same point in time, the predecessor task will be worked on first (Test[Task, Developer]>0))

An important model parameter needed to implement the model heuristic is Threshold[Task], a vector of productivity threshold values used to restrict the availability of developers per task type. For example, if the model has design, implementation, and test tasks, the vector has three cells. For a specific type of task, if a developer does not possess a productivity value higher than that of the corresponding Threshold parameter, then that developer will not be allowed to carry out that type of task. Instead of setting the Threshold parameter manually, REPSIM-2 can be instrumented to use the optimization functionality offered by VENSIMTM, automatically assigning values to each task-specific productivity threshold value such that the overall duration of a calculated release plan becomes minimal. Note that values for these parameters have to be re-calculated at each time a change is made in the planning parameters (e.g., addition of a new feature or drop-out of a developer).

REPSIM-2 contains additional auxiliary variables which are either used for intermediate calculations or for calculations of specific output values. Due to space limitations all auxiliary variables cannot be explained in detail here. The complete set of model equations will be provided on request by the author.

4 Example Application Scenarios

Using the planning example presented in Section 3.2 as a starting point (baseline scenario), in this section we illustrate some of the re-planning and risk analysis

capabilities of REPSIM-2 (cf. introduction of Section 3 for a complete list of capabilities). The baseline scenario had the following characteristics:

- 8 features to be developed: F1, F2, ..., F8.
- 3 tasks to be carried out for each feature: T1, T2, and T3 (e.g., design, implementation, and test, respectively).
- 6 developers available to work on each feature-specific tasks: D1, D2, ..., D6.
- The estimated work volume (in person-weeks) for each feature-specific task and the estimated productivity of each developer per task type (cf. Table 1).
- Task dependency is 100%, i.e., for each feature a subsequent tasks can only be started when the predecessor task has been completed.

Effort estimates may be acquired via expert interviews or by using effort estimation techniques. Productivity estimates may be based on performance data from past projects, ideally in combination with skill analyses [1].

Using REPSIM-2, the operational release development plan is estimated to have a total duration of 22.71 person-weeks when using the developer allocation to feature/task-combinations shown in Figure 1.

4.1 Scenario 1: Re-planning Due to Late Feature Inclusion

Due to space limitations, only one typical re-planning situation will be presented here: addition of a late feature when development activities on an initially defined set of features have already started.

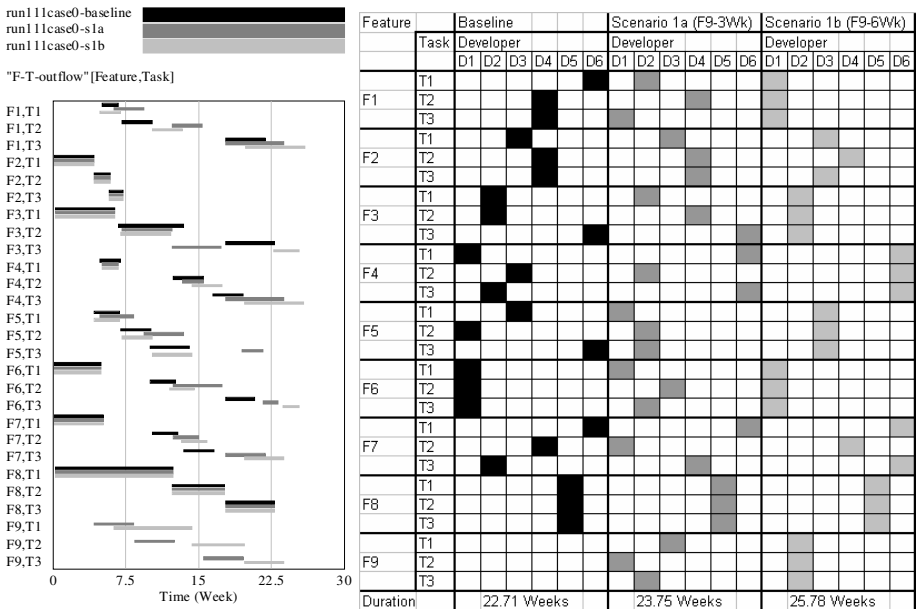


Fig. 4. Feature F9 is included in a release under development after 3 and 6 weeks, respectively

We assume that the development organization started working according to the initial plan presented above and in Section 3 (baseline scenario). After some time has elapsed, a new feature, F9, is required by the customer to be included in the current release. We also assume that the estimated work volume (effort) for each of the tasks T1, T2, and T3 of F9 equals 8 person-weeks. Figure 4 shows how the allocation of developers to feature/task-combinations changes, if F9 is added after 3 weeks (Scenario 1a) and 6 weeks (Scenario 1b), respectively. As is to be expected, the total duration of the release increases: from 22.71 weeks to 23.75 weeks and 25.78 weeks, respectively (cf. left-hand side of Figure 4). In addition, the developer allocation patterns change for most of the features (cf. right-hand side of Figure 4).

It is interesting to see that the relationship between the point in time when a new feature is added and the overall duration of the (larger) release is not linear. Adding feature F9 after 3 weeks (Scenario 1a) increases the total duration of the release increases by only 1.04 weeks (+4.6%), while adding feature F9 after 6 weeks (Scenario 1b) increases the total duration already by 3.08 weeks (+13.6%). This jump in duration increase becomes even larger, when observing that work on feature F9 only starts shortly after end of week 4 in Scenario 1a. The delay by 1 week is due to the fact that none of the five developers with design productivity greater than 0 finishes work on features F2, F3, F6, F7, or F8 before end of week 4 and thus cannot be assigned to a new task.

It should be noted that REPSIM-2 can handle situations with more than one feature added (or deleted) at different points in time. Moreover, addition/deletion of features can be combined with increase/decrease of number of developers and/or changes in effort and productivity estimates at any points in time, thus providing a powerful decision-support tool.

4.2 Scenario 2: Risk Analysis

A common situation in release development projects is uncertainty about the accuracy of effort estimates and developer productivity. The effects of estimation errors on total project duration can be assessed by running stochastic simulation. Instead of assigning deterministic values to model parameters representing task-specific effort estimates for features (model parameter $\text{Eff-F-T}[\text{Feature}, \text{Task}]$) and task-specific productivities of developers (model parameter $\text{Prod-T-D}[\text{Task}, \text{Developer}]$) these values can be sampled from plausible input distributions. The process simulation model REPSIM-2 can be executed in stochastic mode using multivariate Monte-Carlo simulation.

The results from running the baseline scenario introduced in Section 3 in stochastic mode are shown in Table 2. The baseline case represents the deterministic simulation run without variation of feature/task effort estimates and without variation of task-specific developer productivity estimates. In the baseline case, the minimum, maximum, mean and median durations equal 22.71 weeks. Standard deviation and normalized standard deviation (last column of Table 2) equal 0. Scenarios 2a-e represent each 50 simulation runs with different probability distribution sampling patterns for effort and productivity estimates. Both parameters, Eff-F-T and Prod-T-D

are sampled from triangle distributions of the type TRIANG(min, peak, max), where *min* represents the minimum value, *max* represents the maximum value, and *peak* represents the most probable value. The following TRIANG distributions were used in Scenarios 2a to 2e, respectively: TRIANG(0.9, 1, 1.1), TRIANG(0.9, 1, 1.2), TRIANG(0.8, 1., 1.2), TRIANG(0.8, 1. 1.3), TRIANG(0.7, 1, 1.3).

Table 2 shows the average values over all feature/task and developer/task combinations of all simulation runs (per scenario). One can see that the normalized variation of release duration has about the same magnitude (column (Norm)) as the normalized variation of the model input parameters Eff-F-T and Prod-T-D, if the normalized variation is below 0.1. For larger input variations it seems that a damping effect occurs. For example, Scenario 2e has normalized input variations of 0.1209 and 0.1308 for parameters Eff-F-T and Prod-T-D, respectively, while release duration has only a normalized variation of 0.1004. This damping effect may partly be due to mutual compensation between over-estimated and under-estimated effort volumes and productivities. In addition, extreme effort and productivity over/under-estimation might partly be compensated by varying developer allocations to feature/task-combinations. The degree of structural variation, similar to the variation in developer assignment presented in Figure 4 (table on right hand side), has not yet been further analyzed, but it can be assumed based on analysis of individual cases (not shown here due to space limitations) that the structure of developer assignments can change strongly in response to small changes in effort and productivity estimates.

Table 2. Impact of variation in effort and productivity estimates on total duration

Scenario	#Runs	Parameter	Min	Max	Mean	Median	StDev	(Norm)
Baseline	1	Eff. Var.	1	1	1	1	0	0
		Prod. Var.	1	1	1	1	0	0
		Duration	22.71	22.71	22.71	22.71	0	0
Scenario 2a	50	Eff. Var.	0.9199	1.0824	1.0006	1.0007	0.0404	0.0404
		Prod. Var.	0.9117	1.0885	0.9977	0.9947	0.0433	0.0434
		Duration	21.0625	27.5000	23.0194	23.0625	1.0851	0.0471
Scenario 2b	50	Eff. Var.	0.9244	1.1694	1.0344	1.0286	0.0618	0.0597
		Prod. Var.	0.8203	1.0859	0.9626	0.9641	0.0661	0.0687
		Duration	21.5313	27.5938	24.5413	24.4688	1.4198	0.0579
Scenario 2c	50	Eff. Var.	0.8399	1.1647	1.0012	1.0014	0.0808	0.0807
		Prod. Var.	0.8234	1.1770	0.9955	0.9894	0.0866	0.0870
		Duration	19.4688	29.6250	23.0375	22.7813	2.0589	0.0894
Scenario 2d	50	Eff. Var.	0.8446	1.2517	1.0350	1.0289	0.1018	0.0983
		Prod. Var.	0.7320	1.1742	0.9603	0.9594	0.1090	0.1135
		Duration	20.0938	29.6250	24.8225	24.6875	2.1110	0.0850
Scenario 2e	50	Eff. Var.	0.7598	1.2471	1.0018	1.0021	0.1212	0.1209
		Prod. Var.	0.7351	1.2654	0.9932	0.9841	0.1299	0.1308
		Duration	18.0313	28.5938	23.1494	23.0469	2.3240	0.1004

5 Conclusions and Future Work

In this paper, we presented the design and application of the simulation model REPSIM-2. This model is an enhanced version of the simulation model REPSIM-1 (Release Plan Simulator, Version 1), which can be used for analyzing the sensitivity of defined operational software release plans to changes in developer productivity, task-specific feature effort estimates, and task dependencies. A major disadvantage of REPSIM-1 was its lacking capability to generate operational software release plans. REPSIM-2 has overcome this shortcoming by integrating the planning heuristic of the simulation model DynaReP.

REPSIM-2 can be used to support operational planning, re-planning, and risk analyses tasks in the context of the realization of a single software release. One of the strengths of REPSIM-2 is its scalability. New release planning problems with different numbers of features, developers and task types can easily be accommodated by a simple change of subscript value ranges. Also, new types of dependencies, e.g., dependencies between features or developers, or 1-to-n relationships between subsequent tasks could easily be accommodated by re-formulating or enhancing conditions used in one or more model equations. Thanks to the highly reusable model structure, no structural changes in the model would be required in these cases.

It should be mentioned that the example application presented in Section 4 was kept small in order to be able to show a complete set of results in the limited space available. The same types of analyses were actually applied to five larger cases involving up to 65 features and 13 developers. It should also be pointed out that the effort and productivity estimates used in these cases were directly taken from industrial applications conducted in the context of strategic release planning.

The value of REPSIM-2 is two-fold. Firstly, it supports decision-makers in solving complex re-planning problems emerging from any combination (and possibly repeated occurrence) of changes in number of features, number of available developers, task dependencies, and effort or productivity estimates. Secondly, it supports decision-makers in assessing the risk associated with inaccurate estimates of efforts, productivities and task-dependences. The second point is particularly useful as most of these estimates are mostly based on subjective expert estimates and thus planning errors are likely to happen. Based on risk analyses conducted with REPSIM-2, decision-makers can at least anticipate where potential re-planning is likely to occur in response to observed under- or over-estimation of efforts, productivities, and task-dependencies.

The biggest limitation of REPSIM-2 is that it cannot guarantee optimal (initial or revised) plans. As pointed out earlier in the paper, optimization methods such as OPTIMIZE_{RASORP} generate operational release plans that are typically 5-10% more effective, e.g., in terms of release development duration. However, since these optimization methods can neither be directly used for dynamic re-planning nor for stochastic (or sensitivity) analyses, REPSIM-2 offers an interesting alternative for situations where frequent changes in planning parameters are likely to occur.

Future work on improving REPSIM-2 will focus on (i) enhancing the model heuristic in order to improve effectiveness; (ii) including feature dependency constraints that specify whether a feature must be realized before another feature; (iii)

improving model usability (e.g., data input via GUI, connection to external database, etc.); and (iv) validating the proposed approach in an industrial environment.

Acknowledgements

Part of the work presented was financially supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada under Discovery Grant no. 327665-06.

References

1. Acuña, S.T., Juristo, N., Moreno, A.M.: Emphasizing human capabilities in software development. *IEEE Software* 23(2), 94–101 (2006)
2. Al-Emran, A.: Dynamic Re-Planning of Software Releases, Master Thesis, University of Calgary (2006)
3. Alboutrae, T., Ruhe, G., Moussavi, M.: Lightweight Replanning of Software Product Releases. In: *Proceedings of International Workshop on Software Product Management, Minneapolis/St. Paul, Minnesota, USA* (2006)
4. Höst, M., Regnell, B., Dag, J., Nedstam, J., Nyberg, C.: Exploring Bottlenecks in Market-Driven Requirements Management Processes with Discrete Event Simulation. *Journal of Systems and Software* 59(3), 323–332 (2001)
5. Ngo-The, A., Ruhe, G.: Optimized Resource Allocation for Incremental Software Development. TR 062/2006, Laboratory for Software Engineering Decision Support, University of Calgary (2006)
6. Momoh, J.: Applying Intelligent Decision Support to Determine Operational Feasibility of Strategic Software Release Planning. Masters thesis, Department of Electrical and Computer Engineering, University of Calgary, Canada (2004)
7. Penny, D.A.: An Estimation-Based Management Framework for Enhance Maintenance in Commercial Software Products. In: *Proceedings of International Conference on Software Maintenance*, pp. 122–130 (2002)
8. Pfahl, D.: ProSim/RA – Software Process Simulation in Support of Risk Assessment. In: Biffel, S., et al. (ed.) *Value-based Software Engineering*, pp. 263–286. Springer, Berlin (2005)
9. Pfahl, D., Al-Emran, A., Ruhe, G.: Simulation-Based Stability Analysis for Software Release Plans. In: Wang, Q., Pfahl, D., Raffo, D.M., Wernick, P. (eds.) *Software Process Change SPW/ProSim 2006*. LNCS, vol. 3966, pp. 262–273. Springer, Berlin-Heidelberg (2006)
10. Ruhe, G., Ngo-The, A.: Hybrid Intelligence in Software Release Planning. *International Journal of Hybrid Intelligent Systems* 1(2), 99–110 (2004)
11. Ruhe, G., Saliu, O.: The Art and Science of Software Release Planning. *IEEE Software* 22(6), 47–53 (2005)
12. Stark, G., Skillicorn, A., Ameen, R.: An Examination of the Effects of Requirements Changes on Software Maintenance Releases. *Journal of Software Maintenance: Research and Practice* 11, 293–309 (1999)

Project Cost Overrun Simulation in Software Product Line Development

Makoto Nonaka¹, Liming Zhu², Muhammad Ali Babar³, and Mark Staples²

¹ Faculty of Business Administration, Toyo University, Japan
nonaka-m@toyonet.toyo.ac.jp

² National ICT Australia

{liming.zhu,mark.staples}@nicta.com.au

³ Lero, University of Limerick, Ireland

Muhammad.alibabar@ul.ie

Abstract. The cost of a Software Product Line (SPL) development project sometimes exceeds the initially planned cost, because of requirements volatility and poor quality. In this paper, we propose a cost overrun simulation model for time-boxed SPL development. The model is an enhancement of a previous model, specifically now including: consideration of requirements volatility, consideration of unplanned work for defect correction during product projects, and nominal project cost overrun estimation. The model has been validated through stochastic simulations with fictional SPL project data, by comparing generated unplanned work effort to actual change effort, and by sensitivity analysis. The result shows that the proposed model has reasonable validity to estimate nominal project cost overruns and its variability. Analysis indicates that poor management of requirements and quality will almost double estimation error, for the studied simulation settings.

Keywords: process simulation, cost overrun estimation, software product line development.

1 Introduction

Software Product Line (SPL) development can shorten the total cycle time, the duration from the beginning of core asset development to the end of product development, by applying large-scale reuse [1]. However, effort estimation, planning, and development management for SPL are more complex and difficult than those for sequential development, because of inter-connected relationships between core assets and products, concurrency of their projects, and multiple deadline management [2]. In addition, there are still general problems with software effort estimation because of unplanned work [3] and requirements volatility [4]. The total cycle time can sometimes be longer than initially planned because of these problems.

Requirements volatility is the tendency of requirements to change over time. High requirements volatility has a large impact on cost and effort overruns [5]. Some unexpected critical requirements changes are in practice unavoidable, and

can for example be caused by faults on external components to be compensated by software, or by marketing issues requiring new functionality to catch up with other competitive products.

Quality problems are also important factors for cost overruns. A certain number of defects will inevitably remain in released software products, as software testing can not demonstrate the absence of defects [6]. When residual defects in core assets are detected after their release to product projects (not to customers), corrective maintenance is usually performed to modify the core assets. When multiple product projects are undertaken simultaneously during core asset maintenance phase, corrective maintenance in core assets sometimes brings associated rework to all ongoing product projects that depend on the core assets, to adapt the products to the changed core assets. We have called this type of rework “adaptive rework” [7].¹ In addition, each product project will also be delayed caused by defects injected during the project.

Though the reasons why software effort estimation error appears have been shared among software professionals [3,9], it is still difficult to predict the amount of overrun and its variability, or the level of risk, in specific situations. The variability of effort estimation gives us more useful information than traditional minimum-maximum intervals to indicate its uncertainty [10]. This can be estimated by a simulation approach. With regard to the quality problem in core assets, we previously proposed a simulation model for estimating project delay and its variability in SPL development [7]. Though the model was validated to be capable of estimating reasonable project delay and its variability, it did not consider requirements volatility and rework caused by defects injected during product projects as sources of project delay.

Literature shows that avoiding project delay is sometimes considered to be a high priority for project success [11]. For such projects, delay will be avoided even though much additional cost is required. Cost overrun estimation for time-boxed projects therefore should be studied. However, in our previous model, any piece of adaptive rework causes project delay, which in practice may be resolved without delay but with additional cost.

Even in the literature concerning effort estimation and simulation in SPL development, these problems have not been sufficiently considered [2,12,13,14,15]. In this paper, we propose a project cost overrun simulation model for time-boxed SPL development by enhancing our previous model. The major enhancements from the previous model include: consideration of requirements volatility, consideration of unplanned work for defect correction during product projects, and nominal project cost overrun estimation. A homogeneous Poisson process is introduced to represent requirements volatility. We use a similar technical approach as in our previous model to represent effort of defect correction during product projects. We estimate nominal cost overruns in month scale instead of

¹ The meaning of “adaptive rework” in this paper and that of “adaptive maintenance” in an IEEE standard [8] are somewhat different. Adaptive maintenance is defined in [8] as “modification of a software product performed after delivery to keep a computer program usable in a changed or changing environment.”

person-month scale, which makes our previous model applicable to time-boxed projects. We conduct stochastic simulations with fictional project data by using the enhanced model. The following are the research questions to be explored. *How much cost overrun and its variability in SPL development are expected when (a) requirements change in product projects, and (b) product quality changes due to residual defects in core assets and defects injected during product projects?*

The remainder of this paper is organized as follows. Sect. 2 describes the proposed simulation model, which includes the previous model and the enhanced features. Simulation results and derived implications are described in Sect. 3. Sect. 4 discusses model evaluation. Sect. 5 contains a discussion and describes related work. Concluding remarks are described in Sect. 6.

2 Project Cost Overrun Simulation Model

2.1 Assumed SPL Development and Unplanned Work Types

SPL development involves two types of development activities²: core asset development and product development. Core projects create common assets to be reused by products. Core assets are maintained during a core asset maintenance phase to correct residual defects in core assets. Product projects create products by instantiating core asset variation points and by adding individually required functionality. We assume that total product development effort is much larger than total effort of core projects, which is considered to be non-matured SPL development [16].

We also assume that multiple product projects can be undertaken simultaneously, if products are considered independent of each other. In this situation, at least the following three types of unplanned work depicted in Fig. 1 will occur, which affect project cost and schedule overruns:

1. adaptive rework in product projects caused by residual defects in core assets,
2. requirements changes for products, and
3. defect correction in product projects before release.

Note that requirements change and defect correction for core projects are not considered here, as we assume the impact of core projects on total cost overruns is small for non-matured SPL development. Requirements changes occurring after core assets release are considered to be incorporated into the next core project.

The work type “adaptive rework” has been already considered in our previous work [7], which is summarized in the next section. In Sect. 2.3 and 2.4, we explain effort models for the other two types of unplanned work, which are the enhancements from the previous model.

² Another type of activity “managing the SPL as a whole” is described in [1]. We only represent development activities in our model, though the model itself may contribute to an improved understanding of SPL management.

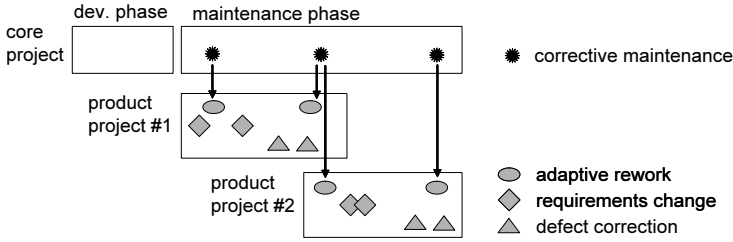


Fig. 1. Assumed SPL development and unplanned work

2.2 Adaptive Rework Effort Model

To determine total adaptive rework effort, the frequency of adaptive rework and its effort are considered. The frequency is closely correlated with the number of residual defects in core assets. The effort of each piece of adaptive rework will in practice relate to the strength of dependency between core assets and products. This assumption is partly supported by [17][18][19] showing that design complexity has a large influence on maintenance effort. The duration will also relate to what development phase it occurs in. Literature reports that the ratio of the cost of finding and fixing a defect during design, test, and field use is 1 to 13 to 92 [20] or 1 to 20 to 82 [21].

From this discussion, we select the following factors to determine the effort of adaptive rework. Note that we do not assume any specific methods to estimate or measure these factors.

1. *The number of residual defects in core assets* (NRD_{core}). It will depend on product size, product complexity, process quality, and other factors. We assume that NRD_{core} can be estimated.
2. *The strength of dependency* (DEP). We consider DEP between core assets and products as well as among core assets. It is represented as a continuous variable that ranges from 0 to 1. $DEP = 0$ means no dependency, and $DEP = 1$ means the strongest. In practice, there may be different levels of dependency for different changes, but we use a single DEP value to represent the worst-case dependency. DEP might reflect attributes such as coupling between core components and product components, or the number of dependent product components reusing a core component.
3. *Work effort multiplier* (WEM). We introduce WEM to represent the ratio of the effort of pieces of adaptive rework for each development phase in which adaptive rework occurs. We assume that each product project follows sequential processes. WEM can be estimated by investigating organizational defect modification data. It is represented as a continuous variable that ranges from 0 to 1, and used to calculate the effort of adaptive rework.
4. *Effort distribution of worst case adaptive rework* ($EffDist_{wcar}$). Worst case adaptive rework is supposed to represent the adaptive rework in the following worst-case settings: the defect correction completion time is at the end of the

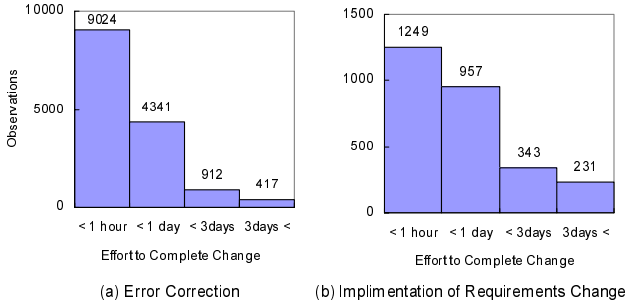


Fig. 2. Change effort distributions from SEL data [22] drawn by the authors

product project, and DEP is the strongest. We use a probability function for representing an EffDist_{wcar} . We assume that an EffDist_{wcar} has a right-skewed distribution, which is based on the Software Engineering Laboratory (SEL) data subset [22] showing that an effort distribution for error correction is right-skewed as depicted in Fig. 2(a). The range of the EffDist_{wcar} will be wider than the SEL data distribution, as it represents worst cases of adaptive rework instead of actual effort. It may also have a larger variability than the SEL data distribution, as some residual defects bring multiple changes in a product.

With these parameters, the effort of adaptive rework can be determined as follows. The nominal effort of adaptive rework $\Delta\text{Eff}_{ar-i}(d_j)$ (in months) caused by the defect d_j in the project i is assumed to be represented by the formula

$$\Delta\text{Eff}_{ar-i}(d_j) = \text{EffDist}_{wcar-i}^{-1}(p) \times \text{WEM}_i(t_{d_j}) \times \text{DEP}_{ki} \times \epsilon, \quad (1)$$

where $\text{EffDist}_{wcar-i}^{-1}(p)$ is the inverse function of EffDist_{wcar} for the project i . The probability p is given at random. WEM for the project i is represented with $\text{WEM}_i(t_{d_j})$ when the defect d_j correction is completed in core asset maintenance phase at the time t_{d_j} . DEP between the core assets k and the product i (or core assets i) is represented with DEP_{ki} . The parameter ϵ is 1 if t_{d_j} is within the period of the project i . Otherwise, ϵ is 0.

The defect correction completion time t_d can be determined by applying a Software Reliability Growth Model (SRGM) [23]. Suppose that all residual defects in core assets which bring adaptive rework are detected during core asset maintenance phase. If we draw an SRGM curve during the phase, the defect correction completion time of these defects can be determined by assigning a time to each defect along with the curve depending on reliability growth. An SRGM curve can therefore be considered to represent the organization’s capability for defect detection.

Note that the scale of nominal adaptive rework effort is in months, not in person-months. As our simulation model does not consider the number of resources as a parameter, we use nominal effort with a month scale. Though

this does not represent the absolute effort overruns, we can compare relative magnitude of the effort.

2.3 Requirements Change Effort Model

To determine the total requirements change effort, we consider as parameters (1) the number of unavoidable critical requirements change requests, (2) the change request arrival time, and (3) the effort of each piece of requirements change.

To represent (1) and (2), we consider that a homogeneous Poisson process is applicable. A homogeneous Poisson process is a stochastic process which is defined in terms of the occurrences of events with a known average event occurrence rate. It is widely used to represent discrete event arrivals in simulation studies. However, we do not have any supporting evidence showing that requirements change arrival is represented by using a Poisson process. Actually, some properties of a Poisson process may not conform to the characteristics of requirements change. For example, the probability of two or more requirements changes in a small interval should be essentially 0 if requirements change follows a Poisson process, which does not reflect practical characteristics of requirements change arrival. Though there are several limitations for its application, we use a Poisson process because of its utility for simulation studies.

Suppose that \overline{RC} represents the mean of requirements changes per month. Based on a Poisson process, the interval between any pair of successive requirements changes T follows the probability distribution

$$\Pr(T > t) = \exp(-\overline{RC} \times t). \tag{2}$$

By generating a continuous value ranging from 0 to 1 at random and assigning it to the inverse function of the formula (2) as probability, each interval between two successive requirements change arrivals can be determined. By repeating this calculation until accumulative intervals exceed the duration of a product project, the number of requirements changes is bounded.

By following the same assumption as adaptive rework effort model, the effort of each requirements change can be determined as follows. The nominal effort of requirements change $\Delta\text{Eff}_{req\cdot i}(r_j)$ (in months) caused by the change request r_j in the project i is assumed to be represented by the formula

$$\Delta\text{Eff}_{req\cdot i}(r_j) = \text{EffDist}_{wcreq\cdot i}^{-1}(p) \times \text{WEM}_i(t_{r_j}), \tag{3}$$

where $\text{EffDist}_{wcreq\cdot i}^{-1}(p)$ is the inverse function of the effort distribution probability function concerning the worst case requirements change for the project i . $\text{WEM}_i(t_{r_j})$ is in the same notation of formula (1). EffDist_{wcreq} can be considered to have almost the same distribution patterns like Fig. 2(b).

Nurmuliani [5] reported that the average rate of overall requirements change during development lifecycle increased sharply when analysis and documents reviews were being completed, and decreased as the project was getting closer to the end of its lifecycle. We can reflect Nurmuliani’s observation by changing RC during product projects.

2.4 Defect Correction Effort Model

To determine the total defect correction effort, we consider as parameters (1) the number of defects injected during a product project ($ND_{product}$), (2) the defect correction completion time of these defects, and (3) the effort of each piece of defect correction.

$ND_{product}$ can be estimated based on estimated product size, process quality, and past experience like NRD_{core} . The defect correction completion time can be determined by using an SRGM in the same way described in Sect. 2.2. By introducing $EffDist_{wcdc}$ representing worst case defect correction, the nominal effort of defect correction $\Delta Eff_{dc.i}(r_j)$ (in months) caused by the defect d_j in the product project i is assumed to be represented by the following formula:

$$\Delta Eff_{dc.i}(d_j) = EffDist_{wcdc.i}^{-1}(p) \times WEM_i(t_{d_j}), \quad (4)$$

where $EffDist_{wcdc.i}^{-1}(p)$ is the inverse function of $EffDist_{wcdc}$ for the project i . $WEM_i(t_{r_j})$ is in the same notation of formula (1). $EffDist_{wcdc}$ can be considered to have almost the same distribution patterns like Fig. 2 (a).

2.5 Model Assumptions

The simulation model relies on the following assumptions.

1. The total effort for product projects is much larger than that for core projects, as stated in Sect. 2.1.
2. Adaptive rework, requirements change, and defect correction occur at the time when the causal defect is detected or the causal requirements change request arrives. Actually, this assumption is not true in practice, as defect correction delay is typically observed [24].
3. The effort of adaptive rework decreases from $EffDist_{wcar}$ depending on DEP and WEM. The effort of requirements change and defect correction decreases from $EffDist_{wcreq}$ and $EffDist_{wcdc}$ respectively, depending on WEM. These assumptions are partly supported by [17,18,19,20,21].
4. Adaptive rework for completed projects is not performed even though later defect corrections in dependent core assets may be performed.
5. Adaptive rework, requirements change, and defect correction never inject other defects. This assumption is supported from a different viewpoint by the observation in [24] showing that the impact of imperfect defect correction is in practice negligible.
6. There is no relationship between any two pieces of adaptive rework, requirements change, or defect correction.

3 Simulation Results

3.1 Project Data and Parameters

We have studied a fictional SPL development project for simulation. There are various kinds of team arrangements for SPL development. We suppose that a

core project along with its maintenance phase and product projects are conducted concurrently by different teams. We also suppose that product projects are scheduled close together to shorten total cycle time as much as possible.

Table 1 shows an overall plan for the project. The first two rows represent when each project is planned to start and finish. The next row “team” represents the team to be assigned for each project. The next four rows represent dependency between core assets and products as well as among core assets. In this project, 4 core assets are scheduled to be developed by the team C independent of the product teams PA and PB, while 10 products by the two product teams concurrently. For example, the product project p3 is planned to start at 4th months, to finish at 7th months, and to be conducted by the team PB. The scheduled total cycle time is 15 months.

Table 1. A SPL development project for simulation

	core projects				product projects									
	c1	c2	c3	c4	p1	p2	p3	p4	p5	p6	p7	p8	p9	p10
start (months)	0	2	5	7	2	2	4	5	7	8	9	11	12	13
finish (months)	2	4	7	9	5	4	7	8	9	12	11	13	15	15
team	C	C	C	C	PA	PB	PB	PA	PB	PA	PB	PB	PA	PB
c1	–	x	x	x	x	x	x	x	x	x	x	x	x	x
c2	–	–					x	x		x	x		x	
c3	–	–	–						x	x		x		x
c4	–	–	–	–							x	x	x	x

Note that product sizes of core assets and products as well as the number of assigned resources for each project are not considered here, because these factors do not directly affect simulation results in the proposed model. Product size does affect NRD_{core} and $ND_{product}$ as described in Sect. 2.2 and 2.5. DEP might be partly dependent on product size.

Several patterns for NRD_{core} , $ND_{product}$, and \overline{RC} are studied to explore the research questions. For the other parameters, a fixed value is applied. Table 2 shows the values for these parameters.

Regarding to \overline{RC} , the values in Table 2 are determined based on [5] reporting that mean requirements change per week is approximately between two and three. To reflect Nurmuliani’s observation [5] stated in Sect. 2.3, we divide a product project into halves and assign different values plus and minus two from those values for the both part of the project. For example, to represent \overline{RC} for the pattern no.2, we use 8 for the first half of a project and 4 for the latter half.

To generate the distributions of $EffDist_{wcar}$, $EffDist_{wcreq}$, and $EffDist_{wcdc}$, we use the right-hand side half of a normal distribution with the parameters shown in Table 2. The parameters μ and σ for each distribution are determined subjectively to follow almost the same distribution patterns in Fig. 2. The reason why $EffDist_{wcar}$ has a larger standard deviation than the others is that some residual defects bring multiple changes in a product, as described in Sect. 2.2.

Table 2. Simulation parameters

parameters	pattern no.				remarks
	1	2	3	4	
NRD_{core}	4	6	8	10	for each core asset
$ND_{product}$	2	4	6	8	for each product project (per month)
\overline{RC}	3	6	9	12	for each product project (per month)
DEP	0.5				a linear model with a factor of 20 the right-hand side of normal distribution the right-hand side of normal distribution the right-hand side of normal distribution
WEM	0.05 to 1.0				
$EffDist_{wcar}$	$\mu = 0, \sigma = 4$				
$EffDist_{wcreq}$	$\mu = 0, \sigma = 2$				
$EffDist_{wcdc}$	$\mu = 0, \sigma = 2$				

For WEM, a simple linear model ranging from 0.05 to 1.0 is used, which can represent a factor of 20 differences of defect correction cost during design and test [20,21].

We need to consider another parameter for SRGM to determine defect correction completion time for both adaptive rework and defect correction. Though numerous SRGMs have been proposed in the literature [23], we apply the following simple logarithmic function

$$y = 1 + \log_a x, \tag{5}$$

where y represents cumulative rate of defect correction and x represents normalized duration ($0 < x \leq 1$). In this simulation $a = 20$ is used. It means that, for example, 60% of defects are corrected before 30% of the duration, and 90% of defects before 75% of the duration.

3.2 Result 1: In-Depth View of Simulation Results

Fig. 3 shows an in-depth simulation result under the parameters $NRD_{core} = 4$, $ND_{product} = 2$ and $\overline{RC} = 3$, which represents when corrective maintenance in core assets and requirements changes occur. One can see that more requirements changes are appeared in the first half of each product project, as we use two different values of \overline{RC} for first and latter half of the project.

Fig 4 shows the effort histograms of generated unplanned work under the same simulation setting in Fig. 3. The shapes of these histograms are all skewed to the right, as $EffDist_{wcar}$, $EffDist_{wcreq}$, and $EffDist_{wcdc}$ also have right-skewed distributions. The range of Fig 4 (a) is almost the same as those of Fig 4 (b) and (c), though $EffDist_{wcar}$ has larger variability than the others. This is because the effort of each piece of adaptive rework is decreased by multiplying DEP whose value is 0.5. The frequency differences among Fig 4 (a, b, c) depend on the parameters such as NRD_{core} , $ND_{product}$ and \overline{RC} as well as the number of product projects.

With these figures, one can understand how cost overruns are calculated by the simulation model.

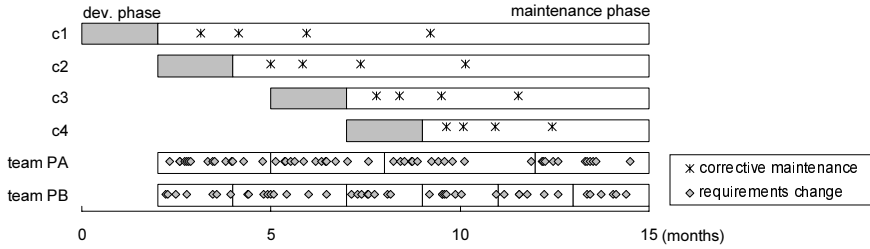


Fig. 3. In-depth view of a simulation result

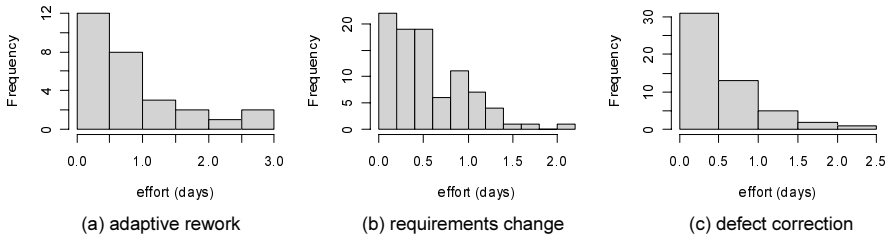


Fig. 4. Effort histograms of generated unplanned work under the same setting in Fig. 3

3.3 Result 2: Variability of Project Cost Overrun

To explore how much project cost overruns and its variability are expected, we conducted 100-run simulations for several combinations of the parameters. The boxplots in Fig. 5 represent the selected results. The mean and the standard deviation of each combination are shown in the tables below the boxplots. All distributions appeared in Fig. 5 are symmetrical shapes. Note that each y-axis has a different range of nominal cost overruns among boxplots. In Fig. 5 (a, b, c), we have focused on one parameters. That is, we set the value zero for the other two parameters to show the isolated effect on nominal cost overrun. Fig. 5 (d) represents the simulation results changing all of these parameters at the same time.

As compared with Fig. 5 (a, b, c), NRD_{core} has the smallest impact on nominal cost overruns as well as its variability than the other two parameters, for the studied simulation settings. Meanwhile, \overline{RC} has the largest impact on nominal cost overrun than the other two parameters. Of course, these results completely rely on what values we have studied for each parameter, which is shown in Table 2. So we can not generalize which parameter has the largest impact on nominal cost overruns. These results only demonstrate the capability of the simulation model to estimate nominal cost overruns as well as its variability based on these parameters. If one gives values of these parameters for a specific project, the possible overruns and its variability can be estimated by the proposed model.

Fig. 5 (d) implies that project cost overruns can be held down if requirements volatility and quality are well managed. The mean nominal project cost overruns

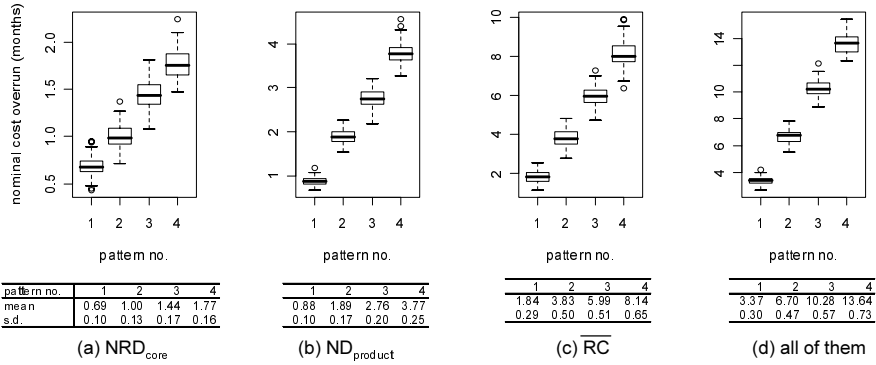


Fig. 5. Simulation results for estimating nominal project cost overruns (Note: y-scales are different)

for the best combinations of these parameters is 3.37 months, while that for the worst case is 13.64 months. As the planned total cycle time is 15 months, the difference between the best and the worst cases are quite significant. The balanced relative estimation error [25] for the mean worst case is 90.1%, which is not regretfully an unrealistic error among typical software projects [26]. It implies that poor management on requirements and quality will result in almost doubled estimation error for the worst case of the studied simulation settings.

On the other hand, the differences of variability are relatively small compare to those of the means. For example, the standard deviation of the best case is 0.30 months, while that of the worst case is 0.73 months. It is because that most pieces of unplanned work are distributed among smaller values, even though a lot of unplanned work is generated for the worst case. This result might be different if the assumption no.6 in Sect. 2.5 is changed.

4 Model Evaluation

Because of the nature of simulation study, it is impossible to validate all aspects of the proposed simulation model comprehensively. However, the utility of the model can be evaluated by using empirical data, even though it will not demonstrate comprehensive validation. As we are in the process of trying to collect empirical data, we follow some of typical validation aspects for simulation studies [27], which has been applied in our previous work [7].

Conceptual model validity and data validity: The proposed model is considered to be reasonably valid under the assumptions described in Sect. 2.5, because the proposed formulas are partly supported by several empirical observations as stated in Sect. 2. Data validity as input to the model is also supported by these empirical observations in terms of determining effort distributions. However, there are some limitations of the model, which is discussed in Sect. 5.1.

Operational validity: In general, operational validity is difficult to assess when no observable problem entity is available. In such a case, comparison to other models is one of meaningful approaches to validate a simulation model [27]. However, this approach is not applicable in this case, because both COCOMO II [12] and COPLIMO [13], a COCOMO II based cost estimation model for SPL development, do not produce variability of estimated effort. These models have a lot of parameters such as effort multipliers, but these parameters are deterministic but not stochastic. In addition, the proposed model is not capable of being compared to these models, as the model uses nominal cost scale. Sensitivity analysis is another useful approach to demonstrate operational validity of the model, which we have already discussed in Sect. 3.3. It can be considered that the model has reasonable validity, but some limitations still exist.

Another possible approach is to evaluate the generated adaptive rework by the simulation program rather than total cycle time. By comparing the distributions of the generated adaptive rework in Fig. 4 to the change effort distribution from the SEL data in Fig. 2, both distributions can be subjectively judged to be similar. At least, we can conclude that the simulation model is capable of producing reasonable adaptive rework distributions.

5 Discussion and Related Work

5.1 Limitation of the Model

One of the most important limitations of this study is that the proposed model uses nominal cost scale but absolute cost scale. That is, the model estimates cost overruns in month scale but not in person-month scale. This limitation comes from the lack of considerations for resources and product size. In addition, the model does not consider unplanned work during core asset development phase. Matured SPL development tends to have more effort on core projects rather than product projects [16], so this limitation should be overcome to increase applicability of the model. Now we are in the process of enhancing the simulation model to overcome these limitations and some assumptions as well.

Another arguable assumption in this model is the usage of DEP. We assume that the effort of a piece of unplanned work decreases linearly from worst-case effort by multiplying DEP. We do not have any supporting evidence for this assumption at this moment. We might say that this assumption is not very unrealistic by carefully looking at the generated unplanned work distributions. In addition, we do not have any specific method to calibrate DEP. It might reflect attributes such as coupling between core components and product components, number of dependent product components reusing a core component, and inheritance depth between core and product components. Those attributes and measured values have to be translated into DEP and calibrated by checking generated adaptive rework distributions like Fig. 4 (a).

5.2 Related Work

Software process modeling approaches can be categorized into the following three types [28]: analytical models such as COCOMO II [12], continuous simulations such as system dynamics models [29,30], and discrete-event simulations [31]. Some studies use a combination of those approaches [14,28,32]. The proposed model is a discrete-event simulation.

Discrete-event simulation models are suitable for detailed analyses of the process and project performance, while continuous simulation models are useful to represent effects of feedback and changes in a continuous fashion [33]. As the proposed model represents sequential events concerning defects and requirements change requests, the use of a discrete-event simulation model is reasonable.

Several studies have appeared in the literature on estimating the benefits of SPL development [13,34,35]. These studies use more macro-level analytical models than our model. The primary purpose of the studies [34,35] is for estimating the return on investment of SPL development compared with non-SPL development. COPLIMO [13] is a deterministic cost estimation model for SPL and does not represent uncertainty, as well as COCOMO II [12]. COCOMO-U [15] introduces uncertainty into COCOMO II, but does not mention how the model can be applied to SPL development.

Chen et al. proposed a discrete-event SPL process simulator using COPLIMO as their cost model [14]. Schmid et al. studied SPL planning strategies through deterministic simulations [2]. These two studies have similar research questions to ours. However, these studies do not explicitly use factors such as NRD_{core} , $ND_{product}$, DEP, and requirements volatility. They are also not capable of calculating the level of risk of estimated effort under uncertainty, as they are based on deterministic simulation models.

6 Conclusions

In this paper, we proposed a stochastic simulation model for estimating nominal project cost overruns and its variability in time-boxed SPL development, based on our previous model. Simulation results demonstrate that the model is capable of estimating reasonable nominal cost overruns and its variability. We have shown that poor management on requirements and quality will result in almost doubled estimation error, for the studied settings. The model has been partially validated by comparing generated unplanned work to actual change effort, which consequently demonstrates the validity of estimated project cost overruns.

Our immediate future work is to enhance the model to overcome limitations and assumptions, consider calibration of the parameters, and to validate the model by using empirical project data. We are now working on these issues.

Acknowledgments. This study was partially supported by the Ministry of Education, Science, Sports and Culture, Grant-in-Aid for Young Scientists (B), 16700042, 2005. NICTA is funded through the Australian Government's Backing

Australia's Ability initiative, in part through the Australian Research Council. The third author was working with NICTA when this paper was produced.

References

1. Clements, P., Northrop, L.M.: *Software Product Lines: Practices and Patterns*. Addison-Wesley, MA (2001)
2. Schmid, K., Biffl, S.: Systematic management of software product lines. *Softw. Process Improve. Pract.* 10, 61–76 (2005)
3. Genuchten, M.v.: Why is software late? an empirical study of reasons for delay in software development. *IEEE Trans. Softw. Eng.* 17 (1991)
4. Subramanian, G.H., Breslawski, S.: An empirical analysis of software effort estimate alterations. *J. Systems and Software* 31, 135–141 (1995)
5. Nurmuliani, N., Zowghi, D., Fowell, S.: Analysis of requirements volatility during software development life cycle. In: *Proc. 2004 Australian Softw. Eng. Conf. (ASWEC'04)* (2004)
6. Dijkstra, E.: Notes on structured programming. In: Dahl, O.J., Dijkstra, E., Hoare, C.A.R. (eds.) *Structured Programming*, Academic Press, London (1972)
7. Nonaka, M., Zhu, L., Babar, M.A., Staples, M.: Project delay variability simulation in software product line development. In: *Proc. Intl. Conf. Software Process (ISCP) (to appear)*
8. IEEE: *Ieee std. 1219-1998, ieee standard for software maintenance* (1998)
9. Jørgensen, M., Moløkken, K.: Reasons for software effort estimation error: Impact of respondent role, information collection approach, and data analysis method. *IEEE Trans. Softw. Eng.* 30, 993–1007 (2004)
10. Jørgensen, M.: Realism in assessment of effort estimation uncertainty: It matters how you ask. *IEEE Trans. Softw. Eng.* 2004, 209–217 (2004)
11. Procaccino, J.D., Verner, J.M.: Software project managers and project success: An exploratory study. *J. Systems and Software* 79, 1541–1551 (2006)
12. Boehm, B.W., Abts, C., Brown, A.W., Chulani, S., Clark, B.K., Horowitz, E., Madachy, R., Reifer, D., Steece, B.: *Software Cost Estimation with COCOMO II*. Prentice-Hall, Englewood Cliffs (2000)
13. Boehm, B.W., Brown, A.W., Madachy, R., Yang, Y.: A software product line life cycle cost estimation model. In: *Proc. Intl. Symp. Empirical Softw. Eng. (ISESE'04)*, pp. 156–164 (2004)
14. Chen, Y., Gannod, G.C., Collofello, J.S.: A software product line process simulator. *Softw. Process Improve. Pract.* 11, 385–409 (2006)
15. Yang, D., Wan, Y., Tang, Z., Wu, S., He, M., Li, M.: Cocomo-u: An extension of cocomo ii for cost estimation with uncertainty. In: Wang, Q., Pfahl, D., Raffo, D.M., Wernick, P. (eds.) *Software Process Change. LNCS*, vol. 3966, pp. 132–141. Springer, Heidelberg (2006)
16. Bosch, J.: Maturity and evolution in software product lines: Approaches, artefacts and organization. In: *Proc. 2nd Intl. Conf. Softw. Product Lines 2002*, pp. 257–271 (2002)
17. Epping, A., Lott, C.M.: Does software design complexity affect maintenance effort? In: *Proc. 19th Softw. Eng. Workshop*. 1994, pp. 297–313 (1994)
18. Bocco, M.G., Moody, D.L., Piattini, M.: Assessing the capability of internal metrics as early indicators of maintenance effort through experimentation. *J. Software Maintenance and Evolution* 17, 225–246 (2005)

19. Ramanujan, S., Scamell, R.W., Shah, J.R.: An experimental investigation of the impact of individual, program, and organizational characteristics on software maintenance effort. *J. Systems and Software* 54, 137–157 (2000)
20. Kan, S.H., Dull, S.D., Amundson, D.N., Lindner, R.J., Hedger, R.J.: As/400 software quality management. *IBM Systems Journal* 33, 62–88 (1994)
21. Remus, H.: Integrated software validation in the view of inspections / reviews. In: *Proc. Symposium on Softw. Validation*, pp. 57–64. Elsevier, North-Holland (1983)
22. SEL: Sel, (software engineering laboratory) data (1997), <http://www.cebase.org>
23. Musa, J.D.: *Software Reliability Engineering*. Osborne/McGraw-Hill (1998)
24. Defamie, M., Jacobs, P., Thollembeck, J.: Software reliability: assumptions, realities and data. In: *Proc. 1999 Intl. Conf. Softw. Maintenance (ICSM'99)* pp. 337–345 (1999)
25. Miyazaki, Y., Takanou, A., Nozaki, H., Nakagawa, N., Okada, K.: Method to estimate parameter values in software prediction models. *Inf. Softw. Technol.* 33, 239–243 (1991)
26. Moløkken, K., Jørgensen, M.: A comparison of software project overruns—flexible versus sequential development models. *IEEE Trans. Softw. Eng.* 31, 754–766 (2005)
27. Sargent, R.G.: Validation and verification of simulation models. In: *Proc. 31st Conf. Winter Simulation* pp. 39–48 (1999)
28. Donzelli, P.: A decision support system for software project management. *IEEE Software* 23, 67–75 (2006)
29. Abdel-Hamid, T., Madnick, S.: *Software Project Dynamics- An Integrated Approach*. Prentice-Hall, Englewood Cliffs, NJ (1991)
30. Calavaro, G.F., Basili, V.R., Iazeolla, G.: Simulation modeling of software development process. In: *Proc. 7th European Simulation Symposium. Soc. for Computer Simulation* (1995)
31. Antoniol, G., Cimitile, A., Lucca, G.A., Penta, M.: Assessing staffing needs for a software maintenance project through queuing simulation. *IEEE Trans. Softw. Eng.* 30, 43–58 (2004)
32. Martin, R., Raffo, D.: Application of a hybrid process simulation model to a software development project. *J. Systems and Software* 59, 237–246 (2001)
33. Kellner, M.I., Madachy, R.J., Raffo, D.M.: Software process simulation modeling: Why? what? how? *J. Systems and Software* 46, 113–122 (1999)
34. Cohen, S.: Predicting when product line investment pays. Technical Report Technical Report CMU/SEI-2003-TN-017, Software Engineering Institute, Carnegie Mellon University (2003)
35. Böckle, G., Clements, P., McGregor, J.D., Muthig, D., Schmid, K.: Calculating roi for software product lines. *IEEE Software* 21, 32–38 (2004)

E-Service Architecture Selection Based on Multi-criteria Optimization

Edzus Zeiris and Maris Ziemā

Riga Technical University, Faculty of Computer Science and Information Technology
Meza 1/3, LV-1048, Riga, Latvia
{edzus,maris}@zzdats.lv

Abstract. The selection of the most acceptable architecture of e-services system is very important issue. One and the same e-service can be designed using different alternative architectures. Each system has different execution indices that are very important for the e-services clients and providers. This article shows solutions for compromise or the most acceptable selection of the architecture of e-services system using more than one criterion at the same time. The solution is based on the theory of graphs and usage of multi-criteria methods and their basics is following: E-service algorithm is described with an algorithm graph. Using segmentation of algorithm graph web service graphs are obtained that are assessed with characteristic numerical values of system architecture. Several characteristics of system architecture are: Reusability, Costs of Production and Time of Execution. The task of multi-criteria optimization of web service graphs is defined when as result the compromise or the Pareto set of web service graphs is evaluated. The most acceptable solution of system architecture is selected from Pareto set by using additional information. The usage of offered method is demonstrated with help of practical example.

1 Architecture of E-Services System

Currently achievements of information technologies create not only possibilities but also a necessity for simple and efficient means how to ensure information receipt, processing, saving and exchange. Internet has become one of main means for furnishing and receiving information and services.

In such context there occurs necessity to talk about electronic services or e-services, their development, structure and architecture. E-service traditionally is realized as a set of actions of information systems. It contains functional possibilities of several systems as a result giving material and nonmaterial wealth for society (physical and legal persons). Electronic services contain four levels of electronization: 1st – information about the service is available on the internet; 2nd – it is possible to download forms that are necessary for receiving the service; 3rd – it is possible to hand in data electronically for receiving the service; and 4th – complete the electronization of service. Providers of the service and clients must ensure the electronization of service. Further in this article we will understand that e-service is a

service that is electronized according to electronization levels for usage on the internet.

For electronic services of levels 3 and 4, it is useful to look at architecture of E-services' systems.

Let's view common architecture of e-services system [1, 2].

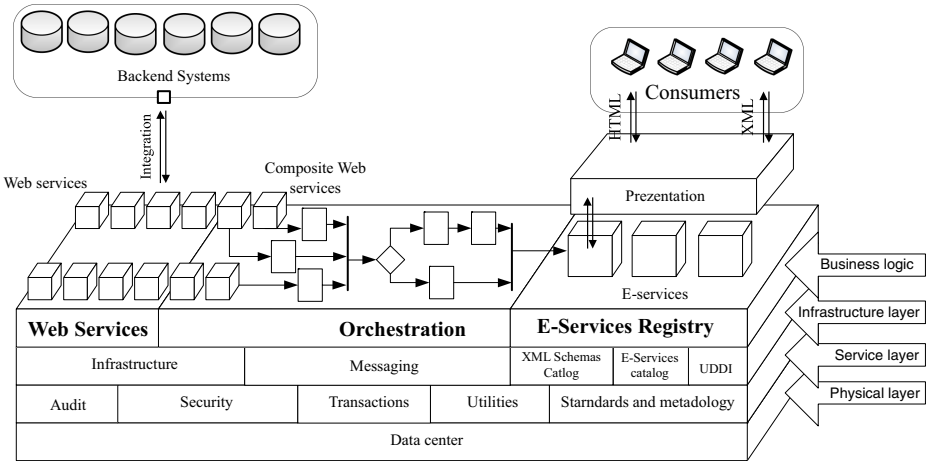


Fig. 1. E-services system architecture

In the logical scheme of architecture (Fig. 1) is shown how the systems that are involved in the service are linked into e-service. There should be elaborated a group of XML schemes for every object of data which should be involved into realization of e-service. Data acquisition from the relevant functional backend system is realized by means of web services. Web service is program component which interfaces can be described with Web Service Description Language (WSDL) and which can be accessed by sending standardized XML messages via standard net protocols. For example SOAP over HTTP. Web services are used to compose e-services. By making calls of web services, also metadata is sent that describes the request. Information that is necessary for filling in auditing registrations is transferred via metadata.

As orchestration (integration) environment of process should be used the BPEL (Business Process Execution Language) processor, as a result of this are e-services that are delivered to applications ensuring e-services for end users, e.g. portals, one-stop agencies, etc. E-services' input forms, milestones, information about payments and results of fulfillment are delivered by HTML or by XML pages which may be used in the portal in order to implement the service by using XSLT transformation.

Holders of web services and e-services, i.e. specialists of institutions and system operators who are responsible for maintenance and development of web services and e-services should have a possibility to communicate mutually on various issues related to execution and development of web services and e-services, as well as execute asynchronous e-services. System of messages is provided for this purpose.

System of messages ensures work with messages and work assignments. Application of messages is integrated with E-services' register from which it receives data about XML schemes, web services and e-services. On the other hand, application of messages is a client of orchestrations, because the messages about the execution of web service and e-service are being received from them. Data about all XML schemes, web services and E-services are registered in one common register "E-services' register". There are maintained all versions of schemes, web services and e-services in order to provide this information available for anyone being involved in elaboration and development of e-services.

In architecture of e-service systems there should be also paid attention to the security system. The main goal of establishing e-services' security system is to ensure successful performance of the e-service in all steps. Main tasks of the security system are to provide authentication and authorization of clients, integrity and accessibility of the service.

It is necessary to regulate all processes and outputs by relevant standards and guidelines.

E-service systems' architecture is established as a hub with external interfaces being able to integrate systems in e-services. Functionality of such architecture allows various types of topologies. Depending from requirements and physical placement, it is possible to select where and how to perform one or the other action. Possible topologies can be as one central hub and as well as divided peer-to-peer architecture of mutually linked hubs or multilevel hierarchies that are much more complicated.

In such architectural type, each e-service basically is determined by its algorithm that is formed as separate orchestration. Such orchestrations link together one or several web services of functional systems into e-service. Combinable web services can be simple and composed. Orchestrations of e-services are the variable part of such architectural type and it is necessary to elaborate it for every service individually. Further in the article we will view one part from architecture of the e-services system that we can see in Fig. 1 on the level of business logics [3, 4, 5, 6].

2 Architecture Selection Problem for E-Service

E-service orchestration is essential component of the service. It defines the e-service algorithm and therefore the execution (result). Web services and their orchestrations that are used in these services characterize the entire e-service. Due to this reason, further by the architecture of e-service we will understand web services involved in the service and their orchestration.

By forming e-services, as well as other systems, it is known that one and the same result may be reached by executing various algorithms. When forming e-services such problem is very topical as orchestrations of services often are made by elaborators with different qualifications and therefore the quality of the established e-services often is different in various measurements. To increase the quality of the elaborated services, it is necessary to assess one or the other orchestration of the e-service and select the best from several ones [7, 8].

In similar situations other authors offer to select the architecture by one or more evaluating criteria that are based on experience. For example Jan Bosch related work

[9, 10]. Solution offered by the author differs in that there is architecture assessing formula and optimal architecture assessing formula. Then selected architecture results are compared with optimal architecture evaluated values. Not always optimum is reached in this, because not all possible solutions are considered. This article contains solution that is different because all possible solutions are examined and all possible solutions of optimum set are evaluated.

A service in its essence consists of several activities that can be fulfilled synchronously or asynchronously. Each of activities has to be complete in order to comply with principles of Service Oriented Architecture (SOA) – execution of one activity is not related with status of other activities. By establishing Web services in the SOA environment, one should be guided by two basic principles [11]. Firstly, a high cohesion – it means that in one Web service should be combined uniform activities, and secondly, a minimal coupling – Web service should work autonomously, independent of other services in order to increase its reusability.

For the sake of convenience in this article by one activity of the service we will understand the web service which realizes it. There must be solved one problem, how to divide the service algorithm throughout web services, while designing e-service. Whether to design one web service, whether as many as possible where each web service contains minimal functionality in order it would conform to SOA?

Let us imagine simple example of e-service offered by some institution that allows service client to request information from institution. The requested information is sent to client in asynchrony way. The requested information is prepared automatically from the knowledge base. If no information is found, then request is formatted and sent to office-worker of this institution for manual preparation. All clients of institution and cooperation with them are registered in CRM system therefore the process of this e-service is following:

1. Find client in CRM system. If it is impossible to find this client, then go to the 2nd step; otherwise go to the 3rd step.
2. Register new client in CRM.
3. Register client's request in CRM system.
4. Find requested information in data base. If there is no such information then go to the 5th step; otherwise go to the 6th step.
5. Prepare request for office-worker of institution for manual execution.
6. Prepare requested information answer, register in the documentation system and send it to the client.

Knowledge base system already has Web service interfaces for integration with other systems that have all required functionality. Documentation system has one interface for document registration that can be realized as Web service. Documentation system is developed by other company, therefore it can't be modified (e-service process is shown in Fig. 2).

This kind of selection problem of architecture service is related to CRM system. Registration in CRM system must be done in three steps that increase time of execution. All CRM registration steps can be merged in one step due to minimize time of execution, but such merging increases CRM costs of additional improvements on the basis of frequently repeated long term usage. For example if institution decides to offer e-service where only client registration will be needed in CRM system.

Further in this article it will be described how to make a selection among various solutions of the e-service architecture.

To select the best e-service architecture it is offered to use theory of graphs. Initially algorithm graph of e-service should be defined, which according to its essence is algorithm description of e-service execution in the form of orientated graph, where each vertex contains certain number of activities to-be-executed in the e-service in order it would conform to SOA principles, and edges are informative links among activities. It is possible to segment the graph of algorithm in various ways, by thus altering parameters of algorithm activities. Graph segmentation means that there are being searched combinations of graph vertexes in all possible ways. The segmented graph may be transformed as web service graph where each web service is realizing functionality from the segment of algorithm graph. Such web service graph that has developed as a result of segmentation of algorithm graph may be regarded as characterization of the e-service architecture.

Selection of architecture can be realized as the task of finding set of web service graphs that correspond to Pareto – optimum according to N set criteria [12].

3 E-Service Architecture Description with Graph

Essence of the e-service is determined by its execution algorithm. If the e-service execution process is known then it is possible to describe the service execution algorithm precisely, e.g. in any programming language. It is possible to describe the e-service as a set of web services with links describing architecture of the given service. Let's define the e-service algorithm graph as an orientated graph $G = (S, L)$, where $S = \{s_1, s_2, \dots, s_n\}$ is a final set – vertexes of the graph which according to their essence are to-be-executed activities of the e-service algorithm, and $L \subset S \times S$ are edges of the graph. The edge $l_i = (s_j, s_k)$ in the graph means that in e-service algorithm after execution of the activity s_j there follows execution of the activity s_k . Edges in the e-service algorithm graph indicate the information flow. Example of the e-service algorithm graph is shown in Fig. 2. The vertex marked by '1' is the beginning of the algorithm. Vertexes „4” and „6” are made by various elaborators, therefore they are in various patterns.

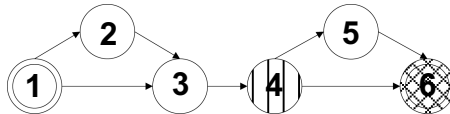


Fig. 2. E-Service graph

There must be taken under consideration some restrictions that are related to SOA when establishing e-service algorithm graph in order it could be used as the basis for establishing architecture of e-services:

Every algorithm graph vertex s_i must be able to realize to-be-executed activity that is included in it. It means that the vertex is acting in an atomic transaction and is not related to realized algorithms in other vertices. This condition is related to high cohesion and minimal coupling.

- Every vertex should contain at least one to-be-executed activity. In practice it is related to algorithm-implementing methods. For further goals let's mark the implementing methods of vertex activities by $M_{s_i} = \{m_{s_i}^0, m_{s_i}^1, m_{s_i}^2, \dots, m_{s_i}^g\}$ and the number of every method's lines $m_{s_i}^j$ by $O(m_{s_i}^j)$
- Activities repeating during the algorithm execution time cannot be established in the graph as various vertices $\forall s_i, s_j, s_i \neq s_j, s_i \in S, s_j \in S, M_{s_i} \cap M_{s_j} = \emptyset$. It is necessary in order to ensure high cohesion and initially exclude processing of unuseful versions.

In order to make the graph segmentation and formally describe it, let's view graph G as depiction Γ arranging for every vertex a subset ($s_i \subset S$) in vertices' set S . This subset according to its essence is reachable vertices from vertex s_i . Γs_i are edges outgoing from the vertex and $\Gamma^{-1} s_i$ are incoming edges into the vertex. For such e-services' algorithm graph exists at least one vertex $s_0 (s_0 \in S)$ which does not have any incoming edge $\Gamma^{-1} s_0 = \emptyset$, that we will name as the beginning of e-service algorithm, and likewise exists at least one vertex $s_r (s_r \in S)$ which does not have any outgoing edges) $\Gamma s_r = \emptyset$, this we will name as the result of e-service algorithm [13].

For example, as in the graph in Fig. 2 $s_0 = \{ "1" \}, s_r = \{ "6" \}, \Gamma \{ "4" \} = \{ "5", "6" \}, \Gamma^{-1} \{ "4" \} = \{ "3" \}$.

4 Graph Measurement

Web services' graph is obtained by segmenting algorithm graph. There are possible several segmentations, let's define set $X = \{G'\}$, containing all possible graphs that are recursively derived from initial web services' graph G . Alterations of the graph are made by merging the vertices. Merge s' of two vertices s_i and s_j is as merge of both vertices' outgoing and ingoing edges. $\Gamma s' = \Gamma s_i \cup \Gamma s_j$ and $\Gamma^{-1} s' = \Gamma^{-1} s_i \cup \Gamma^{-1} s_j$. A set of methods in the newly established vertex is formed

as following $M_{s'} = M_{s_i} \cup M_{s_j}$ Measurements of the number of lines are summing up, by merging vertexes.

Rather often during the execution of e-service there are involved various functional systems that have been created by various elaborators, therefore it is not possible to realize these activities of systems in one web service. Initially for this purpose it is necessary for every vertex of algorithm graph to determine with which one it is not able to be merged. In other words, for every algorithm graph vertex let's arrange indication $I(s_i)$. In such way merge of vertexes s' is possible only then if $I(s_i) = I(s_j)$.

Let's define characteristic values of Web services' graph in terminology of architecture. This article contains only three of all possible characteristics to show the method in selection of architecture. The selection of architecture can be added with $0 < N < \infty$ characteristics.

Graph of web services is characterized by:

- Reusability *ARSD*;
- Costs of Production *C*;
- Time of Execution *T*.

Further the metrics of web service graphs of e-services and their formulas are assigned in a way that all of them can be minimized.

Reusability

Development of web services for establishing of e-services is a very labor intensive process. Costs are sufficiently high and not always in the beginning there are so many e-service users in order to be worthwhile for short-term. Web services should be established in that way so they would be useful as well as for short-term as for long-term. For such purposes it is necessary to establish services in a way so they can be maximally reused. It is necessary to establish services maximally independent from other resources in order to be used for various goals where the functionality of previously established services will be needed. While talking about web services, it is assumed that they conform to SOA, so they are maximally independent from others, do not contain status information and can work autonomously. If there are fulfilled such conditions, then it is possible to talk about maximal cohesion of services and minimal coupling.

In case when web services are used for establishing e-services, by inserting them into the web services' graph of e-service, it is possible to assess dependency of the established web services' graph. In this article with Reusability we will understand Web service dependency of others Web services. In other cases there must be included functionality of web service's atomism in measurements of reusability.

To calculate the average web service's dependency of other services *ARSD* (Average Required Service Dependency) [14] upon other services in web services' graph, let's use the formula as following:

$$ARSD = \frac{1}{n} * \sum_{i=1}^n R_i, \quad (1)$$

where $n = |S|$ is a total number of web services in web services' graph, and R_i is the number of services linked with web service i , or in other words, the number of incoming and outgoing edges in web services' graph vertex $R_i = |\Gamma s_i| + |\Gamma^{-1} s_i|$.

Costs of Production

To decrease costs of effort for the e-service, it is important to put in minimal work for elaboration of each service. As a basic we will take LOC (Lines Of Code) assessment method to calculate the costs of effort [15]. To calculate effort costs C_{s_i} of each web services' graph Vertex, let's use the algorithm as following:

$$C_{s_i} = \left(\sum_{m_{s_i} \in M_{s_i}} O(m_{s_i}) + W_C(s_i) \right)^p \tag{2}$$

$W_C(s_i)$ is the number describing web service implementation code lines in web services' graph vertex s_i , what is constant for every realization environment of web services. We will use degree p according to LOC method that can be influenced from chosen programming language. Degree p characterizes the implementation complexity of realization in the programming language (p according to its essence is a fractional number). E-service elaboration costs C are calculated as following:

$$C = \sum_{s_i \in S} C_{s_i} \tag{3}$$

Time of Execution

For calculating execution time of e-service, let's adjust average execution time in milliseconds t_{s_i} for each web services' graph vertex with average amount of input data [16]. Additional time $t_{ws} = \log(t_{s_i})$ is needed to execute every of vertex in e-services' realization environment. E-service total time of execution activity is calculated as following:

$$T = \sum_{s_i \in S} (t_{s_i} + t_{ws}) + t_{epak} + t_{data} \tag{4}$$

In order to execute e-service in any of execution environment we have to add additional time $t_{epak} = \log(n)$ to the execution time, where $n = |S|$ (the number of web services in the graph) and the time $t_{data} = \sum_{l_i \in L} t_{l_i}$ that is needed to transfer data from one e-service web service graph vertex to the next one. The average time in milliseconds t_{l_i} should be adjusted for each of web services' graph edges $l_i \in L$ in order to transfer data via this edge [17, 18].

5 Multi-criteria Graph Optimization

Essence of the task is to find all possible web services' graphs from the set X , which are the most appropriate (either none the worse) according to all pointed out criteria (reusability, costs of production, time of execution). It is essential to view simultaneously all criteria as it is not possible to determine which one is the most important and they are not comparable with each other. Let's solve this problem as multi-criteria task which can be reduced in order to find Pareto compromise's set P :

$$Q(X) \rightarrow \min_{X \in \Omega} \rightarrow P, \quad (5)$$

where $Q(X) = \{q_1(X), q_2(X) \dots q_N(X)\}$ - criteria that must be minimized; Ω - X definition area.

Let's call solution $X_{i^*} \in \Omega$ as Pareto optimal $X_{i^*} \in P$ only in case if there does not exist $X_j \in \Omega$ such that

$$q_i(X_j) \leq q_i(X_{i^*}) \quad (6)$$

for all $i = \{1, 2, \dots, N\}$, where at least one is a strict nonequivalence. In other words, there will not appear more appropriate value in the set Ω for any value of $X_{i^*} \in P$.

The e-service algorithm graph is given. Initially it is assumed as web services' graph G . The task is to find the set of graphs $P = \{G' * \}$ that conforms to Ω restrictions and minimizes criteria Q (in our case q_1 is reusability that is calculated as in formula (1), q_2 are costs of production that are calculated as in formula (3) and q_3 is time of execution that is calculated as in formula (4)).

The graphs of Web services that are found in the set P are the possible solutions of architecture [19].

6 Multi-criteria Optimization Solution

Let's demonstrate the offered model by a simple example. Let's make selection of web services' graphs as a multi-criteria optimization based on the e-service algorithm graph shown in Fig. 2.

Let's define the following indications of algorithm graph vertexes I . $I(1)=I(2)=I(3)=I(5)=1, I(4)=2, I(6)=3$.

Firstly all possible architecture solutions must be found in order to get optimal architecture solutions. Let's calculate all possible combinations of vertex segments (it's very important to find the combinations, because from the architectural point of view there is no difference in the sequence of vertex segmentation). Segmentation of vertexes can be made by using different algorithms. One of the simplest ways how to find graphs is to use recursive algorithm that merges each two vertexes of graph if the

indices I for the vertexes are equal until the moment when graph has only one vertex. We use the same algorithm for the graphs that we have found. Set $X = \{G'\}$, in our case, contains 15 derived graphs shown in Table 1, from which we are able to find graphs that belong to Pareto set $P = \{G'^*\}$.

When all possible segments of graph vertex are found, the assessment formulas (1), (3) and (4) are used to get numerical values for optimization (see Table 1).

Table 1. All graph segmentation solutions

Nr.	Graph	Costs of Production	Time of Execution	Reusability	Pareto
1.	G_1	52.4976	33.0130	2.333	
2.	G_2	50.5569	31.0913	2.000	
3.	G_3	48.5386	24.0181	2.000	*
4.	G_4	46.4327	26.1841	2.667	*
5.	G_5	48.5386	33.2854	2.500	
6.	G_6	48.5386	33.2874	2.500	
7.	G_7	50.5569	32.9885	2.400	
8.	G_8	48.5386	35.1828	3.000	
9.	G_9	48.5386	35.1831	3.000	
10.	G_{10}	50.5569	35.2251	2.800	
11.	G_{11}	48.5386	26.6704	2.500	
12.	G_{12}	50.5569	25.9691	2.000	
13.	G_{13}	48.5386	28.1635	2.500	
14.	G_{14}	50.5569	35.2253	2.800	
15.	G_{15}	50.5569	35.2269	2.800	

Evaluated values must be compared in all dimensions using the formula (6) to get the set of Pareto optimum.

For the given example the Pareto set consists of two solutions, depicted in Fig. 3. Solutions are obtained by merging vertexes – in one case “1”, “2” and “3”, and in other “1”, “2”, “3” and “5”.

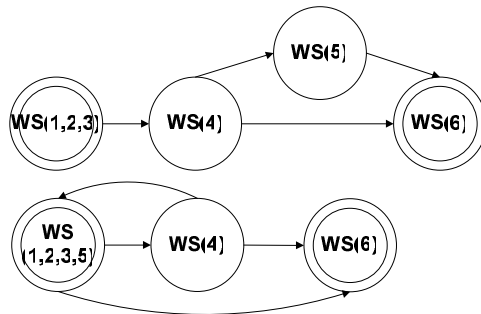


Fig. 3. Optimized graphs of web services

To demonstrate graphically multi-criteria optimization and the set of Pareto optimum, let's choose two criteria:

- Time of Execution;
- Reusability.

Pareto set is shown in Fig. 4. Graphically Pareto solutions are closer to the zero point of coordinate axes.



Fig. 4. Multi-criteria optimization example

Pareto set according to these criteria consists of the solution shown in Fig. 4. If there are not many opportunities to choose from many solutions as in the given example, then the selection of the solution may be made by e-service designers on the basis of their experience, or we can set up priorities that can be realized by decreasing the number of criteria. Also it is possible to use any of multi-criteria optimization methods [19].

7 Conclusion

It is a very complicated task to establish an e-service. One of the problems is the e-service designing – distribution of the service throughout web services and establishing of their orchestration. The selection of the concrete solution affects several essential e-service execution assessments (Costs of Production, Time of Execution, Reusability and others). Execution assessments and number of them are related to the requests, needs of customer and specific characteristics of service. Each added criteria and its evaluation formula must be carefully verified because wrong formula can give incorrect solutions. Initially it is offered to establish an e-service algorithm graph to be transformed as a web service graph in order to select any

concrete solution. The formulas (1), (3), (4) are offered for graphs assessment of web services. It is offered to use multi-criteria optimization in order to make a selection among all possible versions. Multi-criteria optimization is a process of searching for Pareto optimum when as a result Pareto set is obtained. Graphs of web services available in Pareto set are the possible solutions of architecture. Usually the assessments of service's architectural solutions mutually conflicts, therefore the usage of each it must be carefully considered, because they increase the number of possible optimal solutions. Web services' graphs available in the Pareto set may be designed in details, implemented and executed in the execution environment of e-services.

The approach mentioned in the article may be applied for all types of e-services, as well as in similar situations.

References

1. Secretariat of Electronic Government Affairs of Latvia home page, <http://www.eps.gov.lv/>
2. Latvian E-Government home page, <http://www.e-parvalde.lv/>
3. The Open Web Applications Security Projects. A Guide to Building Secure Web Applications and Web Services. 2.0 Black Hat Edition (2005.07.27)
4. Hofmeister, C., Nord, R., Soni, D.: Applied Software Architecture. Addison Wesley, London (2000)
5. Secretariat of Electronic Government Affairs of Latvia. Conception of Integrated Information System of Government. Riga (2005)
6. Microsoft Corporation. Connected Government Framework. Architecture and Design Blueprint (2005)
7. Papazoglou, M.P.: Service – Oriented Computing: Concepts, Characteristics and Directions. In: Keynote for the 4th International Conference on Web Information Systems Engineering, – pp. 3–12 (December 10-12, 2003)
8. Papazoglou, M.P., Yang, J.: Design Methodology for Web Services and Business Processes. In: Proceedings of the Third International Workshop on Technologies for E-Services, – pp. 54–64 (2002)
9. Bengtsson, P.B.J.: Assessing optimal software architecture maintainability. In: Fifth European Conference on Software Maintenance and Reengineering, – pp. 168–175 (2001)
10. Lundberg, L., Bosch, J.: Daniel Häggander and Per-Olof Bengtsson Quality Attributes in Software Architecture Design. In: Proceedings of the IASTED 3rd International Conference on Software Engineering and Applications, – pp. 353–362 (October 1999)
11. Zeiris, E., Ziema, M.: E-Services Development Problems. In: Scientific Proceedings of Riga Technical University. Computer Science. Series 5. Riga vol. 19. –pp. 48–53 (2004)
12. Chatterjee, S., Webber, J.: Developing Enterprise Web Services. An Architect's Guide. Hewlett-Packard Corp. (2004)
13. Dambits, J.: Modern graph theory. Computer Science Centre Riga (2002)
14. Qian, K., Liu, J., Tsui, F.: Decoupling Metrics for Services Composition. In: Proceedings of the 5th IEEE/ACIS International Conference on Computer and Information Science and 1st IEEE/ACIS International Workshop on Component-Based Software Engineering, Software Architecture and Reuse (ICIS-COMSAR'06) (2006)
15. Kan, S.H.: Metrics and Models in Software Quality Engineering, 2nd edn. Addison Wesley, London (2003)

16. Sion, R., Tatemura, J.: Dynamic Stochastic Models for Workflow Response Optimization, 2005. In: IEEE International Conference on Web Services (Industry Track) IEEE ICWS (2005)
17. Cardoso, J., Sheth, A., Miller, J.: Workflow Quality of Service. In: International Conference on Enterprise Integration and Modelling Technology and International Enterprise Modelling Conference (ICEIMT/IEMC-02). Valencia, Spain, – p. 13 (2002)
18. Yu, T., Lin, K.-J.: Service Selection Algorithms for Composing Complex Services with Multiple QoS Constraints. In: International Conference on Service-Oriented Computing 2005 – pp. 130–143 (2005)
19. Miettinen, K.M.: Nonlinear Multiobjective Optimization. Kluwer Academic Publishers, Boston (1998)

A Component-Based Process for Developing Automotive ECU Software*

Jin Sun Her, Si Won Choi, Du Wan Cheun, Jeong Seop Bae, and Soo Dong Kim

Department of Computer Science, Soongsil University
511 Sangdo-dong, Dongjak-Ku, Seoul, Korea 156-734
{jsher, swchoi, dwcheun, jsbae}@otlab.ssu.ac.kr, sdkim@ssu.ac.kr

Abstract. Software plays a vital role in operating modern automobiles, and it is a key element in providing innovative features such as Collision Prevention System. There are two essential issues to be resolved; managing software complexity, and reducing software cost and time-to-market. A key solution to these two issues is to maximize reusing components in building various Electronic Control Units (ECUs). Component-based development (CBD) is regarded as an effective reuse technology. However, current CBD methodologies do not effectively support developing reusable automotive components and ECUs. Hence, in this paper, we first define variability types and variation points for ECUs. Based on the variability types, we propose a component-based development process for developing ECUs. To assess the applicability of the proposed CBD process, we present the case study of developing an innovative automotive ECU for Automatic Parking System (APS).

1 Introduction

For modern automobiles, software is regarded as important as mechanical hardware elements since software monitors and controls various hardware devices and components. Moreover, software plays a key role in providing innovative features.

ECU of automobiles is an embedded microcontroller which provides automobile-intrinsic essential functionality such as controlling engine, operating air bags, and controlling traction stability. A ECU consists of several software components, and the components interact with various setpoint generators, sensors and actuators [1].

However, there are two essential issues to be resolved; managing software complexity, and reducing software cost and time-to-market [2]. Software of modern automobiles nowadays now handles around 80 controllers and hundreds of sensors and actuators connected on multiple bus systems. Hence, such software provides up to 2,500 functionalities, and its size can be up to 10 million lines of code.

A key solution to these two issues is to maximize reusing components in building various ECUs. Among the few reuse technologies, CBD is known to be effective for developing automotive software since CBD provides effective features for supporting

* This work was supported by the Korea Science and Engineering Foundation(KOSEF) grant funded by the Korea government(MOST) (No. R01-2005-000-11215-0).

modularity, capturing commonality into components, customizing variability within components, assembling in *plug-n-play* fashion, and maintaining through replacement.

However, current CBD methodologies do not effectively support developing reusable automotive components and ECUs. Moreover, current research works including [1] do not handle variability among various ECUs and automobiles explicitly; rather they emphasize its significance.

Hence, in this paper, we first define types of variability which can occur in ECUs. Based on the variability types, we propose an effective component-based development process for developing ECUs. To show the applicability of the proposed CBD process, we present the case study of developing an innovative automotive ECU for Automatic Parking System (APS). Then we assess the approach in terms of process evaluation criteria.

2 Related Works

AUTomotive Open System Architecture (AUTOSAR) is a standard architecture for automotive software [3]. In this architecture, AUTOSAR provides a process for generating ECU software using reusable components. However AUTOSAR does not address concrete specification of software components and the method for mapping software components into ECU.

Schauffele's research provides the core process for the development of electronic systems and software as well as methods and tools [4]. The software components are implemented through general software life cycle. However, this research does not provide the process for developing reusable automotive software component which embeds variability. In addition, the core process does not include detailed instructions and activities for developing this software.

Hardung's work proposes a framework which supports automotive manufacturers to reuse software [1]. This framework consists of three processes; core asset development process, product development process, and management process. However, this work mainly focuses in classifying the types of software components and relating the process to the environment such as repository and tools. Therefore, specific instructions and guidelines are not sufficiently provided for developing automotive software.

3 Variability of ECU

In this section, we define types of variability which can occur in ECUs. ECU consists of ECU architecture and several software components as shown in Fig. 1. The ECU architecture is a generic structure for an ECU, and it consists of software components and their relationships. Star icon denotes the places where variability occurs.

Variability on Software Component: *Software component* provides the functionality of ECU and it contains attributes for persistent data. *Attribute variability* can occur in a software component and it denotes occurrences of variation points on the set of attributes needed by components. More specifically, variations on attributes can occur

on the different number and/or data types of attributes. For example, attributes of Antilock Brake System (ABS) components for automobile X can be the road surface roughness and speed of the car’s wheel. And, the attributes for automobile Y can include the throttle condition in addition to the two attributes of automobile X.

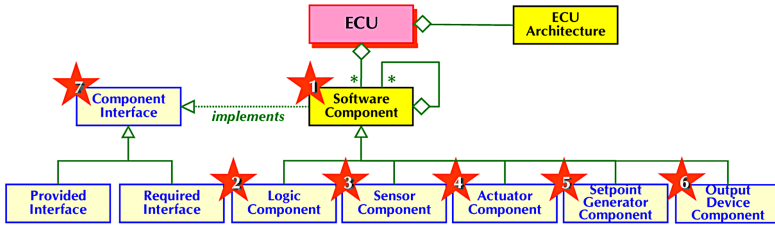


Fig. 1. Places Where Variability Occurs

Variability on Logic Component: *Logic component* implements the control logic and workflow of the ECU and makes decision for providing the functionality. It receives data from setpoint generator and sensor components, and sends the manipulated data to the actuator and output device components. Hence, the variability can occur on logics and workflows. *Logic variability* denotes occurrences of variation points on the algorithm or logic of methods in a component. For example, logics for *computeParkingPath()* method of *Automatic Parking System (APS)* varies according to automobile models such as sedan, SUV, and Van.

Workflow variability denotes occurrences of variation points on the sequence of method invocations. For APS, sequence of invoking *applyBrake()*, *applyAccelerator()*, *setForward()*, *setBackward()*, *turnLeft()*, and *turnRight()* methods varies depending on the automobile models.

Variability on Sensor Component: *Sensor component* acquires various signals and data through hardware sensor elements and sends it to the logic components. *Variability on sensor components* denotes occurrence of variation point on the types of sensors needed. For APS, depending on the types of sensor, different device drivers such as *ultrasonic sensor driver* or *laser sensor driver* are required.

Variability on Actuator Component: *Actuator component* delivers decisions and controlling commands to various hardware actuators. *Variability on actuator components* denotes occurrence of variation point on the types of actuators needed. For example, depending on the types of actuator, different transmission controller such as *4 step transmission controller* or *5 step transmission controller* are needed.

Variability on Setpoint Generator Component: *Setpoint generator component* is to acquire input from hardware setpoint generators. *Variability on setpoint generator components* denotes occurrence of variation point on the types of setpoint generators needed. For APS, *Parking Button* can be a setpoint generator component. Parking button can be implemented as *touch pad* on LCD or *switch type button* on dashboard.

Variability on Output Device Component: *Output device component* outputs the status of the automobile, needed information, audible data, and so on. *Variability on*

output device components denotes occurrence of variation point on the types of output devices needed. For instance, LCD software component appears differently depending on the LCD resolution type such as 480*320 or 640*480.

Variability on Component Interface: Software components implement pre-defined *Component Interfaces*. *Variability on component interface* denotes occurrences of variation points on the method signatures of the component interface. For a same functionality, each ECU may have its own convenient form of API, i.e. method name, orders and types of parameters, and its return type.

4 Component-Based Process for ECU

In this section, we propose two sub-processes for developing ECUs; *Component Engineering* (CE) and *ECU Engineering* (EE) as in Fig. 2. The CE process is to engineer reusable components, and the EE process is to develop target ECUs by reusing components.

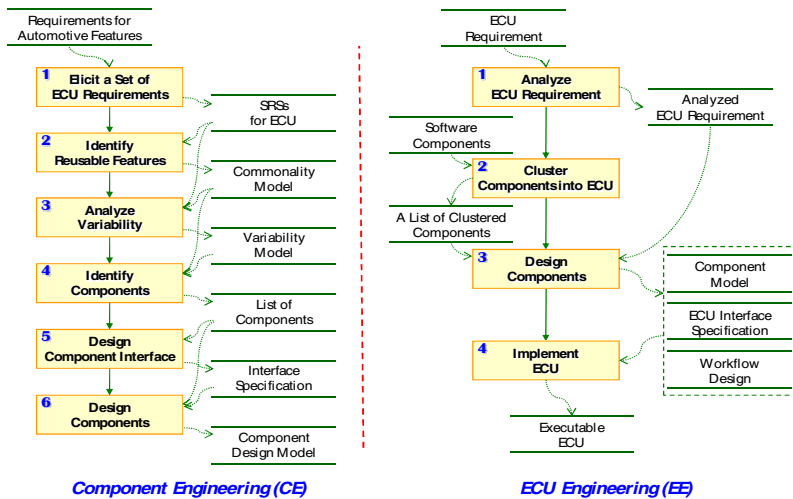


Fig. 2. Process for Developing ECUs

4.1 Component Engineering (CE)

CE-1. Elicit a Set of ECU Requirements: This activity is to elicit a set of ECU requirements from automotive requirements for potential ECUs in a domain. We suggest the following four steps for this activity. The first step is to decide the scope of ECUs because there are many kinds of requirements, constraints, and field of automotive development. The second step is to trim the acquired requirements for the ECU scope because the acquired requirements may cover different scopes. The third step is to extract the software requirement from system requirements which consist of software and hardware requirements. The fourth step is to normalize the various ECU

software requirements. The acquired requirements may mostly be inconsistent in their vocabulary used, the writing styles and the levels of description details. Finally we define a list of features, for each normalized software requirement specification (SRS) which includes functional and non-functional requirement.

CE-2. Identify Reusable Features: This activity is to analyze the normalized requirements and to extract features that are common to a number of target ECUs. We suggest two criteria, *Commonality* and *Business Value*, to determine whether the features identified in CE-1 will be designed into reusable components or not.

We suggest the following three steps for this activity. The first step is to measure reusability for each feature which is identified in CE-1. The criterion of *Commonality* is to measure how many target ECUs requires each feature. This can be computed by using $Comm(F_i)$ metric; $Comm(F_i) = \frac{\sum_{j=1}^m E_j(F_i)}{\sum_{k=1}^m E_k}$, where $\sum_{j=1}^m E_j(F_i)$ is the summation of ECU which requires feature_i (F_i) and $\sum_{k=1}^m E_k$ is the number of total ECUs in the normalized requirement specification. The decision whether a feature is common or not can be made by feature comparison table. This table consists of feature list, feature description, and a degree of commonality.

The criterion of *Business Value* is to evaluate the business value of each feature. This evaluation can be made on various value appraisal factors such as *market demand*, *financial sponsorship*, *high mark-up*, and *future value*. The complete list of such factors cannot be defined due to the diversity of ECUs and project situations. Hence, we define a generic metric where different sets of factors can be uniformly applied; $BizValue(F_i) = \frac{\sum_{j=1}^k I_j(F_i)}{k}$, where $\sum_{j=1}^k I_j(F_i)$ is the summation of all business value item (Val_j) for the feature_i (F_i) and k is the number of business value.

Now, we combine two metrics into a single metric, $Reusability(F_i)$ for each feature. We use weight values for the two criteria.

$Reusability(F_i) = Comm(F_i) * W_{comm} + BizValue(F_i) * W_{value}$, where W_{comm} is the weight for commonality and W_{value} is the weight for the business value. We define a reusability range/degree of features to decide whether a feature can be reusable feature or not after measuring reusability for each feature. The criteria for reusability range of features are applied in various projects differently due to the diversity of projects.

The second step is to specify a family ECU SRS based on reusable features from a set of normalized SRS. This family ECU SRS is a basis for performing CE-3 through CE-6. The third step is to define a *Commonality Model*. The commonality model includes a feature diagram which consists of the reusable features and description of reusable features [5]. The set of reusable features identified here determines the scope of components that will be identified.

CE-3. Analyzing Variability: This activity is to analyze the variability of reusable features [6]. That is, we need to identify minor difference of a reusable feature for the diversity of ECUs. Variability of a reusable feature is a minor difference on attributes, logics, workflows, and interfaces [7][8]. Hence, the variability exists within the commonality. For each reusable feature, we identify variation points and their types referring to the variability types defined in section 3.

We suggest following three steps for this activity. The first step is to define variability comparison table. For each reusable feature, compare all the feature

description in the commonality model. If a minor variation occurs on attribute, logic, or workflow among the requirement specifications for each reusable feature, the feature contains variability. The second step is to define variable feature table. Once variable features are identified, define the name of variable feature, variation point, variation point type, set of variant, and scope of variation point. The third step is to add variability to the feature diagram. Finally, variability model includes variability comparison table, variable feature table, and feature diagram with variability.

CE-4. Identify Components: This activity is to identify reusable components from the identified reusable features. We suggest two step identification technique based on the bottom-up approach suggested in [9]. The first step is to cluster related features from the feature diagram in CE-2 to identify preliminary component list according to the rules suggested in [9] and [10]. According to the identified type of component, sensor, actuator, setpoint generator, or output component variability may occur. The second step is to conduct structural modeling. We analyze the structural view to revise the preliminary component list. This is because identifying component is affected by data coupling and cohesion such as class hierarchy and relationship among classes. The preliminary component list is revised by clustering the classes according to the rules suggested in [9] and [10]. Then, a domain expert adds additional reusable components by considering ECU profile and business ROI. This activity delivers a list of components with the list of features and classes assigned to each component.

CE-5. Design Component Interfaces: This activity is to design the interfaces for the identified components. The first step is to define provided interface by referring to the feature lists of each component. The second step is to define required interface which specifies the external services needed to fulfill the functionality of the component. When defining the provided and required interfaces, ECU level interfaces and workflow design can be considered. Typically, interfaces are classified into component interfaces and ECU interfaces as in Fig. 3.

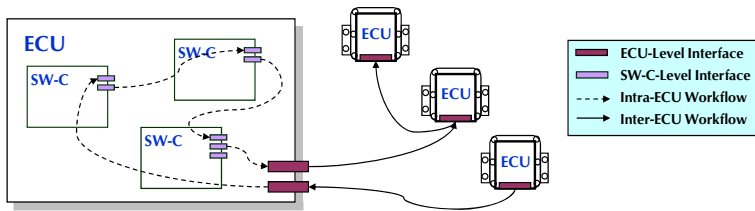


Fig. 3. Two Kinds of Interfaces in Automotive Software

Component interface is used for interacting among components within one ECU, whereas ECU interface is used for interacting among ECUs. Workflows can also be classified into intra-ECU workflow and inter-ECU workflow as in Fig. 3. By referring to the intra-ECU workflow, we can identify additional interfaces. In addition, interface variability can occur on the identified interfaces. Therefore, we should identify the variation points and variants for them. The third step is to specify

interface with related components, interface signature, interface type, pre-condition, post-condition, and constraints.

CE-6. Design Components: This activity is to refine the internal details of each component and design variability within components. When designing components, the implementation environment such as development language and platform should be considered. We suggest a three-step instruction.

The first step is to refine the structural model from CE-4. In this step, we first determine the target platform and language such as C++ and Java. And then we refine conceptual class diagram to platform specific class diagram which has platform specific data type, method signature, and data structure.

The second step is to represent the variability in the class diagram and design variability mechanism in detail. In order to present and design variability, we should refine feature level variability information into class level variability information. When designing the variability mechanism, various techniques such as selection technique, plug-in technique, and external profile technique can be considered [6].

The third step is to design the dynamic model. Based on the platform specific class diagram, we design a data flow diagram and platform specific sequence diagram. Workflow variability is represented on the data workflow diagram and platform specific sequence diagram.

4.2 ECU Engineering (EE)

EE-1. Analyze ECU Requirement: This activity is to analyze the requirement for a target ECU. ECU requirement may come from the previous projects or be newly developed by considering the current needs and future expectations. We suggest analyzing the requirement from two views; *functional* and *non-functional view*.

The first step is to analyze the functionality of a target ECU. Here, we need to analyze functional, structural, and behavioral models by using any conventional method such as *structured analysis* and *object-oriented analysis*. We suggest using *Use Case Model* for depicting functionalities, *Data Flow Diagram* or *Class Diagram* for analyzing required and provided data, and *Sequence Diagram* or *State Machine Diagram* for representing workflows of the target ECU. Additionally, we suggest *Function Type Table* for identifying open loop and closed loop which are used as the basis for analyzing dynamic view and comprehending functionality.

The second step is to analyze non-functional requirement such as quality requirement, real-time requirement, hardware constraints, and resource constraints. We suggest *Quality Requirement Specification* for specifying quality attributes. It consists of the name of quality attribute, description of each attribute, and priority among them. We also suggest a *Time-Schedule Table* for specifying real-time requirement. It consists of the name of function, execution time, response time, and activation/deadline point among them.

EE-2. Cluster Components into ECU: This activity is to identify and group the components needed to fulfill the requirement of the target ECU. Like the ECU of AUTOSAR, a ECU typically consists of several components [3]. We now propose a four-step metric based instruction.

The first step is to comprehend the *specifications* of available components in terms of their functionality, interfaces, constraints, non-functional requirement, and variability. The second step is to compare the available components to the ECU requirement to determine candidate components. As shown in Fig. 4, we consider the functional conformance of each component when determining the candidate components. Functional conformance can be acquired from the metric below. And here are the terms used in the metric. Let *AvailableComp_i* be an available component, and *CandidateComp_j* be a candidate component which is selected from the available components. Let *F_i(S)* be the *ith* function of *S* where *S* can be a component or a ECU.

$$\begin{aligned}
 & \text{FunctionalConformance}(F_i(\text{AvailableComp}_n), F_i(\text{ECU})) \\
 &= \frac{\text{Number of Syntactic and Semantic Elements Provided by } F_i(\text{AvailableComp}_n)}{\text{Number of Syntactic and Semantic Elements Required from } F_i(\text{ECU})}
 \end{aligned}$$

where syntactic elements include data, logic, workflow, and interface, and the semantic elements include precondition, postcondition, invariant, and side effect. Note that we should also consider variants provided by each component. According to the value of this metric, we can determine the components to be discarded and those to be customized in the further activity.

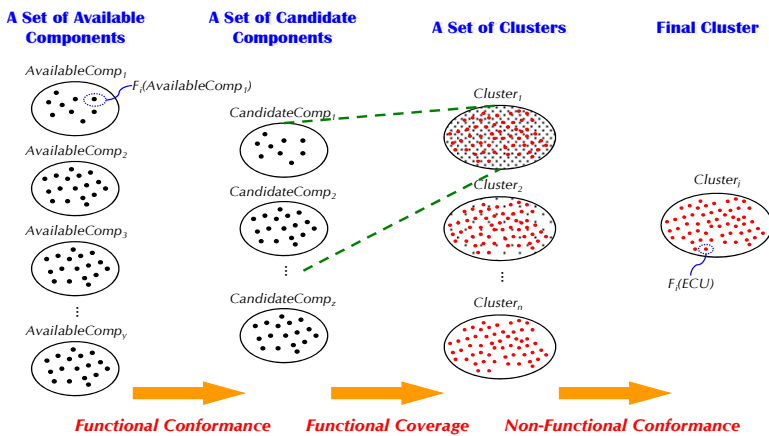


Fig. 4. Metrics Used in Clustering Components

The third step is to define all possible clusters of components which satisfy the functionality of the target ECU. As shown in Fig. 4, functional coverage is used to define the clusters. Let *Cluster_i* be a set of *CandidateComp*, and let *SetF(Cluster_i)* be the union of the set of functions of the candidate components allocated into the *Cluster_i*. We should select the clusters with *FunctionalCoverage(Cluster_i)* near 1.

$$\text{FunctionalCoverage}(Cluster_i) = \frac{\text{Number of Functions in } SetF(Cluster_i)}{\text{Number of Functions in } SetF(ECU)}$$

The fourth step is to select a cluster of components which highly satisfies the nonfunctional requirement of the target ECU with no complication. This can be determined by using the metric, *NonFunctionalConformance*.

$$\text{NonFunctionalConformance}(\text{Cluster}_i) = \frac{\text{Number of } REQ_{NFR} \text{ Provided by Cluster}_i}{\text{Number of } REQ_{NFR} \text{ Required from Target ECU}}$$

Here the REQ_{NFR} is the quality requirement, real time requirement, cost and memory size preferences. Finally, this activity outputs a list of clustered components that will be allocated to the ECU.

EE-3. Design ECU: This activity is to design the internal details of each target ECU. This will include customizing the clustered components and defining ECU workflow between the clustered components. We propose a five-step instruction.

The first step is to define ECU interfaces for invoking services provided by the ECU. ECU interfaces can be selected among component interfaces or be newly defined by considering the connection between ECU and other devices such as sensor, actuator, or virtual function bus (VFB), etc. The second step is to resolve any conflicts among components since clustered components from EE-2 can have overlapped functions between components. Since resources of ECU are restricted, we remove the redundant functions during the customization. The third step is to design the workflow of method invocations among components referring to the behavioral models and time-schedule table from EE-1. The fourth step is to customize components which require customizations. By following the customization instruction in the component specification, we supply the right variant for the target ECU. Common customization techniques are parameterization, selection, plug-in, and external profiles. The fifth step is to design ECU specific components. Since customized components cannot fully satisfy functionalities of ECU requirements, the ECU specific components are designed. Then, a ECU specific component model is integrated with the customized components. The deliverables of this activity are ECU interfaces, workflow, and component model.

EE-4. Implement ECU: This activity is to implement the target ECU. This activity takes ECU specification, ECU interface specification, ECU workflow and produces ECU executable. We propose three-step instruction.

The first step is to generate ECU interface from ECU interface specification. Then, we implement the workflow among components according to the workflow design. Furthermore, we may also implement additional design elements such as connectors to make the cluster of components fully satisfy the ECU requirements. The second step is to simulate the generated code before it is applied to real world. The specific ECU software is generated in this step. The time scheduler such as RTOS scheduler is also generated. The third step is to port and configure the generated code for ECU software. The executable code is loaded into ECU hardware. The technical simulation is performed in this step.

After performing the four activities, typical verification and validation activities are followed, including integration test, acceptance test and system test.

5 Case Study

We conduct a case study of applying the proposed development process; *CE* and *EE* on a ECU for an innovative automotive feature called *Automatic Parking System (APS)*. Due to the paper length, we only show the representative results of applying the activities.

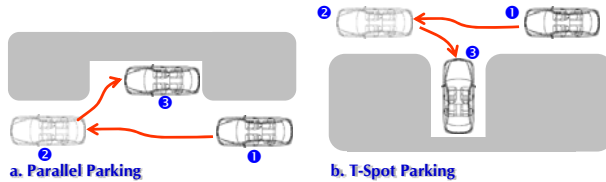


Fig. 5. Two Modes of Automatic Parking Systems

CE-1. Elicit a Set of ECU Requirements: There are different levels of automation and different methods of automatic parking. We gathered requirement specifications for three different automobile models. Although there are minor variations, the general functionality of APS is to find the optimal path for parking by measuring the distance between the automobile and its surrounding objects, and to drive the car automatically without the driver’s intervention. There are two basic modes for APS, as shown in Fig. 5.

CE-2. Identify Reusable Features: Table 1 is a result of comparing features among three automobile models. 33 reusable features are identified from three domains according to criteria of commonality and business value.

Table 1. Reusable Feature List

Reusable Feature ID	Domain	Reusable Feature Name	Family Member			Remarks
			H	D	K	
RF01	Parking Region Definition	Create Parking Path	✓	✓	✓	Rule i)
...		
RF09		Store Parking Path	✓	✓	✓	Rule i)
RF11		Detect Obstacle	✓	✓	✓	Rule i)
RF12	Parking Path Creation	Calculate Distance	✓	✓	✓	Rule i)
...	
RF19		Create Path Line	✓	✓	✓	Rule ii) + BizValue
RF23		Search Tracks	✓	✓	✓	Rule i)
RF24	Parking Execution	Alert Alarms	✓	✓	✓	Rule i)
RF27		Adjust Steering	✓	✓	✓	Rule i)
RF28		Active Throttle	✓	✓	✓	Rule i)
RF29		Active Brake	✓	✓	✓	Rule ii) + BizValue
RF32		Alarms Fault	✓	✓	✓	Rule i)
...		

CE-3. Analyzing Variability: The variability is analyzed from the commonality model of CE-2. We analyzed six variable features among 33 reusable features; RF01, RF02, RF05, RF08, RF10, and RF11 as shown in Table 2.

Table 2. Variable Feature Table

VP ID	Variable Feature	Variation Point	Variation Point Type	Set of Variants			Scope of Variation Point
				H	D	K	
VP01	RF01	Workflow of Create	Workflow	Detecting Obstacles → Generate PathID → Calculate Distance → Generate Path → Display Path	...	Detecting Obstacles → Calculate Distance → Generate Path → Display Path	Selection
...
VP06	RF11	Workflow of Alarms Fault	Workflow	Detect Fault → Determine Criticalness → Display Info. → Alarm to Driver	Detect Fault → Determine Criticalness → Alarm to Driver → Display Info		Selection

CE-4. Identify Components: From the reusable features of CE-2, we identified ten components for sensors, actuators, setpoint generators, and logic types as shown in Table 3.

Three sensor components are to acquire the car speed, video scene, and the distances constantly during automatic parking, and the four actuators are to control the engine throttle, transmission mode (either forward or backward), steering direction, and brakes. Two logic components are to find the optimal parking path and to execute the parking by using sensor and actuator components. The setpoint generator component *Parking Mode Selector* lets the driver to select the various automatic parking options.

Table 3. A Part of Component List

Component		Feature	Domain
Type	Name		
Actuator Component	Throttle Controller Component	Control Throttle	Powertrain
		Calculate Throttle Position	
		...	
	Transmission Controller Component	Shift Transmission	Powertrain

Logic Component	Path Finder Component	Compute Parking Path	...
	

CE-5. Design Component Interfaces: In this activity, we designed workflow and defined interfaces for the components from CE-4. To define a set of interfaces for the components, we first explored the functionalities. For example, Parking Executor component controls wheel, steering, throttle valve and fuel injector. To control these actuators, the calculated path for specific parking spot is needed. We defined these controls as the provided interfaces such as `setWheelAngle()`, `setSteeringAngle()`, `regulateThrottleValve()`, and `regulateFuelInjector()`. And we defined the requiring services as the required interfaces such as `getParkingPath()`. Then we finally specified these interfaces in terms of related component, interface signature and type, pre- and post- condition, and constraints as shown in Table 4.

Table 4. Interface Specification for *ParkingExecution*

Related Comp.	Interface Signature	Interface Type	Pre-condition	Post-condition	Constraints
parking Execution	setWheelAngle (turningRadius, turningAngle)	Provided	Initialization of wheel angle (0)	Regulation of wheel angle (turning angle)	Wheel angle can be regulated from 0°~42°.

CE-6. Design Components: In this activity, we designed a class for each component. For example, *PathFinder* component has one class with 13 attributes and 11 operations. As a result, we performed this activity using the object-oriented approach, due to the granularity of components.

EE-1. Analyze ECU Requirement: In this activity, we analyze the APS, for a simulator vehicle, *SSU-SUV-7* which is a new SUV model. The APS uses an ultrasonic sensor for calculating the distance from the vehicle to object. The gap between the vehicle and object is less than 10 inches. The APS button on LCD is used for setpoint generator.

EE-2. Cluster Components into ECU: To acquire a set of components for the APS ECU, we first comprehended the specifications of the 46 available components and extracted 24 candidate components. Using the candidate components, we derived 9 clusters of components, and finally acquired a cluster with 10 components as shown in Fig. 6.

To select the candidate components, we computed the functional conformance for the functions. For example, the *compute steering angle* function of the *ParkingExecutor* component has the functional conformance of 0.875 since the number of syntactic and semantic elements provided by *compute steering angle* function of the *ParkingExecutor* component was 7 and those required from the ECU was 8. We get 24 candidate components by selecting the components with high functional conformance. Then, we made 9 matches of clusters with the functional coverage near 1. For each cluster, we computed the non-functional conformance and *cluster 3* had the highest value.

Hence, the final list of components is *SpeedSensor*, *Camera*, *DistanceSensor*, *ParkingRegionAnalyzer*, *PathFinder*, *ParkingExecutor*, *TransmissionController*, *ThrottleController*, *SteeringController*, and *BrakeController*.

EE-3. Design ECU: In this activity, we customized the setpoint generator component, *Parking Mode Selector*, for a simulator vehicle, *SSU-SUV-7* which is a new city SUV model. Using the same APIs, we only customized the visual interfaces using a LCD touch screen.

In this activity, we also designed ECU interface and ECU workflow. Fig. 7 shows a workflow for executing the parallel parking. This workflow shows normal flow for the parking. As an exceptional flow, collision may be detected, and in this case the system controls the brake of vehicle.

EE-4. Implement ECU: In this activity, we show the pseudo code of computing parking path which is invoked during the parking execution workflow of Fig. 7. As shown in Fig. 8, a parking path is acquired by computing the way from the car's initial position to the target parking space.

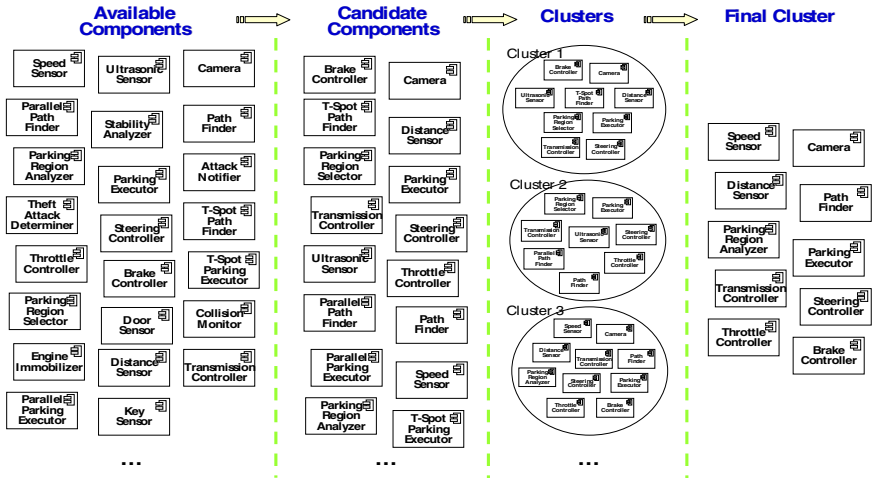


Fig. 6. Clustering Components

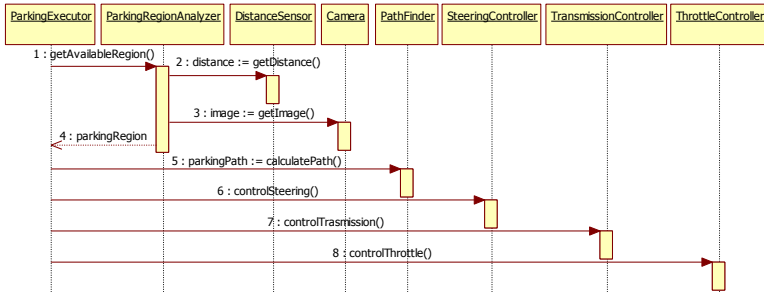


Fig. 7. Workflow for Parallel Parking

```

PathFinder PathFinder::calculatePath(CarPosition initialPosition, TargetParkingSpacePosition
lastPosition){
    ...
    bool side;
    //Calculate Direct Path
    side = decideParkingSide(frontLeftWheelCoordinateX, spaceFrontLeftX,
        frontRightWheelCoordinateX);
    turningWheelCoordinateX = decideParkingSideX(side);
    turningWheelCoordinateY = decideParkingSideY(side);
    ...
    //Calculate First Turning Path
    if (side == RIGHT){
        turningWheelCoordinateX = decideParkingSideX(LEFT);
        turningWheelCoordinateY = decideParkingSideY(LEFT);
        idealCoordinateY = (spaceFrontLeftY + spcaeRearLeftY) / 2 + (wheelBase / 2);
    }
    ...
    return PathFinder;
}
    
```

Fig. 8. Pseudo Code of Computing Parking Path

6 Assessment

6.1 Assessment by Evaluating the Process

In this section, we compare the proposed process with three related works specified in section 2, in terms of process evaluation criteria.

Table 5. Comparison Result

Criteria \ Processes	AUTOSAR	Schauffele's	Hardung's work	Suggested Process
Systematic Activity	○	●	○	●
Precise & Stepwise Instructions	-	-	-	○
Conciseness	○	△	△	●
Comprehensiveness	△	△	△	●
Applicability	○	△	○	○
Specification of Artifacts	△	△	-	○
Traceability	△	-	-	●

● Well supported ○ Supported △ Partially supported - Not supported

Here are the criteria that a process should adhere to. The criterion of *systematic activity* is to evaluate if the set of activities or the sequence of activities are logically defined. The criterion of *precise and stepwise instructions* is to evaluate if the instructions of each activity are defined in a detailed and concrete manner. The criterion of *conciseness* is to evaluate if the set of activities, instructions, and artifacts are concisely defined. Too many activities or artifacts should make the process more complicated. The criterion of *comprehensiveness* is to evaluate if the instruction sets

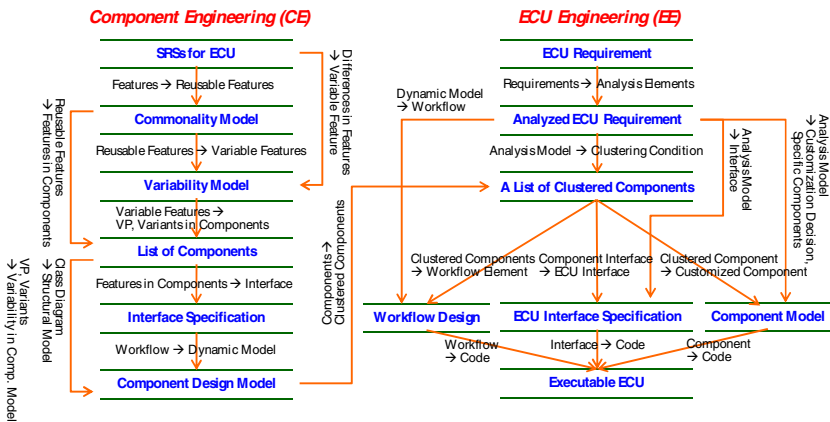


Fig. 9. Traceability among Artifacts

are described easily and concretely to be followed. The criterion of *applicability* is to evaluate if the activities, instruction sets and the artifacts are applicable in real projects. The criterion of *specification of artifacts* is to evaluate if the key elements and templates of the artifacts are concretely defined. The criterion of *traceability* is to evaluate if the elements of the artifacts between the preceding and the following activities can be traced.

As evaluated in Table 5, the process is given with systematic activities and the activities are given with stepwise instructions. We showed that the process is applicable by applying the process to APS. And it can be said that traceability between artifacts are well supported since the elements among the artifacts can be traced as shown in Fig. 9.

6.2 Assessment by Evaluating ECU Specific Criteria

In this section, we assess our approach if it well supports the requirements of an ECU engineering process.

AUTOSAR Compliance. Our proposed meta- model of ECU conforms to the ECU structures specified in AUTOSAR [3] and implied in SAE International [1]. By conforming to AUTOSAR, it becomes possible to focus on developing reusable software components which are independent to hardware.

ECU Specific Variability. The six types of variability we identified were derived from technical observations on variability among existing automobile parts. And, the variation points identified during the case study were modeled with the six variability types we defined.

ECU Engineering Specific Activities. Defining components, clustering components into ECU, defining ECU level interfaces are the essential activities of ECU engineering. Identifying commonality and variability are also the activities that are on high demand nowadays in ECU engineering.

7 Concluding Remarks

Software is a key component of modern automobiles since software monitors and controls various hardware devices and components. Software is essential in implementing innovative and intelligent automotive features. Software is presented in the ECU unit of automobiles. We pointed out two issues in designing ECUs; managing software complexity and reducing the software cost. Component-based development (CBD) is believed to be effective in resolving the issues. Especially, variability management of CBD can increase the reusability of ECU software components and so reduce the software cost.

Hence, in this paper, we first identified six places where the variability may occur in ECU. Interdependency between variability types emerges as an issue here and this is left as a future work. Based on the variability types, we proposed a component-based development process for developing ECUs. The process consists of two sub-processes; *Component Engineering* and *ECU Engineering*. And, we provided instructions for all the activities. We also conducted a case study of applying the

process in implementing *Automatic Parking System*. Through the case study and the assessment in section 6, it is shown that ECUs can be developed in a cost-effective way by reusing ECU components.

References

1. Hardung, B., Kolzow, T., Kruger, A.: Reuse of Software in Distributed Embedded Automotive Systems. In: Proceedings of the 4th ACM International Conference on Embedded Software, pp. 203–210 (2004)
2. Broy, M.: Challenges in Automotive Software Engineering. In: Proceeding of the 28th International Conference on Software Engineering (ICSE '06), pp. 33–42 (2006)
3. AUTOSAR, version 2.0.0, (March 2006), www.autosar.org
4. Schaufele, J., Zurawka, T.: Automotive Software Engineering: Principles, Processes, Methods, and Tools, SAE International (2005)
5. Griss, M.: Product-Line Architecture. In: Chapter 22 of Component-Based Software Engineering, Addison Wesley, London (2001)
6. Kim, S., Min, H., Rhew, S.: Variability Design and Customization Mechanisms for COTS Components. In: Gervasi, O., Gavrilova, M., Kumar, V., Laganà, A., Lee, H.P., Mun, Y., Tanian, D., Tan, C.J.K. (eds.) ICCSA 2005. LNCS, vol. 3480, pp. 57–66. Springer, Heidelberg (2005)
7. Kim, S., Her, J., Chang, S.: A Theoretical Foundation of Variability in Component-based Development. Information and Software Technology 47, 663–673 (2005)
8. Pohl, K., Bockle, G., Linden, F.: Software Product Line Engineering: Foundations, Principles, and Techniques. Springer, Heidelberg (2005)
9. Choi, S.W., Chang, S.H., Kim, S.D.: A Systematic Methodology for Developing Component Frameworks. In: Wermelinger, M., Margaria-Steffen, T. (eds.) FASE 2004. LNCS, vol. 2984, pp. 359–373. Springer, Heidelberg (2004)
10. Kim, S., Chang, S.: A Systematic Method to Identify Software Components. In: Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC '04), Busan, Korea, pp. 538–545 (2004)

A Systematic Approach to Service-Oriented Analysis and Design*

Soo Ho Chang and Soo Dong Kim

Department of Computer Science
Soongsil University, Seoul, Korea
shchang@otlab.ssu.ac.kr, sdkim@ssu.ac.kr

Abstract. Service-oriented computing (SOC) has several unique features which are not typically presented in conventional object-oriented development (OOD) and component-based development (CBD), including the commonality of service functionality, publish and discovery paradigm of services and dynamic composition of possibly 3rd party's independent unit services. Hence, OOD and CBD-based analysis and design methods are not effective and expressive enough to model service-oriented applications. Rather, Service-Oriented Analysis and Design (SOAD) has to be defined by using some of the two modeling paradigms and by adding SOC-unique modeling mechanisms. In this paper, we first present a technical comparison among OOD, CBD and SOAD to derive the design criteria for SOAD. And, we define the key artifacts that have to be delivered during SOAD. Based on this, we present a SOAD process which takes service requirements as the initial input and delivers service specifications, compositions, and verified service components as the final deliverables. Each of the five phases in the process is defined with its overview, artifacts, and work instructions. Finally, we present a case study of applying our process to a service domain to reveal its effectiveness and applicability. Once the proposed SOAD is well applied, SOAD artifacts can be more systematically and efficiently implemented with current SOA standards.

1 Introduction

Service-Oriented Computing (SOC) is emerging as a new paradigm for building and maintaining applications in a cost effectiveness way. With increased support with SOA tools, the paradigm of *publish-discover-compose* services becomes more common, and it has significant benefits for ROI perspective as well as technical perspective. However, there is a considerable amount of gap between the promises of SOC and the maturity of service engineering methodology. The main cause of the gap is the lack of effective analysis and design methods. That is, there is a great demand on effective service-oriented analysis and design (SOAD) methodology [1].

Service-oriented computing (SOC) has several unique features which are not typically presented in conventional object-oriented development (OOD) and component-based

* This work was supported by Seoul Research and Business Development Program (10557).

development (CBD). In SOC, services should be designed not just for a dedicated client, but for a family of potential clients. Hence, the commonality of services should be well modeled into service components. Services are not tightly coupled to certain client applications; rather they are published in service repositories, discovered by clients dynamically, and composed to fulfill the functionality expected by clients. Hence, SOAD must provide facilities for supporting SOC-specific features in addition to conventional software modeling facilities.

Since current SOA standards are largely based on the object-oriented paradigm and CBD, existing OOD and CBD methods may seem to be useful for SOAD. However, OOD and CBD methods by themselves are not effective and expressive enough to model service-oriented applications. Rather, SOAD has to be defined with SOC-specific modeling facilities on top of the two existing paradigms.

Hence, our research goal is to define a systematic and effective SOAD which extends OOD and CBD. In this paper, we first present a technical comparison among OOD, CBD and SOAD to better understand their similarities and differences in Section 3. From the comparison, we derive the design criteria for our SOAD method. And, we define the key artifacts that have to be delivered during SOAD in Section 4. Based on these preliminary works, we propose a SOAD process which takes service requirements as the initial input and delivers service specifications, compositions, and verified service components as the final deliverables in Section 5. Each of the five activities in the process is defined with its overview, artifacts, and work instructions. We present a case study of applying our process to a service domain to reveal its effectiveness and applicability in Section 6. Once the proposed SOAD is well applied, SOAD artifacts can be more systematically and efficiently implemented with current SOA standards of WSDL [2], UDDI [3], BPEL[4] and SOAP[5].

2 Related Works

Zimmermann's Work introduces an integration of Business Process Modeling (BPM), Enterprise Architecture (EA), and Object-oriented Analysis and Design (OOAD) [6]. It introduces three major level of abstractions in SOA; operations as single logical units of a work, services as logical groupings of the operations, and business processes. It also represents how the three methods can be applied to service-oriented analysis and design, what should be more defined.

Arsanjani's work presents seven layers of SOA; *operational system layer, enterprise component layer, services layers, business process choreography layer, presentation layer, integration layer, and QoS layer* [7]. For each of these layers, design and architectural decision activities are needed. Especially, they proposes five activities called service identification, service classification or categorization, subsystem analysis, component specification, service allocation, and service realization, which are performed in service-oriented modeling.

Erl addresses a service oriented-analysis and design process with subordinate instructions [8]. The work proposes three steps for service-oriented analysis and five steps for service-oriented design process. Especially, in the service modeling step in service oriented analysis, service candidates, candidate service compositions, application services, and service operations are modeled. Based on the analyzed

business process, three layers, *business process layer*, *service interface layer*, and *application layer*, are designed with SOA standards such as BPEL, WSDL, and SOAP.

There are more works on service-oriented analysis and design such as [9] and [10]. Those works emphasize identification and specification of services as well as integration of the services. And, in the most works, services are derived from business process. However, defining approaches or methods to service-oriented analysis and design is still in early stage. They generally represent what should be done rather than mentioning how can be done. For more comprehensive and detailed analysis and design method, it would be refined with step-wised process with concrete artifacts and instructions.

3 Comparison of OOAD, CBD, and SOAD

OOAD is widely used in developing various applications [11]. CBD is another common development paradigm which extends OOAD with the notion of *component*, which is a larger grained unit than objects [12][13]. Both OOAD and CBD are well supported by major middleware standards such as CORBA, EJB and .NET. In this section, we compare the three approaches to highlight the similarities and differences as the basis for defining our SOAD method.

We first derive comparison criteria from the perspectives of software process and methods; *Reuse Unit*, *Reuse Coverage*, *Variability Management*, *Interface*, *Aggregation Facility*, *Key Artifacts*, *Artifact Representation* and *Representative Processes*. The comparison is summarized as in the Table 1.

Table 1. Comparison among OOAD, CBD, and SOAD

	OOAD	CBD	SOAD
Reuse Unit	Object/Class	Component	Service
Reuse Coverage	Application-specific	Reusable across Applications	Reusable across domains
Variability Management	Polymorphysm	Port, Requires Interface, Customization	Service Adaptation (Immature)
Interface	Provided Interfce Bundled with Objects	Interface separated from Components	WSDL Interface separated from Component Interfaces
Platform Independence	Dependent	Dependent	Services are independent due to XML-based standards
Aggregation Facility	Composite Objects	Component	Composition s.a. Orchestration and Choreography
Key Artifacts	Objects, their relatinoships	Component, Interface	Process, Service, Composition, ...
Artifact Representation	UML	UML	SOA: BPEL, WSDL, UDDI, SOAP
Representative Processes	RUP	Catalysis, etc.	None Yet

All three approaches provide facilities for increasing reusability, but at different degrees. CBD provides enhanced reusability through interfaces and coarse-grained components, and SOAD even extends the CBD reusability with the notion of *publish-discover-compose* paradigm.

For variability management, OOAD is limited to polymorphism (i.e. overloading and overriding), and CBD provides additional mechanisms of component customization and connectors [14]. In SOAD, variability management is considered to be an essential requirement for successful SOC, but there is not yet a strong consensus on how the variability should be handled. There only exist some preliminary works on service adaptation [15] [16].

For the interfaces, there is no standard for representing interfaces in CBD. However, interface specification in WSDL is common for SOC. Moreover, component interface is realized for specific implementation platforms, whereas WSDL interfaces of services are independent from any specific implementation platforms.

For aggregating smaller-grained functions or elements into a large one, CBD provides a unit of component, and SOAD adds a composition mechanism such as orchestration and choreography. Also, the task of service discovery for composition is new.

For the artifacts, CBD provides more artifacts than OOAD, and SOAD provides even more artifacts as shown in the table. For the artifact representation, SOC provides more standards such as WSDL, UDDI, BPEL and SOAD. While there exist several representative processes for OOAD and CBD, there is neither a standard for SOAD process nor detailed SOAD instructions yet [9].

From the comparison, we derive criteria for designing our SOAD. Fundamental modeling facilities of OOAD and CBD can also be applied to SOAD. For example, the notion of component in CBD can still be used in implementing service components. Separating interfaces from components in CBD should be same for SOAD. However, SOAD needs to be enhanced with additional facilities such as service modeling, service interface design, and composition. SOAD should be defined for additional SOC artifacts such as WSDL interfaces, service registry in UDDI, and composition specification in BPEL. Moreover, a well-defined SOAD process along with engineering instructions is demanded.

4 Layers of Service-Oriented Architectures

SOA-based applications typically embed some form of layered architecture [8][17]. Among the different architectures, we identify four layers for the purpose of service analysis and design elements as shown in Figure 1. Each layer is defined with its goal and key artifacts, in subsequent sections.

Business Process Layer: As the top layer, *Business Process Layer* is to define business processes expected by service clients. *Business Process (BP)* represents a cohesive unit of the service perceived by service clients, not by component engineers. Hence, it is defined independently from implementation technology and platforms. Typically, a BP is a larger-grained than a use case and a method of objects, and it is defined with a *service workflow* among smaller-grained *activities*.

The goal of this layer is to specify all the business processes expected by clients, and so they are used as the basis for further engineering works such as identifying unit services, interfaces and eventually service components.

The key artifacts of this layer are *Business Process Specification*, which includes *workflows* of participating *activities*. Each activity is defined with its functional and non-functional requirements and any constraints.

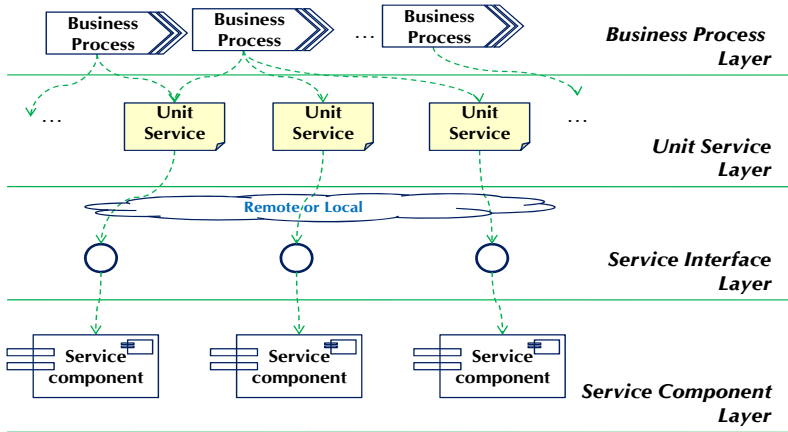


Fig. 1. Key artifacts of SOAD in the four layers

Unit Service Layer: An *activity* of a workflow (i.e. a business process) is a conceptual unit of work, perceived by clients. It will eventually be performed by a software element, which we call a unit service, *UnitService*. That is, an activity is fulfilled by running a unit service. The main distinction between them is that an activity is a conceptual unit perceived by clients and a *UnitService* is its corresponding task defined from engineering perspective. Hence, the notion of *UnitService* is a vehicle that bridges clients' view to engineers' view.

Another key value of introducing *UnitService* is that a *UnitService* can be reused by more than one activity. That is, we analyze the various activities of the workflows, and define a set of unit services. Some unit services may be common among the business processes, and hence they are reusable in several business processes.

The relationship between activities and unit services can be in different forms, as shown in Figure 2. An activity itself can be decomposed into even smaller-grained sub-activities such as *activity 1.2* and *activity 1.2.2*. A *UnitService* can map to an activity of any granularity as long as the functionality and extra-functional property are conformed. Also, a group of multiple (sub) activities can be map to a single *UnitService*. Hence, there is a many-to-one relationship between the activities and the unit services and the figure shows several different mapping relationships. The motivation for elaborating different forms of mappings is to maximize the identification of unit services which can be reused by various activities and sub-activities. Once unit services are identified, it is specified with its *service functionality*, an *interface*, *pre and post conditions*, and *constraints*.

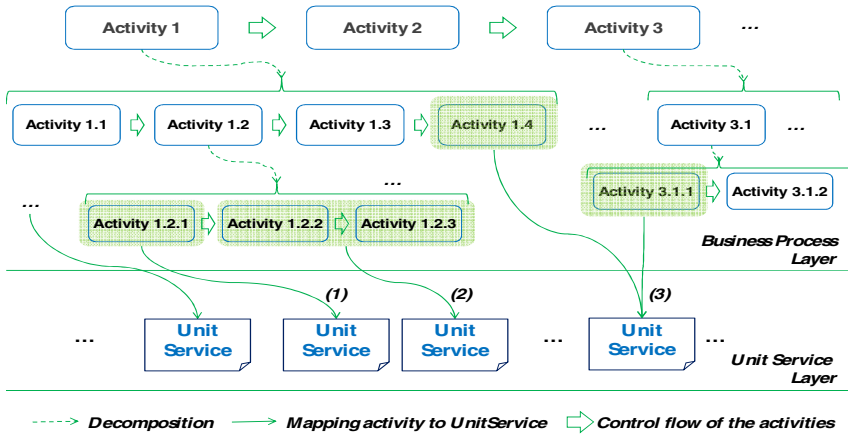


Fig. 2. Relationships between Activities and Unit Services

Service Interface Layer: In SOC, the interfaces of services are specified separately from service components, and service providers publish the services in WSDL in UDDI service registries. Hence, the unit services identified should be bound to *interfaces* of the published services which fulfill the requirement of the unit services. Therefore, the *Service Interface Layer* contains the interfaces of services published by service provides, and it separates the unit services from the service components. By having this layer, unit services can be bound to any compatible interfaces, and the interfaces can be realized by and bound to any compatible service components.

Service Component Layer: This layer is to specify service components which implement the service interfaces. Some components are like the one in CBD, and typically implemented with objects on OO/CBD platforms such as EJB [12]. Other components can be simply wrappers of legacy applications. There is a difference between the two types on how components are implemented, but they both have to provide physical interfaces that conform to the published interfaces (in WSDL) of the *Service Interface Layer*. For example, we may implement service components in EJB and provide physical interfaces in the forms of *EJB Home* and *Remote* interfaces, which conform to the WSDL service interfaces.

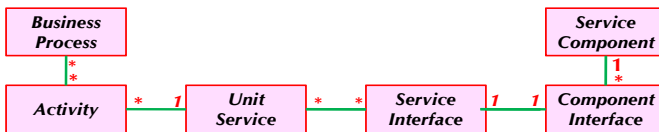


Fig. 3. Key artifacts of SOAD in the four layers

We summarize the relationships among the key artifacts of the four layers in Figure 3. The workflow of a business process is executed by one or more unit services, and unit services are bound to service interfaces. And, the service interfaces are bound to component interfaces of service components.

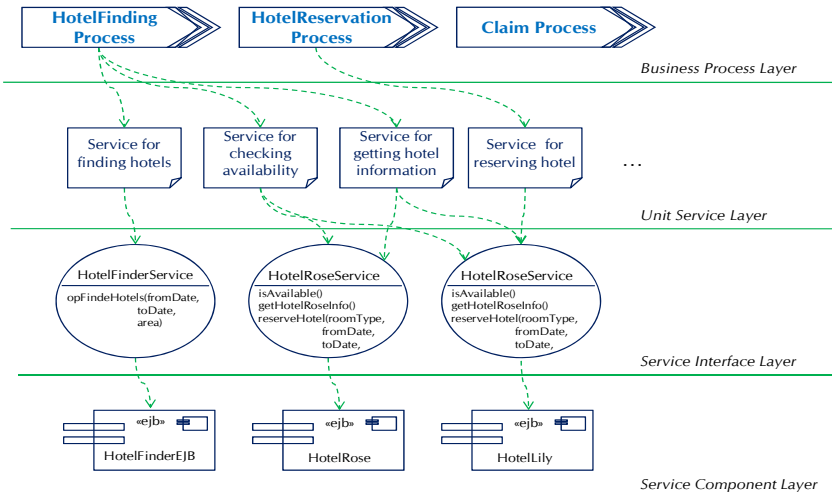


Fig. 4. Four Layers for Hotel Search and Reservation Services

We show an example of instantiating the four layers for a domain of hotel search and reservation service, as in Figure 4.

5 Process and Instructions

In this section, we present a SOAD process which consists of five phases; *Identifying Business Processes*, *Defining Unit Services*, *Discovering Services*, *Developing Services*, and *Composing Service*. The Figure 5 shows the phases with associated deliverables.

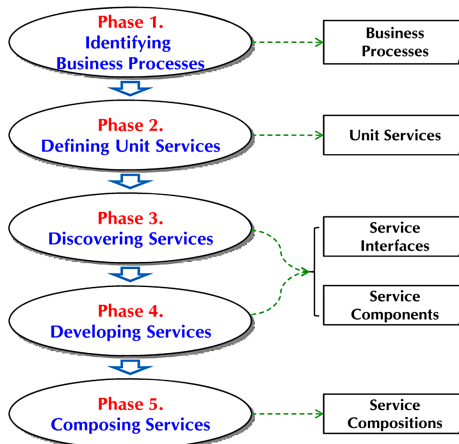


Fig. 5. Overall Process with Key Artifacts

5.1 Phase 1. Identifying Business Processes

This phase is to analyze the set of available service requirements, and to identify the set of business processes which are expected by service clients. It is carried out in three steps.

The first step is to acquire a set of service requirements. Since services in SoC are not just for a dedicated client but for potentially a family of clients, we need to acquire a comprehensive set of available service requirements. The service requirement is analogous to the conventional software requirement specification, but the requirement is formulated for the notion of *service* rather than software functionality. In addition to the available service requirements, it may be useful to predict and foresee the future potential services.

The second step is to compare the services from different requirements, and to derive a set of *target services* which will be eventually implemented. This can be done by conventional commonality and variability analysis techniques of CBD [18] or feature analysis of product line engineering [19].

The third step is to define a business process (i.e. workflow) for each of the target services. A business process is a flow of business logic and rules for the target service. Hence, it is best described as a workflow of one or more *activities* where an *activity* is a smaller grained business task within a workflow. Workflows can be described in any business process specification language or notation such as BPM.

5.2 Phase 2. Defining Unit Services

This phase is to take the business process specifications from phase 1, and to design and specify reusable unit services. The first step is to analyze the workflows and their activities for the business processes and to identify the commonality. Web services generally have the characteristics of loose coupling, composability, interoperability, reusability, extensibility, vendor diversity, and discoverability [1][9]. Hence the unit services that will be derived in phase 2 should have the characteristics of *loose coupling* and *reusability*, since the other characteristics are provided by WSDL interfaces, component implementations, and UDDI registries.

Extracting unit services would not be carried out mechanically, rather it requires the understanding of the domain, skill of commonality analysis, and intuition. But we formulate and suggest some guidelines for conducting this task, although these guidelines are not meant to be complete but to provide some clues.

- *Considering the data sets manipulated*; Functions specified in activities are to manipulate data and to provide its responsibility. Therefore, consider the data sets manipulated by activities in determining the cohesion between the activities. If two activities are highly cohesive, they can be satisfied by and bound to a single *UnitService*.
- *Considering Service Clients/Invokers*; Clients initiate activities in a business process. If the clients are different, two activities can be invoked at different time. Then, it would be better to design different unit services so that the unit services can be effectively invoked in different time lines.
- *Considering player performing the activity*; The player performing the activity is mapped to the organization of the corresponding service provider. Then, it

would be effective for *UnitService* to be differently defined if it is expected that providers are different.

- *Considering reusability of the activity;* Some activities are repeated in several business processes or higher level of activities. The activities have a possibility of reuse even in the business processes. If *UnitService* for the activities is defined, reusability for the business processes would be leveraged.

Once we identify the unit services, as the second step, we define an interface of each *UnitService*. The interface includes one or more operations which are represented with a set of an operation name, input parameters, and a return type. In this step, what should be done with a *UnitService* is syntactically defined in terms of the operation name, the input parameters, and the return type.

Then pre- and post conditions of defined interface should be defined. The states before and after invoking the *UnitService* are identified. The *UnitService* is expected to be implemented as components and to be configured flexibly. For the characteristics of cohesion and loose coupling, identifying pre and post conditions to which service components should conform is essential.

With the interface and pre and post conditions, we should design behavioral and structural models for the functionality of the *UnitService*. In OOAD, UML is generally used for the models and it can be applied to the models in this steps. During the modeling step, any associated constraints should be identified. The last step is to write unit service specifications with their interfaces, behavioral and structural models, and the constraints.

5.3 Phase 3. Discovering Services

Unit services are bound to service interfaces which can be either the interfaces of externally developed components or interfaces of internally developed components. This phase is to discover service interfaces which conform to the unit service specification. It can be done by searching candidate service interfaces from service registries, comparing the found interfaces with the specification of *UnitService*, and acquiring appropriate service interfaces.

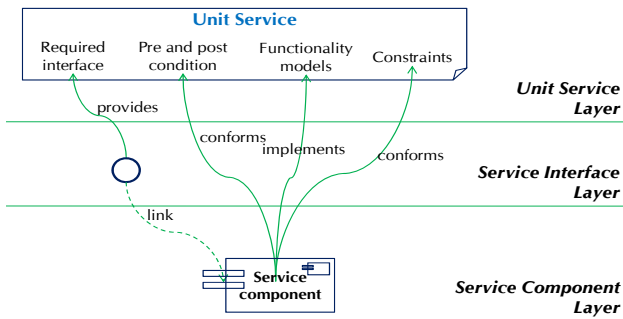


Fig. 6. Relationships between *UnitService*, Service Interface and Component

To find candidate service interfaces from UDDI registries, keyword-based matching technique is commonly used. However, it would be better to apply semantic-based matching using the semantic description of unit service specification.

As explained in Section 4.2, the elements of *UnitService* include an interface, pre and post conditions, structural and behavioral models, and constraints. The operations in the required interface should be mapped to the operations in the service interface. And, the service components for the service interface should implement the functionality models, and it should also conform to the pre and post conditions and other constraints. Figure 6 shows the relationships between the elements of a *UnitService* and a service interface and components.

5.4 Phase 4. Developing Services

After conducting phase 3, there may be some unit services which are not yet bound to the published service interfaces. For such unit services, one should develop service components from scratch or by wrapping legacy systems. This phase is to develop such new services and their service interfaces.

Initially, we try to identify available legacy systems for the functionality of the *UnitService*. If there are candidates, such system can be wrapped with service interfaces which specified with standard specification such as WSDL. If there still remain unresolved unit services, new service components with their interfaces are newly developed by using the unit service specifications. Finally, both types of new services and interfaces are registered in UDDI registries.

5.5 Phase 5. Composing Services

By applying the phases 1 through 4, we now have a set of unit services and service interfaces. This phase is to define service compositions with unit services and service interfaces. Physical representation of service compositions can be done a business process specification language such as BPEL.

Service composition can be in two styles; choreography and orchestration [20]. By analyzing the characteristics of business processes, we determine the composition style and define the composition, say in BPEL. Then, it can be executed by process executing engine such as Active BPEL.

6 Case Study of Hotel Reservation

Based on the proposed process in the Section 5, we present a case study for *Hotel Reservation* service. This service is to explore candidate hotels based on user's preference and perform the following scenario as shown in the Figure 8. For BPEL execution as well as Web service development, we use WebSphere Process Server 6.0 and IBM WebSphere Integration Developer 6.0.2.

Phase 1. Identifying Business Processes. Based on the scenario, we identify three activities; *A1.Finding Appropriate Hotels*, *A2.Choosing a Hotel*, and *A3. Reserving*

the Hotel. Then, we decompose the A1 into five sub-activities; A1.1 Get user profile, A1.2 Find hotel finding services, A1.3, Check availability of hotel service, A1.4 Take hotel information, and A1.5 Return the hotel information as shown in the Figure 8.

- John enters a total hotel reservation service system (THRSS).
- John enters ID and PW.
Based on the ID, the THRSS analyzes the user's preference and history.
(John is a scholar who frequently attends international conference in software engineering
- John enters the period, area, room type, and the hotel grade.
- THRSS selects a hotel finding service from candidate hotel finding services, which is competitive to find hotels in the area John enters.
- The competitive hotel finding service returns a list of candidate hotels and their WSDL information for service interfaces by which THRSS can access to each hotel.
- THRSS checks the current availability of the individual hotel.
- THRSS takes individual information for currently available hotels by using the returned WSDLs.
- THRSS returns the information.
- John chooses one hotel from the returned information.
- THRSS reserves the hotel John chose.
- John may want to cancel the reservation because
-

Fig. 7. The part of the Hotel Reservation Scenario

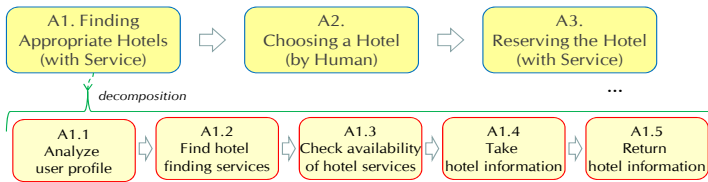


Fig. 8. The Decomposition of Activities in the Hotel Reservation Process

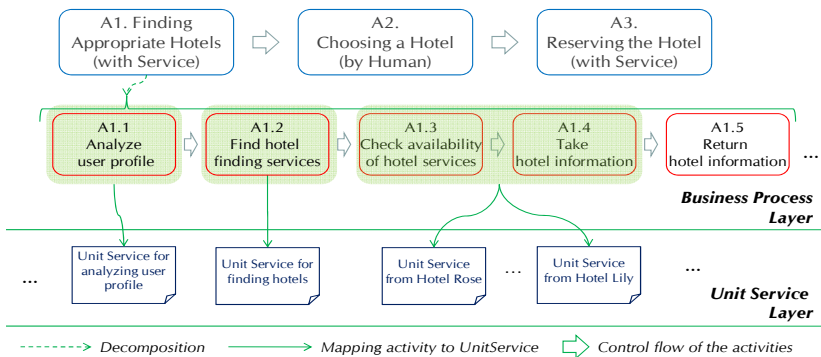


Fig. 9. Identifying unit services from activities in business process

The activity, A3, can be more decomposed. But, we omit the parts due to the page limitation.

I. Required Interface Specification

❖ Operation 1 - findCandidateHotels

- Set of input parameters
 - Check in date: Date
 - Check out date: Date
 - Room type: enumeration type of {single, double, twin, suite}
 - Level: enumeration type of {1,2,3,4,5}
- Return type

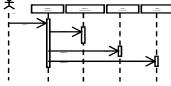
The operation requires a list of object type elements for the return type where the object includes:

 - The name of the hotel
 - Interface type of the hotel service including operation name, a set of input parameters, and return type.
 - The endpoint of the interface
- Pre condition
 - The use preference should be identified with pre-defined type.
- Post condition
 - N/A

II. Functionality Specification

❖ Operation 1 - findCandidateHotels

• Behavioral Model



- Finding logic
- ...

III. Constraints Specification

- Response time
 - It should not be over 3 sec.
- Reliability of the transaction
 - The found hotel should provide the service.
- Exception handling
 - ...

Fig. 10. A *UnitService* specification of the service, *finding candidate hotels*

```

<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://ejbs" .... >
<wsdl:types>
...
  <element name="opFindHotels">
    <complexType>
      <sequence>
        <element name="dFrom" nillable="true" type="xsd:string"/>
        <element name="dTo" nillable="true" type="xsd:string"/>
        <element name="sArea" nillable="true" type="xsd:string"/>
      </sequence>
    </complexType>
  </element>
...
<wsdl:portType name="HotelFinderA">
  <wsdl:operation name="opFindHotels">
    <wsdl:input message="impl:opFindHotelsRequest" name="opFindHotelsRequest"/>
    <wsdl:output message="impl:opFindHotelsResponse" name="opFindHotelsResponse"/>
  </wsdl:operation>
  ...
<wsdl:binding name="HotelFinderASoapBinding" type="impl:HotelFinderA">
  <wsdlsoap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="opFindHotels">
  ...
<wsdl:service name="HotelFinderAService">
  <wsdl:port binding="impl:HotelFinderASoapBinding" name="HotelFinderA">
  <wsdlsoap:address location="http://203.253.23.170:9082/routerProject/services/HotelFinderA"/>
  ...

```

Fig. 11. Service Interface of the *finding candidate hotels*, in WSDL

Phase 2. Defining Unit Services. Drawing from the decomposition, we identify four unit services; *Analyzing User Profile*, *Finding candidate hotels*, *checking availability of the each found hotel*, and *requesting information to the found hotels* as shown in the Figure 9. For the activities A1.1 and A1.2, the activities should be performed by

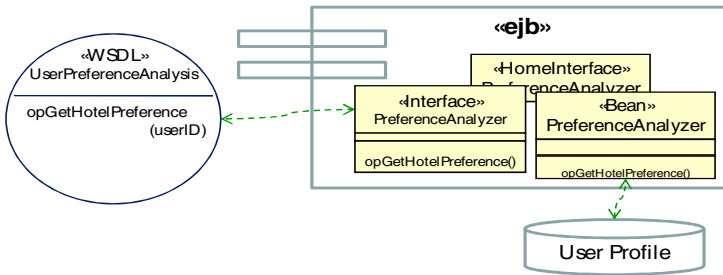


Fig. 12. Service Component and Interface for Analyzing User Preference Service

different player with different data. Therefore, they are defined with different unit services. For the activities A1.3 and A1.4, the activities can be carried out by each hotel service. Therefore, the two activities are defined with one *UnitService*. However, several candidate hotels can be identified from A1.2.

For the each *UnitService*, we write *UnitService* specifications. Figure 10 shows the example of *UnitService* specification for the unit service, *finding candidate hotels*.

In the specification, there is one operation, *findCandidateHotels*. The operation has three input parameters and returns an object including hotel name, its service interfaces, and the endpoint of the interfaces.

Phase 3. Discovering Services. For the defined unit services, Figure 11 shows the WSDL interface of the *UnitService*, *finding candidate hotels*. In the XML interface, *opFindHotels* is defined with three parameters and a return type.

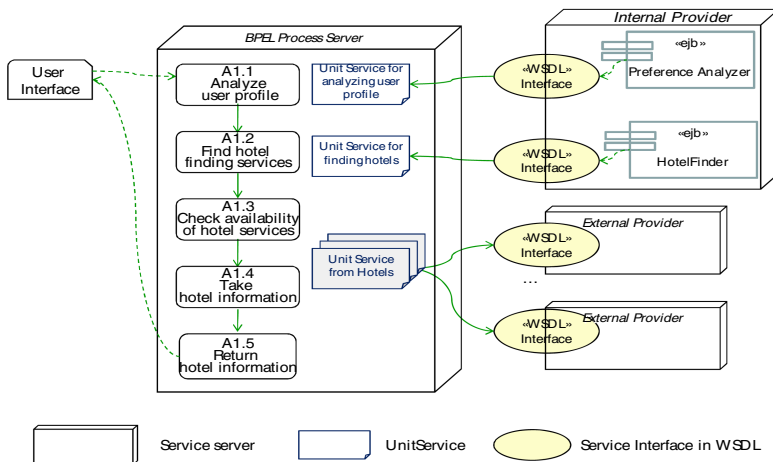


Fig. 13. The Service Integration Architecture

Phase 4. Developing Services. For *analyzing the user preference*, finding external services is not enough. It should be developed internally. So, the *UnitService* is implemented with EJB as shown in the Figure 12. The unit service has WSDL interface including a operation, *opGetHotelPreference*. The operation is bound to the component interface (i.e. EJB remote interface, *PreferenceAnalyzer*)

Phase 5. Composing Services. After discovering and developing the services, we integrate the services with BPEL. The Hotel reservation process includes two internal services and other external services. The *PreferenceAnalyzer* component is newly developed, and *HotelFinder* is discovered from internal UDDI registry. Other services, providing hotel information, are provided by each hotel. So, we don't need to know the service components but service interfaces.

7 Conclusion

SOC is emerging as a new paradigm for building and maintaining applications in a cost effectiveness way. However, there is a considerable amount of gap between the promises of SOC and the maturity of service engineering methodology. The main cause of the gap is the lack of effective analysis and design methods. That is, there is a great demand on effective service-oriented analysis and design (SOAD) methodology.

SOC has several unique features which are not typically presented in conventional OOD and CBD. They include the increased commonality of services and the paradigm of *publish-discover-compose* services. Hence, SOAD has to be defined with SOC-specific modeling facilities on top of the two existing paradigms.

In this paper, we proposed a systematic and effective SOAD which extends OOD and CBD. We first presented a technical comparison among OOD, CBD and SOAD to better understand their similarities and differences. From the comparison, we derived the design criteria for our SOAD method. And, we defined the key artifacts that have to be delivered during SOAD. Based on these preliminary works, we proposed a SOAD process which takes service requirements as the initial input and delivers service specifications, compositions, and verified service components as the final deliverables. Each of the five activities in the process was defined with its overview, artifacts, and work instructions. Finally, we presented a case study of applying our process to a service domain to reveal its effectiveness and applicability. As a future work, the process can be extended with service adaptation techniques. That is, analyzing and designing service variability and its adaptation can be appended. Once the proposed SOAD is well applied, SOAD artifacts can be more systematically and efficiently implemented with current SOA standards of WSDL, UDDI, BPEL and SOAP.

References

- [1] Sigh, M., Huhns, M.: *Service-Oriented Computing: Semantics, Processes, Agents*. Wiley, Chichester (2005)
- [2] W3C, *Web Services Description Language (WSDL) Version 2.0, W3C Candidate Recommendation* (March 27, 2006)

- [3] OASIS, Universal Description Discovery & Integration (UDDI) specification, Version 3.0 (July 2002), <http://uddi.org/pubs/uddi-v3.00-published-20020719.htm>
- [4] OASIS, Web Services Business Process Execution Language (BPEL) Version 2.0, Public Review Draft, (August 23rd, 2006)
- [5] W3C, Simple Object Access Protocol (SOAP) Version 1.2, W3C Recommendation (June 24, 2003)
- [6] Zimmermann, O., et al.: Elements of Service-Oriented Analysis and Design, IBM Developer Works (2004)
- [7] Arsanjani, A.: Service-oriented Modeling and Architecture, IBM Developer Works (2004)
- [8] Erl, T.: Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall, Englewood Cliffs (2005)
- [9] Marks, E., Bel, M.: Service Oriented Architecture: A Planning and Implementation Guide for Business and Technology. Wiley, Chichester (2006)
- [10] OASIS, A Methodology for Service Architectures, OASIS Draft, (October 26th, 2005)
- [11] Blaha, M., Rumbaugh, J.: Object-Oriented Modeling and Design with UML, 2nd edn. Prentice Hall, Englewood Cliffs (2005)
- [12] Heineman, G.T., Councill, W.T.: Component-Based Software Engineering. Addison-Wesley, London (2001)
- [13] Szyperski, C.: Component Software Beyond Object-Oriented Programming. Addison-Wesley, London (2002)
- [14] Min, H., et al.: Using Smart Connectors to Resolve Partial Matching Problems in COTS Component Acquisition. In: Crnković, I., Stafford, J.A., Schmidt, H.W., Wallnau, K. (eds.) CBSE 2004. LNCS, vol. 3054, Springer, Heidelberg (2004)
- [15] Sam, Y., et al.: Web Services Customization: A Composition-based Approach. In: the proceedings of the International Conference on Web Engineering (ICWE) '06, IEEE, Orlando (2006)
- [16] Kongdenfha, W., et al.: An Aspect-Oriented Framework for Service Adaptation. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, Springer, Heidelberg (2006)
- [17] Arsanjani, A.: Service-Oriented Modeling and Architecture (SOMA), IBM Developerworks (2004)
- [18] Kim, S., et al.: A Theoretical Foundation of Variability in Component-Based Development, Information and Software Technology (IST), vol. 47(10) (July 2005)
- [19] Pohl, K., Bockel, G., Linden, F.: Software Product Line Engineering. Springer, Heidelberg (2005)
- [20] Peltz, C.: Web Services Orchestration and Choreography. In: Computer (October 2003), vol. 36(10), pp. 46–52. IEEE, NJ (2004)

Improving the Problem Management Process from Knowledge Management Perspective

Marko Jäntti, Aki Miettinen, Niko Pykkänen, and Tommi Kainulainen

University of Kuopio, Department of Computer Science,
P.O Box 1627, 70211, Kuopio, Finland
mjantti@cs.uku.fi

Abstract. IT organizations are continuously looking for systematic methods to manage IT services. Combining IT service management processes and knowledge management processes is an interesting topic because knowledge management will be included in the next release of IT Infrastructure Library (ITIL). In this paper, we focus on examining how knowledge management can be used to support the improvement of the ITIL-based problem management process. The research question in this paper is: how to improve the software problem management process by using a knowledge management framework. We use a case study research method to examine how an IT service provider identifies, creates, stores, shares and uses the knowledge of software problems. The main contribution of this paper is to provide a list of process improvement ideas collected during a knowledge management study in the case organization: TietoEnator Energy, Finland.

1 Introduction

The software problem management process is focused on collecting information on problems in IT products and services, identifying defects related to problems, removing defects and preventing problems and defects before they occur. Problem management is one of the key subprocesses within a service support and maintenance process [1]. In ISO 20000 Service Management standard [2], that is fully aligned with IT Infrastructure Library (ITIL) [3], problem management belongs to resolution processes that also include incident management and change management [4]. In CoBIT framework [5], problem management is categorized into Delivery and Support processes (DS10). Problem management is responsible for both resolving already reported problems but also preventing problems before they occur.

Problem management can be understood as “defect management performed by the service desk”. In the ITIL, problem management is divided into problem control and error control activities. Problem control aims to identify the root cause of the problem and define a temporary or permanent solution to the problem. In this study, we define a problem as “any difficulty that a user or a customer encounters while using the software product or an IT service”. If the root cause of a problem is a software fault (a defect), the problem will be escalated

to the error control activity that is similar to the defect management process [6], [7], [8]. Hence, traditional defect management methods such as defect causal analysis [9], defect profiles [10], and defect estimation models [11] can be used to support the problem management activities. Additionally, traditional defect management metrics (defect density, defect removal rate) [12] are still useful for problem managers.

Knowledge Management in turn is focused on generating, representing, storing, transferring, transforming, applying, embedding and protecting organizational knowledge [13]. There are various frameworks and approaches available for organizations who are planning to implement a knowledge management process: Knowledge management framework of CEN (European Committee for Standardization) [14], DISER (Design and implementation of Software Engineering Repositories) [15], CRISP-DM (Cross Industry Standard Process for Data Mining) [16], and Knowledge Value Chain model [17]. In this paper we focus on CEN framework that consists of five core knowledge activities: identify, create, store, share and use knowledge.

1.1 Our Contribution

The purpose of the knowledge management is to create and apply the knowledge about products and services, customers and technology. In this study, we focus our research on exploring how the knowledge about software problems is identified, created, stored, shared and used in our case organization. Previous studies have discussed problem management from the viewpoint of software maintenance [18], [19].

This study is a part of the work of an ongoing research project SOSE (Service Oriented Software Engineering) at the University of Kuopio, Finland. SOSE project aims to research methods for improving the quality of IT services. This study continues the work of our previous studies where we identified difficulties regarding defect management and presented a list of challenges in the problem management processes.

The main contribution of this paper is to provide IT organizations with information how knowledge management models and IT service management processes can be combined to improve the quality of the service support processes. The research question in this paper is: how to improve the software problem management process by using a knowledge management framework. The Lessons Learned list with process improvement suggestions is based on the case study with one case organization: a large IT service provider.

The rest of the paper is organized as follows. In Section 2 we describe the research method of this study. In Section 3, main findings from the case study are presented. Section 4 is the analysis of findings. The discussion and the conclusions are given in Section 5.

2 Research Methods

This paper focuses on examining software problem management from knowledge management perspective. The research question in this paper is: how to improve

the software problem management process by using a knowledge management framework. We examine how IT organizations identify, create, store, share and use the knowledge of software problems.

A case study research method was used in this study [20] to investigate the problem management process within its real-life context. At first, we selected an appropriate knowledge management framework for the study. The CEN model was selected because the basic structure of the model (with 5 phases) is simple and it is published by a well-known standardization organization. The CEN model also addresses that customers, suppliers, and business partners play an important role in the knowledge management process (see Fig. 1).

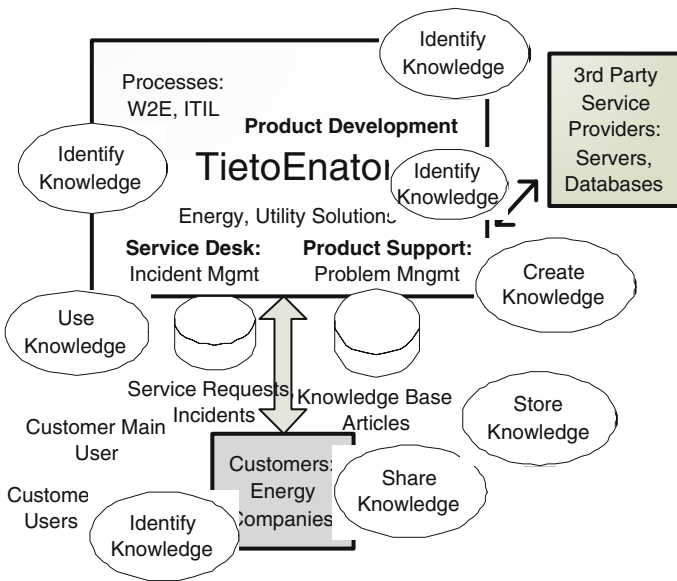


Fig. 1. The context of the case study

2.1 Case Organization

The case organization of this study (TietoEnator Oyj) is a large IT service company in Scandinavia with over 15 000 employees. TietoEnator provides application services and information systems to customers in various domains, such as energy, telecom and media, and healthcare. The business unit TietoEnator Energy develops and maintains customer information systems and energy data management systems. This case was selected for this study because they were interested both using ITIL framework and improving the knowledge management process. We used a case study research method because it is a good research method for studying information systems in organizations.

2.2 Data Collection Methods

The study started in February 2006 as a pilot project with the case organization. The goal of the pilot project was to improve the service support process including incident management and problem management. The following data collection methods were used in this study:

- The observations made by a researcher and a research assistant (February-March 2006) in the case organization
- Informal interviews and discussions in monthly SOSE research project meetings (researchers and three companies including the case organization)
- A research assistant's observations in the case organization (March-December 2006)
- A service improvement meeting between the case organization and its customer (an energy company) (December 21, 2 hours)
- A knowledge base meeting (January 4, 2 hours)

Data collection methods in February - March 2006 were participative observation (a researcher and a research assistant) in support and maintenance team meetings, informal interviews with the case organizations's workers including a service desk engineer, an incident manager, a problem manager, and a system analyst. From April 2006, only a research assistant has participated in the support team meeting of the case organization and had an access to the incident/problem repository. The members of the service improvement meeting included one researcher, two research assistants, two incident managers, a customer main user and a service desk engineer. The knowledge base meeting included one researcher, two research assistants, two incident managers, a change manager and a product development team member. This case study consisted of the following questions:

- General questions
 - Why is the organization interested in knowledge management?
 - Which tools are used for knowledge management purposes?
 - Has the organization already defined any knowledge management procedures?
- Identify knowledge: Who are the teams or persons responsible for identifying knowledge for problem management?
- Create knowledge: What kind of knowledge is created for problem management?
- Store knowledge: How does the organization store the problem management knowledge?
- Share knowledge: Which methods are used to share knowledge about problems?
- Use knowledge: How the problem management process uses the gathered knowledge?

2.3 Data Analysis Method

As a background work for the data analysis, questions were categorized by knowledge management process areas: identify, create, store, share and use knowledge. A within-case analysis technique was used to analyze data from the case organization [21]. Data analysis was focused on 1) identifying how knowledge management activities are related to the case organization's problem management process, and 2) identifying improvements to the problem management process.

3 Using Knowledge Management to Support the Problem Management Process

The case organization was interested in knowledge management because they had just launched a knowledge base. The problem was that the knowledge base was poorly connected to the business process descriptions. Their goal was that defining knowledge management activities would help the introduction of the knowledge base in problem management and incident management processes. SOSE research project's task was to help the case organization to define a process how the knowledge base can be combined to the service support processes. Because the organization had not used any knowledge management models earlier, we started to analyze the knowledge management activities through the CEN knowledge management framework [14]. The CEN model consists of five activities: 1) Identify, 2) Create, 3) Store, 4) Share, and 5) Use knowledge. In this section, we examine how these activities are related to the case organization's problem management process.

3.1 Identify Knowledge

In the first phase (Identify knowledge), the organization should identify what knowledge is needed and why it is needed. All members of the organization should know the goal of knowledge management and their own role in the knowledge management process. In the case organization, the goal of knowledge management was to support proactive problem management with a knowledge base. We observed that knowledge about problem management comes from the following sources:

- Customers (main users, normal users)
- Service desk teams (service desk workers, incident manager)
- Product support teams (problem specialists, problem manager)
- Product development teams (developers, coders and testers)
- Company partners (3rd party service providers, research organizations)

3.2 Create Knowledge

In the second activity, the organization creates either new knowledge for products and services that might lead to innovations or new knowledge for processes

and procedures (for example, process improvements). The knowledge can also be classified into personal level or team level knowledge, or explicit and tacit knowledge. Explicit knowledge is recorded or formal knowledge and thus easy to share. Tacit knowledge in turn is not available as a text and it includes personal beliefs and experiences. Our case organization creates following knowledge for the problem management process:

- Customers create problem descriptions, queries and product requirements.
- Service desk teams create incidents (see Fig. 2), service requests, change requests and development ideas. Incident manager performs customer satisfaction surveys regarding service support activities.
- Product support teams create problem records, work-arounds (temporary solutions for problems) and descriptions of permanent solutions.
- Product development teams create product releases, release notes, and user documentation.
- 3rd party service providers create problem descriptions, and research organizations create process improvement ideas.

The screenshot shows a web-based incident record form. At the top, there are tabs for 'Identification', 'Details', 'Delivery', 'Attachments', 'Related cases', and 'Open cases'. The 'Identification' tab is active. The form contains the following fields and values:

- Case id: 90556
- *Status: Closed
- *Title: Print-button does not work
- Service desk: SD Other
- *Type: Problem
- *Assigned to: PS PowerGrid
- Priority: 2 - Medium
- *Customer: Customer Ltd
- *Found by: testuser testi
- *Product: Product x
- Component: (empty)
- Business prior.: Kosmeettinen virhe
- Project: (empty)
- Description: Error in printing module
- Instance: Customer Ltd
- Solution: If Print-button does not work, printing is possible by selecting File - Print command. New Printing module (version 2.3) will be available 24th February on Customer Support sites

Fig. 2. The service desk creates incident records

3.3 Store Knowledge

Third activity (Store knowledge) can be implemented, for example, by using knowledge bases and document management systems. The primary goal should

be holding the knowledge, not the people within the organization. Besides formal database records, the knowledge can appear as organizational routines, process diagrams or checklists [22]. In the case organization, the service support tool (including incident and problem records) plays an crucial role regarding this activity. The case organization used following methods to store knowledge:

- A description of the problem management process has been stored in the organization's business framework WayToExcellence.
- Incidents, service requests, problems, RFCs, and development ideas are stored in the service support tool.
- General (not customer-specific) problem solutions are stored as knowledge base articles (see Fig. 3).
- Release notes are stored in the customer extranet.
- User documentation is usually stored as power point files, PDF and MS word files in windows directories.

The screenshot shows a knowledge base article interface with the following fields and values:

- Case id:** 90558
- *Product:** ProductX
- Version:** (empty dropdown)
- Component:** (empty dropdown)
- *Title:** Print command does not work
- Summary:** This article gives instructions how to proceed if print command does not work in Product X v. 3.0
- *Description:** Print command in invoice module does not work. Symptoms: Printing failure error message
- *Solution:** Download the bug fix (KB article 90675) on the customer support site.
- *KB article type:** Software Bug
- Platforms:** Windows 2000
- *Access:** Public
- *Status:** Open
- Entered by:** Järnti Marko
- *Date/time ent.:** 16.2.2006 10:40
- Last changed:** 8.3.2006 11:44

Fig. 3. A knowledge base article for problem management

3.4 Share Knowledge

The goal of the fourth activity (Share knowledge) is to share knowledge to the right place at the right time taking account in quality requirements for

the knowledge. Transferring knowledge is most effective through interaction between people, for example, in coffee break discussions, team meetings, and workshops. Sharing the knowledge can be done using various information channels: databases, intranet, internet, seminars, and training. However, it might happen that workers do not accept knowledge provided by their colleagues because of their own beliefs. The following techniques were used by the case organization to share knowledge:

- Customers can browse their own service requests, frequently asked questions and knowledge base articles (see Fig. 4) in the web-based support system.
- Customers have access to the customer extranet that contains marketing material, release notes and user manuals.
- Service desk teams, product support teams and product development teams have access to all incident and problem records.
- Product development teams share information on errors through the service support tool.

Asiakastuki - Customer Support

The screenshot shows a web interface for a customer support system. On the left is a vertical sidebar with navigation links: Report case, Follow up cases, Reports, FAQ, Knowledge base, User account, Logout, Home, and Powered by Requeste. The main content area is titled 'Detailed KB information' and contains the following details:

- Navigation: Back to list, Previous, Next
- KB ID: 122142
- Product: ITIL definitions
- Title: Incident definition
- Description: Any event which is not part of the standard operation of a service and which causes, or may cause, an interruption to, or a reduction in, the quality of that service. An incident does not necessarily lead to a problem or a defect. They can also be service requests (for example, a user needs instructions or advice).
- Solution: -
- Last changed: 6.11.2006 15:09
- Date/time ent.: 24.10.2006 9:00

Below the details is a rating section: 'How useful was this article for you:' with radio buttons for 'Not useful', 'Useful', and 'Very useful', and a 'Submit rating' button. At the bottom of the main content area is a 'Back to list' link.

Fig. 4. A shared knowledge base article

3.5 Use Knowledge

The benefits of knowledge management are not achieved until the knowledge is used in the organization. The final activity (Use knowledge) connects the previous activities together. Using the knowledge from previous activities helps organization to identify weak areas in the process and find new ways to create

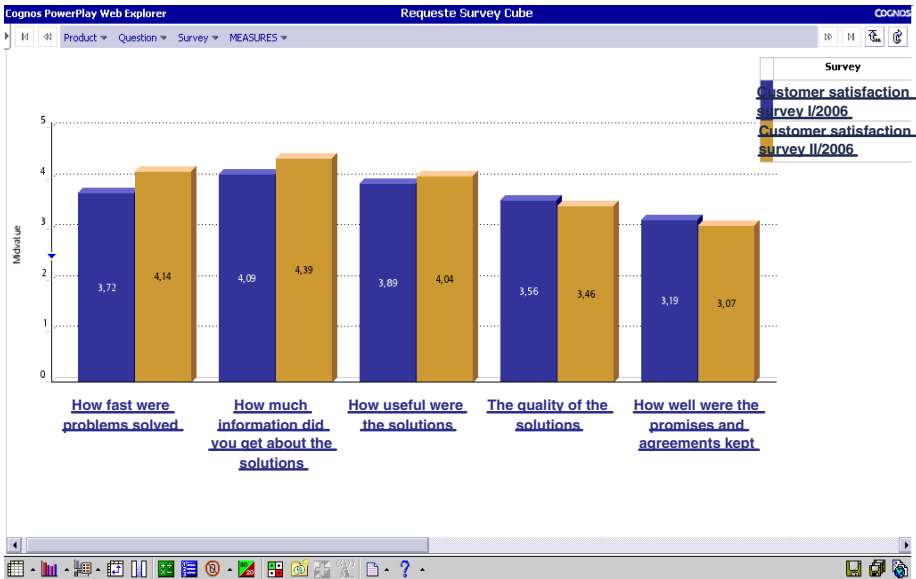


Fig. 5. Customer satisfaction data helps to improve the problem management process. Data values are not from a real case.

and use knowledge. Therefore, this activity serves as a bridge between identifying, creating, storing and sharing of knowledge. A good example how the case organization uses knowledge, are frequent customer satisfaction surveys that are conducted to improve the quality of support services including the problem management process (see Fig. 5).

4 Analysis

The following list of “Lessons Learned” describes the process improvements identified during the case study.

1. Identify knowledge - Customer: A customer-level should be divided into customer main user and customer user levels in process descriptions. These stakeholders use different ways to find problem resolutions and to report problems.
2. Identify knowledge - Service desk: Service desk workers do not have enough time to produce knowledge base articles during a working day but they can provide ideas for KB articles.
3. Identify knowledge - Service desk: Service desk workers have difficulties to find latest versions of user manuals and instructions from the Windows directories.

4. Create knowledge - Customer: Customers should be able to create free form feedback of the problem solutions described in knowledge base articles. In addition to a KB rating function (see Fig. 4), a feedback text field is needed.
5. Create knowledge - Problem management: Problem management teams should be responsible for creating workarounds for problems as knowledge base articles.
6. Store knowledge - Problem Management: Problem solutions should be written by using customers' language avoiding too technical terms.
7. Store knowledge - Problem Management: Persons, who store KB articles, should check that the attachments of the KB article are visible to customers (attachment access = external)
8. Share knowledge: Customers would like to have information about new knowledge base articles by email.
9. Share knowledge: There is need for a service board that reviews knowledge base articles in the case organization before they are published and is responsible for maintaining the knowledge base content.
10. Use knowledge: More persons need to be trained to use customer satisfaction survey tool for problem management purposes.
11. Use knowledge: A problem review process is needed. Knowledge from problem trend analyses (for example, statistics of component failures) is important for problem reviews.

4.1 The Summary of the Evaluation

Knowledge management systems consist of both information system activities and management and organizational activities [23]. Information system activities include identifying, creating, storing, sharing, and using knowledge. In the case organization, these activities had not been defined. Instead of establishing a formal knowledge management process, we suggested that knowledge management activities could be integrated with the organization's business processes.

Management and organizational activities include, for example, creating a knowledge culture in the organization, and defining practices for knowledge management [24]. The service support manager of the case organization considered the knowledge management ideas presented by researchers as important and useful for improving the problem management process. The management had started creating a knowledge culture where anybody within the organization could access the information about problems.

During this study, we identified that our case organization got the following benefits of the integration of knowledge management and problem management models: 1) clear rules how to maintain (add, modify, delete) the content of problem management knowledge base, 2) identification of new knowledge sources for the problem management process and 3) more information about the communication gaps within the support processes.

Besides the CEN model, also other knowledge management models might be useful for improving the problem management process, such as a knowledge domain model for customer services [25]. The knowledge domain model consists of

three tiers. Tier one includes general data and information services that are usually provided by call-centres, tier two consists of advisory services. These services provide customers with information on software problems and their resolutions. Tier three consists of knowledge and expert services which also include meta-level knowledge of the other tiers. In further studies, it could be interesting to combine the structure of the knowledge domain model and the activities of the CEN model together and use the combined model to analyze service support processes.

5 Discussion and Conclusions

This study aimed to answer the research question how problem management process can be improved by using a knowledge management model. Major benefits of the integration of knowledge management and problem management processes in this study were 1) clearer rules for maintaining the problem management knowledge base, 2) identification of problem management knowledge sources and 3) information about the communication gaps within the organizations processes.

However, the analysis of integrating problem management and knowledge management processes is not exhaustive. More research efforts are needed to explore the sources of problem management knowledge and the communication between stakeholders involved in knowledge management activities. As a result of the analysis, several interesting challenges regarding problem management and knowledge management were identified. The results of this study are useful for process managers who are responsible for improving support processes such as problem management. We observed that using the combination of problem management and knowledge management methods is relatively easy within one organization. However, the larger the network of stakeholders is, the more difficult it is to describe the combined process. This paper provides process managers with a list of process improvements identified during the study that focused on combining an ITIL-based problem management model and a knowledge management model.

However there are threats to the validity of this case study. First, regarding the construct validity of the case study research, we should be able to collect evidence from several sources. This study described experiences on one knowledge management model with one case organization that uses an ITIL-based problem management model. In order to get a richer view of integration of problem management methods and knowledge management methods, we need to test different types of models in practice. Additionally, we need to interview, besides service desk and product support teams, also product development teams that were ignored in this study. Second, there is the threat to external validity. The results presented in this paper are valid only in our case organization: TietoEnator, Energy. Results may not be generalizable to other service provider organizations because they use different service management tools and processes.

The main contribution of this study lies in helping IT organizations to identify how knowledge management concepts can be used to support the problem management process. In future studies we focus on examining the multi-actor network regarding problem management.

Acknowledgment

This paper is based on research in the SOSE project (2004-2006), funded by the National Technology Agency TEKES, European Regional Development Fund (ERDF), TietoEnator Corp., Savon Voima Oyj, Softera Solutions Oy, DNA Finland Oy, and Navicore Oy.

References

1. ISO/IEC 12207: Information Technology Software Life-Cycle Processes. ISO/IEC Copyright Office (1995)
2. ISO/IEC 20000:2005: Information Technology - Service Management. ISO/IEC Copyright Office (2005)
3. Office of Government Commerce: ITIL Service Delivery. The Stationary Office, UK (2002)
4. Hochstein, A., Tamm, G., Brenner, W.: Service-oriented it management: Benefit, cost and success factors. In: Proceedings of the Thirteenth European Conference on Information Systems, Regensburg, Germany, University of Regensburg (2005)
5. COBIT 4.0: Control Objectives for Information and related Technology: COBIT 4.0. IT Governance Institute (2005)
6. Quality Assurance Institute: A software defect management process. Research Report number vol. 8 (1995)
7. Florac, W.: Software quality measurement a framework for counting problems and defects. Technical Report CMU/SEI-92-TR-22 (1992)
8. Mays, R.G., Jones, C.L., Holloway, G.J., Studinski, D.P.: Experiences with defect prevention. *IBM Syst. J.* 29(1), 4–32 (1990)
9. Leszak, M., Perry, D.E., Stoll, D.: A case study in root cause defect analysis. In: ICSE '00: Proceedings of the 22nd international conference on Software engineering, New York, NY, USA, pp. 428–437. ACM Press, New York (2000)
10. Hirmanpour, I., Schofield, J.: Defect management through the personal software process. *Crosstalk, The Journal of Defense Software Engineering* (2003)
11. Biffi, S.: Evaluating defect estimation models with major defects. *J. Syst. Softw.* 65(1), 13–29 (2003)
12. Jalote, P.: CMM in Practice, Processes for Executing Software Projects at Infosys. Addison-Wesley, London (2000)
13. Schultze, U., Leidner, D.: Studying knowledge management in information systems research: discourses and theoretical assumptions. *MIS Quarterly* 26(3), 213–242 (2002)
14. CEN Workshop Agreement CWA 14924-1: European Guide to Good Practice in Knowledge Management, Part 1. European Committee for Standardization (2004)
15. Bomarius, F., Feldmann, R.: Get your experience factory ready for the next decade: Ten years after how to build and run one. Profes 2006 Tutorial, Amsterdam, Netherlands (2006)
16. Chapman, P., Clinton, J., Kerber, R., Khabaza, T., Reinartz, T., Shearer, C., Wirth, R.: CRISP-DM 1.0: Step-by-step data mining guide. CRISP-DM consortium (2000)
17. Lee, C., Yang, J.: Knowledge value chain. *Journal of Management Development* 19(9), 783–794 (2000)

18. Kajko-Mattsson, M.: Problem management maturity within corrective maintenance. *Journal of Software Maintenance* 14(3), 197–227 (2002)
19. Kajko-Mattsson, M., Forssander, S., Olsson, U.: Corrective maintenance maturity model (cm3): maintainer's education and training. In: ICSE '01: Proceedings of the 23rd International Conference on Software Engineering, Washington, DC, USA, pp. 610–619. IEEE Computer Society, Los Alamitos (2001)
20. Yin, R.: *Case Study Research: Design and Methods*. Sage Publishing, Beverly Hills, CA (1994)
21. Eisenhardt, K.: Building theories from case study research. *Academy of Management Review* 14, 532–550 (1989)
22. Kokkonen, J.: Experiences from generating checklists. In: Boumedine, M., Touzet, C. (eds.) *Proceedings of the Fourth IASTED International Conference on Knowledge Sharing and Collaborative Engineering*, St. Thomas, US Virgin Islands, IASTED, ACTA Press, pp. 51–62 (2006)
23. Malhotra, Y.: Integrating knowledge management technologies in organizational business processes: getting real time enterprises to deliver real business performance. *Journal of Knowledge Management* 9(1), 7–28 (2005)
24. Iske, P., Boersma, W.: Connected brains: Question and answer systems for knowledge sharing: concepts, implementation and return on investment. *Journal of Knowledge Management* 9(1), 126–145 (2005)
25. Cheung, C., Lee, W., Wang, W., Chu, K., To, S.: A multi-perspective knowledge-based system for customer service management. *Expert Systems with Applications* 24(4), 457–470 (2003)

Experience on Applying Quantitative and Qualitative Empiricism to Software Engineering

Marcus Ciolkowski^{1,2} and Andreas Jedlitschka²

¹ TU Kaiserslautern

P.O. Box 3049

67653 Kaiserslautern, Germany

ciolkows@informatik.uni-kl.de

² Fraunhofer Institute for Experimental Software Engineering

Fraunhofer-Platz 1

67663 Kaiserslautern, Germany

jedl@iese.fraunhofer.de

Abstract. The workshop addresses practitioners and/or researchers who are interested in empirical software engineering, software process improvement, and quality management. Practitioners are being addressed specifically, since this workshop is also intended to find out what kind of information practitioners need, which kind of support they expect from research regarding the aggregation of information, and how they select software engineering technology.

1 Motivation

Many academics and practitioners believe that evaluation has a vital role to play in software engineering. As well as evaluating both application level and component level products, software engineers need to be concerned with the evaluation of development processes, engineering methods and supplier organizations.

The state of the art of empirical software engineering was assessed in the 2006 Dagstuhl Seminar on Empirical Software Engineering [1]. Workshop participants agreed that the community has matured since 1992 but is still in a very early phase compared to other disciplines. Improvements were suggested, among others, with regard to complementary usage of quantitative and qualitative studies [2].

The aim of the workshop is to address the question of future directions for empirical software engineering. This includes improving empirical methodology, or information needed by industry. It is planned that participants submit short submission statements, and that selected participants are invited to submit an extended version of their paper for submission.

The workshop itself is the fifth one in the workshop series on Empirical Software Engineering. The first one was held in conjunction with the PROFES 2002 in Rovaniemi, the second one was held in conjunction with the Empirical Software Engineering International Week 2003 in Rome, the third one was held in conjunction with PROFES 2005 in Oulu, the fourth one took place in Amsterdam in conjunction with PROFES 2006.

2 Topics of Interest

- Innovative approaches for empirical software engineering methods. Example questions addressed:
 - What are your experiences from applying quantitative and qualitative-methods (preferably a combination of different methods)?
 - How can contributions from other fields, such as medicine and psychology, help in Empirical Software Engineering?
 - What are your experiences with advanced methods that are not yet standard in empirical software engineering?
 - Which approaches for analyzing / summarizing sets of empirical studies can be worthwhile for Empirical Software Engineering?
- Clearly Information needed for decision making from empirical studies. Example questions addressed:
 - Have you tried to introduce a technology and wanted to base your decision on empirical findings?
 - Which types of empirical studies were helpful?
 - Which were not? For which reasons (e.g., wrong type of study, orunhelpfully reported)?
 - Under which conditions are studies helpful to practitioners?
 - How does empirical research have to adapt to industrial needs?
- Weaknesses in current empirical methodology. Example questions addressed:
 - Where do we need to improve existing empirical practice, and how?
- Conditions under which can we integrate/synthesize evidence. Example questions addressed:
 - Which kind of evidence can be integrated?
 - Which information is of interest?
- Handling of context information. Example questions addressed:
 - How can we specify for which context areas an analysis is valid?
 - Can we define a “standard” set of metrics that are relevant to measure context in every situation?

3 Workshop Chairs

Marcus Ciolkowski (University Kaiserslautern and Fraunhofer IESE)
 Andreas Jedlitschka (Fraunhofer IESE)

4 Program Comittee

Reidar Conradi, NTNU	Helen Sharp, Open University
Tore Dyba, SINTEF	Silke Steinbach-Nordmann, Fh IESE
Tracy Hall, University of Hertfordshire	Mikael Svahnberg, BTH
Natalia Juristo, UPM	Guilherme Travassos, COPPE/UFR
Dietmar Pfahl, University of Calgary	Sira Vegas. UPM
Per Runeson, Lund Univ.	Alf Inge Wang, NTNU

References

1. Basili, V.R., Rombach, D., Schneider, K., Kitchenham, B., Pfahl, D., Selby, R.W. (eds.): Empirical Software Engineering Issues: Critical Assessment and Future Directions, International Workshop Dagstuhl Castle, Germany, January 2007, 4336. Springer, Heidelberg (to appear)
2. Basili, V.R., Rombach, D.: Schneider: Preface; in [1] pp. V-XI technology

Using Metrics to Improve Software Testing

Alfred Sorkowitz

International Institute for Software Testing,
636 Mendelssohn Avenue North,
Golden Valley, MN 55427, USA
sork1@hotmail.com

Abstract. Software Metrics can aid in improving your organizations Testing Process by (1) providing insight and early visibility into the "real" status of the testing effort, and (2) aid in making assessments as to whether progress, productivity and quality goals are being met. This tutorial presents a practical guide on how to start taking advantage of these new tools/techniques to aid in improving the testing process. These metric based tools and techniques have successfully been used by (1) software test teams, (2) software developers and, (3) SQA and IV&V staffs.

1 What You Will Learn

- The cost of inadequate software testing. The economic impacts of poor testing from a recent report from the National Institute of Standards and Technology.
- A set of "best practices" software metrics with numerous examples, variations, and case studies. These metrics can track the "real status", quality, and productivity of the testing effort, as well as provide an indication of future problems.

2 Audience

A practical overview of metrics-based testing designed for technical and managerial professionals concerned with improving quality, performance, and productivity of software testing.

3 Presenters' Background

Mr. Alfred Sorkowitz was a Computer Scientist with the Department of the Navy, and was responsible for developing real-time, software-intensive systems. Prior to joining the Dept of the Navy, he was Director of the Standards and Quality Control Staff, United States Department of Housing and Urban Development. The staff was responsible for Software Standards and SQA, for all in-house as well as contractor developed software. While at HUD, he initiated a successful testing procedure to improve the quality of testing that utilizes automated tools and software metrics. A paper on this

effort was published in a special issue of the IEEE Computer Society magazine devoted to Software Quality Assurance, and was later reprinted and widely distributed in the Department of Defense Computer Institute "Selected Computer Articles".

Mr. Sorkowitz has published papers and has presented seminars on Software Metrics, SQA, and Testing at conferences sponsored by the IEEE Computer Society, ACM, and the British Computer Society.

Increase ICT Project Success with Concrete Scope Management

Carol Dekkers¹ and Pekka Forselius²

¹ Quality Plus Technologies, Inc.
8430 Egret Lane
SEMINOLE FL USA 33776

dekkers@qualityplustech.com

² Software Technology Transfer Finland STTF Ltd
Tekniikantie 14, 2nd floor
FIN- 02150 Espoo, Finland
pekkaf@sttf.fi

Abstract. With the Standish group's CHAOS report proclaiming ICT project success on a mere one-third of projects, project managers have an obligation worldwide to gain control of the situation. Through concrete scope management processes, ICT project managers can learn and embrace proven approaches that measure the size of software projects, streamline the requirements articulation and management, and impose solid change management controls, to keep projects on time and on budget. Scope management is not rocket science, however, with 2/3 of the world's ICT projects deemed as failures, it is apparent that managing scope is not a natural byproduct of project management. Learn approaches and tips used in Europe, Australia, and North America that have dramatically increased the success on ICT projects by trained scope managers.

1 Outline

1. Introduction to unique project management issues on ICT projects
2. Scope management as PMBOK knowledge area-opportunities and concepts for ICT projects
3. Scope manager role and responsibilities.
4. Scope management processes and areas of application.
5. How to apply solid scope management for success on ICT projects.

2 Learning Objectives

- Identify the unique challenges and opportunities on ICT (Information and communication technology)
- Clearly apply PMBOK scope management concepts to ICT projects through globally proven scope management

- Embrace the role and responsibilities for professional scope management for ICT projects

3 Presenters' Background

Carol Dekkers, PMP, CMC, P.Eng. A recent past president of the International Function Point Users Group (IFPUG) Board of Directors. Previously held various volunteer positions including 13 years of service to the IFPUG board and membership (5 years on the Board). Technical advisor to the International Software Benchmarking Standards

Group (ISBSG). Past Chair of PMI Metrics SIG, past member of PMI Leadership Institute LI'04 class, Member of the American Society for Quality (ASQ) Software Division council, and track chair for the annual Congress – Software Division track (Since 2002) Project editor to ISO on Functional Size Measurement (ISO/IEC JTC1 SC7 WG 12), and a current U.S. delegate to ISO SC7 (Software Engineering) since 1994.

Professional designations include: Certified Management Consultant (CMC), Certified Function Point Specialist (CFPS), Information Systems Professional (ISP), and Professional Engineer (P.Eng.-Canada)

Recognized expert in the software metrics field and author of many articles on function points, software metrics, and the human aspects of introducing change. Visiting scientist for measurement with the Software Engineering Institute (SEI) at Carnegie Mellon University. Management consultant, trainer and practitioner with international experience in software metrics, function points and measurement program start-up. Frequent presenter and trainer at metrics and quality conferences including IFPUG, Quality Assurance Institute (QAI), American Society for Quality Control (ASQC), Applications of Software Measurement (ASM), Canadian Information Processing Society (CIPS), Applied Computer Research (Client/Server conference).

Carol's system development background includes ten years of progressive experience in all phases of the systems development life cycle across a wide range of methodologies and development technologies. Her project management experience involved financial, judicial, MIS, engineering and scientific systems, in both private and public sectors. She is co-author of two books: Practical software measurement: Advice from the experts (IFPUG, Addison-Wesley, 2004), and Practical Project Estimation, 2nd Edition (ISBSG, 2005).

Pekka Forselius, MBA, MSc. Pekka Forselius is a Business partner, CEO and project management consultant at Software Technology Transfer Finland (STTF) Oy. Developed the Experience Pro data-collection concept and is the product manager of Experience Pro software. Researcher and developer of project management methods and concepts, including FiSMA Scope Management and KISS Functional Sizing. As research associate at INSEAD since 1996 and at University of Brunel since 2003, his research specialty is organisational learning, in particular corporate memory and benchmarking.

MSc in informatics and an executive MBA from the University of Jyväskylä. Since 2000, is the primary representative of the national body of Finland to ISO/IEC JTC1/SC7 standardisation working group WG12, Functional Size Measurement. Vice

President of the international benchmarking organisation (ISBSG), a member of the executive committee of the COSMIC consortium and past board member of the Finland Information Processing Association (FIPA).

Co-author of two books: *Tivi-projektien johtaminen* (ICT project management, TTL, 2005) and *Practical Project Estimation*, 2nd Edition (ISBSG, 2005) and also co-author of chapter 5 of *Applied Statistics for Software Managers* (Prentice Hall, 2002).

Agile Software Development: Theoretical and Practical Outlook

Pekka Abrahamsson¹ and Jari Still²

¹ VTT Technical Research Centre of Finland

P.O. Box 1000, FI-02044 VTT

Pekka.Abrahamsson@vtt.fi

² F-Secure Corporation

Tammasaarenkatu 7

PL 24, 00181 Helsinki, Finland

jari.still@f-secure.com

Abstract. Agile Software development has become to be one of the most prominent approaches in the field of software engineering. The amount of empirical evidence is quickly building up and it is known that already one out of seven software companies use agile processes. Many of the major corporations have announced to pursue for agile solutions with the aim of improving dramatically the lead-times, costs and quality aspects. Little is still known about the theoretical underpinnings of agile approaches. This tutorial serves for two purposes. First it demonstrates how agile solutions are theoretically founded and then in very pragmatic terms shows a longitudinal four-year case study how F-Secure, an antivirus company, from Finland transformed from iterative development to agile development framework. The case study also demonstrates which solutions worked and which ones proved to offer serious obstacles in the way. Finally, the business impact of the agile development is outlined based on empirical data. The tutorial will consist of interactive lectures, exercises and group work. The tutorial is targeted to software engineers and managers as well as academics.

1 Outline

1. Agile software development: Background, principles and approaches (Scrum, XP, Mobile-D™).
2. A theory for agile software development.
3. The four deployment strategies.
4. Case F-Secure: Four years of agile development
5. Business impact figures.

2 Learning Objectives

- Understand the rationale, background and theoretical basis for agile development
- Learn different ways of deploying agile methods and principles
- Gain concrete understanding of the possible business impact envisioned

3 Presenters' Background

Pekka Abrahamsson is research professor at VTT Technical Research Centre of Finland. Currently, he is on leave from the University of Tampere where he is a full professor in the field of Information Systems and Software Engineering. His current responsibilities include managing a FLEXI-ITEA2, which involves 38 organizations from 7 European countries. The project aims at developing agile innovations for global software development. His research interests are centred on mobile application development, business agility, agile software production and embedded systems. He leads the team who has designed an agile approach for mobile application development - the Mobile-D™. He has coached several agile software development projects in industry and authored 50+ scientific publications focusing on software process and quality improvement, agile software development and mobile software. His professional experience involves 5 years in industry as a software engineer and a quality manager.

Jari Still – Jari Still F-Secure Oy's Oulu Office site manager and the head of mobile R&D. Still has been working at F-Secure since 2000. Between the years from 1991 to 2000 Still was working as CEO of Modera Point Oy. Before 1991 Still had worked with e.g. Nokia Mobile Phones and Siemens. Still is one of the founders of the Oulu Software Forum, and currently acting as a Chairman of the Forum (www.swforum.net). Other posts have been e.g. Chairman of revontuliryhmä and member of the Board in several companies. Related to Agile process and product development Still is acting as a leader in AGILE-ITEA -project at F-Secure and he is also a member of the management group of AGILE-VTT -project.

Author Index

- Abrahamsson, Pekka 410
Abran, Alain 273
Al-Emran, Ahmed 315
Amengual, Esperança 108
Arnicane, Vineta 175
- Babar, Muhammad Ali 118, 330
Bae, Jeong Seop 358
Belkébir, Youssef 35
Bičevska, Zane 262
Bičevskis, Jānis 262
Bjørnson, Finn Olav 132
Bourque, Pierre 35
- Car, Željka 51
Catal, Cagatay 300
Chang, Soo Ho 374
Chen, Weibing 5
Cheun, Du Wan 358
Choi, Si Won 358
Ciolkowski, Marcus 402
Conradi, Reidar 5
Cuadrado-Gallego, Juan Jose 273
- Dekkers, Carol 4, 407
Díaz-Ley, María 247
Dingsøy, Torgeir 132
Diri, Banu 300
Doucet, Mikel 35
- Figueiredo, Sávio 81
Flohr, Thomas 147
Forselius, Pekka 407
Fukuoka, Tomoyuki 284
- Galinac, Tihana 51
García, Félix 247
- Harada, Yoko 284
Henderson-Sellers, Brian 222
Her, Jin Sun 358
- Jäntti, Marko 389
Jedlitschka, Andreas 402
Ji, Junzhong 5
- Kääriäinen, Jukka 188
Kainulainen, Tommi 389
Kim, Soo Dong 358, 374
Krikhaar, René 65
- Laporte, Claude Y. 35
Li, Jingyue 5
Liu, Chunnian 5
Lübke, Daniel 147
- Ma, Jianqiang 5
Machado, Fernando 273
Madeyski, Lech 207
Mas, Antònia 108
Mermans, Martin 65
Miettinen, Aki 389
Mishra, Alok 237
Mishra, Deepti 237
Moe, Nils Brede 20, 132
Montoni, Mariano 81
- Namiki, Rieko 284
Niazi, Mahmood 96, 118
Nonaka, Makoto 330
- Ocampo, Alexis 160
- Pfahl, Dietmar 315
Philipp, Michael 201
Piattini, Mario 247
Pylkkänen, Niko 389
- Qumer, Asif 222
- Rocha, Ana Regina 81
Rodríguez, Daniel 273
Rombach, H. Dieter 1
- Santos, Gleison 81
Schneider, Kurt 147
Schweigert, Tomas 201
Šmite, Darja 20
Sorkowitz, Alfred 405

Soto, Martin 160
Stålhane, Tor 132
Stapel, Kai 147
Staples, Mark 330
Still, Jari 3, 410
Urtāns, Guntis 2
Välimäki, Antti 188

Washizaki, Hironori 284
Watanabe, Hiroyuki 284
Wilson, David 96
Zeiris, Edzus 345
Zhu, Liming 330
Ziema, Maris 345
Zowghi, Didar 96