Tuomas Mäkilä

Software Development Process Modeling

Developers Perspective to Contemporary Modeling Techniques

Turku Centre *for* Computer Science

TUCS Dissertations
No 148, November 2012

It was a dark and stormy night.

Software Development Process Modeling

Developers Perspective to Contemporary Modeling Techniques

Tuomas Mäkilä

To be presented, with the permission of the Faculty of Mathematics and Natural Sciences of the University of Turku, for public criticism in Auditorium Lambda on November 2, 2012, at 12 noon.

University of Turku
Department of Information Technology &
Business and Innovation Development Unit
20014 Turku, Finland

2012

Supervisors

Docent Timo Knuutila
Business and Innovation Development Unit
University of Turku
FIN-20014 Turku
Finland

Professor Ville Leppänen
Department of Information Technology
University of Turku
FIN-20014 Turku
Finland

Reviewers

Professor Ivan Porres
Department of Information Technologies
Åbo Akademi University
Joukahainengatan 3-5, FIN-20520 Åbo
Finland

Professor Dmitry V. Koznov
Mathematics and Mechanics Faculty
St. Petersburg State University
198504, Universitetsky pr., 28, Stary Peterhof
Russia

Opponent

Professor Hannu Jaakkola
Tampere University of Technology, Pori
P.O.Box 300, FIN-28101 Pori
Finland

ISBN 978-952-12-2790-5
ISSN 1239-1883

Abstract

Formal software development processes and well-defined development methodologies are nowadays seen as the definite way to produce high-quality software within time-limits and budgets. The variety of such high-level methodologies is huge ranging from rigorous process frameworks like CMMI and RUP to more light-weight agile methodologies. The need for managing this variety and the fact that practically every software development organization has its own unique set of development processes and methods have created a profession of *software process engineers*. Different kinds of informal and formal software process modeling languages are essential tools for process engineers. These are used to define processes in a way which allows easy management of processes, for example process dissemination, process tailoring and process enactment.

The process modeling languages are usually used as a tool for process engineering where the main focus is on the processes themselves. This dissertation has a different emphasis. The dissertation analyses modern software development process modeling from *the software developers' point of view*. The goal of the dissertation is to investigate whether the software process modeling and the software process models aid software developers in their day-to-day work and what are the main mechanisms for this. The focus of the work is on *the Software Process Engineering Metamodel (SPEM)* framework which is currently one of the most influential process modeling notations in software engineering.

The research theme is elaborated through six scientific articles which represent the dissertation research done with process modeling during an approximately five year period. The research follows the classical engineering research discipline where the current situation is analyzed, a potentially better solution is developed and finally its implications are analyzed. The research applies a variety of different research techniques ranging from literature surveys to qualitative studies done amongst software practitioners.

The key finding of the dissertation is that software process modeling notations and techniques are usually developed in process engineering terms. As a consequence the connection between the process models and actual development work is loose. In addition, the modeling standards like SPEM are partially incomplete when it comes to pragmatic process modeling needs, like light-weight modeling and combining pre-defined process components. This leads to a situation, where

the full potential of process modeling techniques for aiding the daily development activities can not be achieved.

Despite these difficulties the dissertation shows that it is possible to use modeling standards like SPEM to aid software developers in their work. The dissertation presents a light-weight modeling technique, which software development teams can use to quickly analyze their work practices in a more objective manner. The dissertation also shows how process modeling can be used to more easily compare different software development situations and to analyze their differences in a systematic way. Models also help to share this knowledge with others.

A qualitative study done amongst Finnish software practitioners verifies the conclusions of other studies in the dissertation. Although processes and development methodologies are seen as an essential part of software development, the process modeling techniques are rarely used during the daily development work. However, the potential of these techniques intrigues the practitioners.

As a conclusion the dissertation shows that process modeling techniques, most commonly used as tools for process engineers, can also be used as tools for organizing the daily software development work. This work presents theoretical solutions for bringing the process modeling closer to the ground-level software development activities. These theories are proven feasible by presenting several case studies where the modeling techniques are used e.g. to find differences in the work methods of the members of a software team and to share the process knowledge to a wider audience.

Acknowledgements

This dissertation is published about three years after its latest article. At the end of the day, I'm the only *person* responsible for the delay. However, I have been working at the University of Turku since 2004 and there has been years when I have worked as a professional, full-time teacher and acted as a hobbyist, weekend doctoral student. Luckily, the things are getting better for the future doctoral candidates working at the University of Turku.

Despite the minor critique, the fact is that the University of Turku enabled me to fulfill my dreams and write this dissertation. In addition to my home university, Turku Centre for Computer Science supported my conference trips and the publication of the dissertation book. I am also grateful of a grant from the Nokia Foundation which helped me financially during the dissertation work. The last organization I wish to thank is Neoxen Systems whose research project with the University of Turku basically started my dissertation work and gave its initial direction.

There is only one name on the cover of this book but whole lot more behind it. Supervisors Ville Leppänen and Timo Knuutila have dragged me through the learning experience called the doctoral studies. Without their effort this book would not probably exists. I would also like to thank reviewers Ivan Porres and Dmitry V. Koznov for their invaluable comments and patience during the review process, and Hannu Jaakkola for kindly agreeing to act as an official opponent. I am also grateful for professor Olli Nevalainen of his feedback and comments.

This dissertation would not be possible without the research group and the co-authors interested in the same topic. I wish to express my special gratitude to Antero Järvi and Harri Hakonen, who mentored and guided me by example through the process that led to this dissertation. The other co-authors, Luka Milovanov, Henrik Terävä, Jouni Smed and Andy Best, gave their valuable contribution to the respective articles.

I also wish to mention Joanna Airiskallio, who checked the language of the dissertation, and Tomi "bgt" Mäntylä, who handled the practicalities concerning the printing and the publication of the book.

Besides the colleagues I wish to thank my family and friends who had to listen me lecturing about the dissertation topic all these years. My sister Annastiina and her cohabitant Toni Selkälä cheered me forward and gave tips on scientific think-

ing. My parents Sirpa and Tapio gave me food and shelter, when I was young, and my parents-in-law Leena and Ilkka Saarinen gave my wife and son food and shelter, when I needed peace to write the dissertation.

Susanna, Milo, and Nemo, you are important to me.

Turku, September 28th 2012,

Tuomas Mäkilä

List of original publications

1. Antero Järvi and Tuomas Mäkilä. *Observations on Modeling Software Processes with SPEM Process Components.* In Proceedings of The 9th Symposium on Programming Languages and Software Tools, pages 59–69. University of Tartu, 2005.

2. Tuomas Mäkilä and Antero Järvi. *Spemmet – A Tool for Modeling Software Processes with SPEM.* In Proceedings of the 9th International Conference on Information Systems Implementation and Modelling, ISIM 06, pages 87–94. MARQ, 2006.

3. Antero Järvi, Tuomas Mäkilä and Harri Hakonen. *Changing Role of SPI – Opportunities and Challenges of Process Modeling.* In Proceedings of the 13th European Conference on Software Process Improvement (EuroSPI), pages 135–146. Springer-Verlag, 2006.

4. Tuomas Mäkilä, Antero Järvi and Luka Milovanov. *Light-weight Approach for Software Process Modeling – A Case Study.* In Proceedings of New Exploratory Technologies 2007, pages 12–16. Korea Electronic Forum, 2007.

5. Tuomas Mäkilä, Harri Hakonen, Jouni Smed and Andy Best. *Three Approaches Towards Teaching Game Production.* M. Kankaanranta, P. Neittaanmäki (Eds.), Design and Use of Serious Games, Intelligent Systems, Control, and Automation: Science and Engineering, pages 3–18. Springer Netherlands, 2009.

6. Tuomas Mäkilä and Henrik Terävä. *Survey of Practitioners Attitudes to Software Process Modeling.* In Industrial Proceedings of the 16th European Conference on Software Process Improvement (EuroSPI), pages 12.25–12.33. Delta, 2009.

Contents

Chapter 1

Introduction

1.1 Motivation

ICT industry is a huge industry sector the significance of which is still constantly growing world-wide. In year 2008 ICT sector generated more than 8% of the business value added and employed almost 16 million people in the OECD countries. The value of the global ICT trade was almost 4 trillion US dollars in year 2008 and trend shows that it has almost doubled during the past decade. [1] If we look beyond the numbers, the significance of the information technology can be seen all around us - the digital technology surrounds us in our daily life and a growing amount of work is done using software-based tools.

Software is an integral part of all ICT systems. Although only 4% of the top 250 ICT companies in the world were pure software companies [1], the whole ICT sector employs software engineers and runs software development projects. In addition, the trend is that software is becoming a part of an increasing number of traditional services and industrial products. A recent study approximated that about one third of *all* Finnish industrial companies' turnover came from software-dependent products [2]. Although the result of the study cannot be fully generalized, it gives a hint of the influence of the role of software in the modern society.

The numbers presented above speak for the economic importance of software engineering as an engineering discipline and as a branch of research. The discipline is still relatively young and it has struggled with problems for all of its history that have been mostly related to inefficient work practices. These bad practices have affected both time frame and quality of software projects.

Dijkstra presented the concept of *the software crisis* as early as 1972 [3]. He had noticed that an increasing complexity of the computer systems leads to the increased complexity of software. Without change in the software development processes, this leads to overdue and failed software projects even if new software developers are constantly hired. The infamous Chaos report by the Standish Group published in 1994 presented alarming findings about the state of software projects:

Only 16% of the projects succeeded [4]. Although the validity of the report's success criteria has been questioned since [5], the numbers still indicate that the discipline had some serious delivery problems.

The concept of the software crisis is still familiar to present software practitioners and overdue software development projects are ordinary although the general situation has gotten better. Process thinking in software development broke through during the 90's with large scale methodologies like CMMI [6], SPICE (ISO/IEC 15504) [7], and RUP [8, 9]. During the first decade of the 21st century agile methodologies have brought more light-weight alternatives for systematic software development and increased the reliability of the delivery of software projects.

In this dissertation the concept of software development process is analyzed thoroughly. The main focus is in investigating whether the modern software process modeling techniques would help to further aid software developers in their efforts to deliver good quality software within budget and time constrains.

1.2 Research Theme and Questions

Process modeling is often very process-centric, which means that modeling is used primarily as a tool for improving the processes themselves. Process engineers and quality managers use process models as a medium for communicating changes in development methods to development teams. Data gathered from the development projects is linked to the process models which are then improved further. In this traditional setting the assumption, that good process guarantees good quality of software [10, 11, 12], is often taken for granted. However, during the ground-level development work, the process models are considered the "necessary evil" and are often not actively used and modified.

Experiences from the agile methodologies show that in some cases well-defined processes are not needed to develop quality software. The agile methodologies promote simple and straightforward development techniques without a heavy process overhead [13]. This leads to a situation where there is no actual process to be improved. An interesting matter is to analyze, whether this development will make the process modeling an obsolete technique or is there room for process modeling in the software development environment which is becoming more and more agile.

The most common practice in the software modeling research is to analyze the technical aspects of the software development process modeling, e.g. what kind of notation should be used, how the parts of the models can be merged etc. However, in this work the processes and process models are investigated from the viewpoint of a software development project and an individual developer.

The dissertation analyzes the software development process modeling from several different viewpoints to form a wide understanding of the relationship between the process modeling and the software development work. To guide the

research, two general questions were formed. These questions set the theme for this dissertation and define the area of the software process modeling research to which the dissertation mostly contributes. The questions are:

- Can the software process modeling and models help software developers in their daily work?
- What are the main mechanisms for utilizing the modeling techniques in the daily development work?

As can be seen, these two questions are very broad to be comprehensively answered in a single dissertation. In fact, there probably is not a single, unambiguous answer to the questions. To investigate the questions one has to take many issues into consideration, e.g. what are the possible applications of the process modeling in operational development work, how do developers actually use the modeling and what are their attitudes towards it, how the use of modeling affects the quality of the development work, and how the current modeling principles and techniques could be further improved.

The dissertation investigates the research theme through a series of studies. Each study has a certain viewpoint to the software process modeling: Some studies map the field of software process modeling, some investigate the mechanisms of using the modeling languages and some report how the modeling is applied in real-life scenarios. To further elaborate the research theme and narrow the scope of the dissertation, the following research questions for the studies were set:

Q1. **What are the prerequisites for the process model components that could be re-used in several development scenarios?**

Q2. **What issues are important when a process metamodel based tool is implemented?**

Q3. **How do the changes in process modeling languages impact the process improvement and therefore software development work?**

Q4. **How can the process modeling techniques be applied when the resources for the modeling efforts are low?**

Q5. **How is process modeling used to share the knowledge of a certain software development domain?**

Q6. **What are the opinions of the software practitioners towards the software process modeling techniques?**

The common factor to all the studies is that they focus on analyzing the modeling issues that are relevant in operational software development work and that way they are directly connected to the research theme. Since the dissertation utilizes

mostly qualitative research techniques, no exact hypotheses are formulated. The research questions set above guided the research work and formed the roadmap for the individual studies. Each research article clarifies a research question by analyzing the software development process modeling from a certain point of view.

The research questions are discussed throughout the dissertation and projected to the topic of each article. The final conclusions are made in the concluding Chapter 6, where the research questions are analyzed based on the findings of the individual articles and recent research. The individual findings are also discussed in the context of the general research theme, in order to evaluate how this dissertation contributes to the overall software development process modeling research.

1.3 Contents

The dissertation consists of an introductory part and six research articles. The articles can be found after the introductory part in a chronological order.

In Chapter 2, the key concepts and terminology of software processes and software process modeling are presented. Especially the meta-process of the software process modeling and the Software Process Engineering Metamodel (SPEM) are examined to provide sufficient background knowledge for the dissertation work.

Chapter 3 presents a literature survey to form a perspective on the history and the current state of software process modeling. This kind of analysis is needed to understand the foundations of this study. The analysis also shows how the study field is constantly changing as new modeling tools are introduced.

In Chapter 4, the research process in software engineering is presented as interpreted in this work. Different studies of the dissertation research are linked to this general research process. In addition, the research methods used during the making of this dissertation are presented and rationalized.

The articles that form this dissertation are introduced in Chapter 5. The chapter helps to understand how software process modeling can be used during the various activities of software development work and how the articles answer the research questions. In the first articles the current situation of software process modeling is investigated and analyzed. Based on this analysis, new tools and methods for process modeling are developed and analyzed. In the most recent articles, these methods are utilized in an interdisciplinary setting and validated by a survey amongst software practitioners in Finland.

In Chapter 6, the research theme and questions are re-evaluated and the work is concluded. The dissertation shows that there are several ways in which the process modeling techniques can be used in software development work. However, most of the techniques require special skills that development teams usually lack. The skills can be taught to the team or an external expert can help the team with the modeling. The key issue is whether the benefits from the modeling exceed its costs.

Chapter 2

Background

This chapter presents the key concepts and terminology related to the dissertation study. First the history and the current state of software development processes are discussed, in order to set the context for the software development process modeling. Both the traditional plan-driven processes and the contemporary agile techniques are presented. Second part of this chapter presents the actual software development process modeling concept by explaining notations and techniques used for software process modeling. Essential structures of the Software (and Systems) Process Engineering Metamodel (SPEM) modeling notation [14] are also presented at the end of the chapter, since most of the dissertation research is connected to this metamodel.

2.1 Software Development Processes

IEEE defines *software engineering* as: "(1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software. (2) The study of approaches as in (1)." [15] As the definition indicates *the software development activities* are an essential part, or even the heart, of software engineering discipline. Also, the infamous Guide to the Software Engineering Body of Knowledge (SWEBOK) concentrates on development related topics [16].

Software Development Activities

Software development process is "the process by which user needs are translated into a software product" [15]. IEEE defines the software development process to include activities of *requirements elicitation*, *software design*, *implementation* and *testing*. Usually, also the high-level analysis of the requirements is presented as a separated activity. These activities form the traditional software development life-

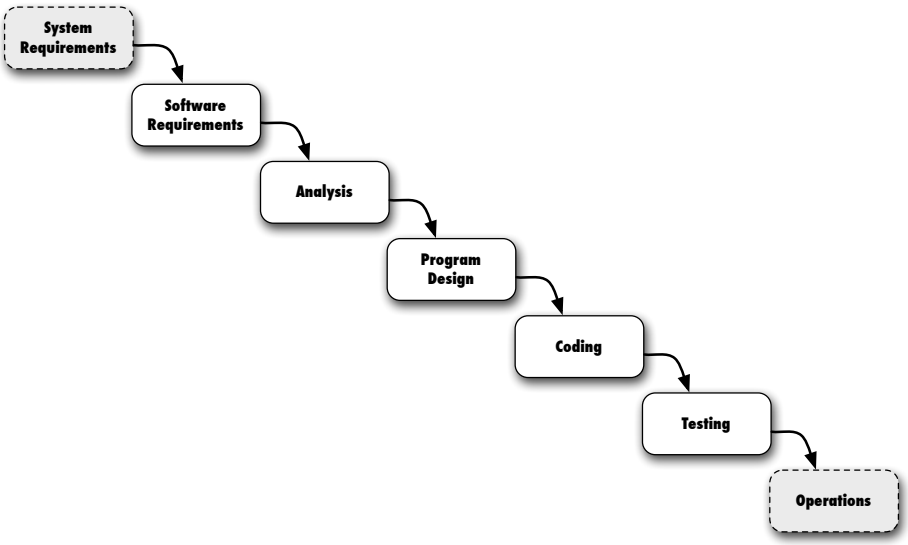

Figure 2.1: The original waterfall model where consequential phases follow each other. The RADIT activities are emphasized in the figure with white background. It should be noted that the Software Requirements phase corresponds the Requirements activity and the Coding phase corresponds the Implementation activity in the RADIT chain. [17]

cycle which is called here *the RADIT cycle* after Requirements, Analysis, Design, Implementation / Integration, and Testing.

The RADIT activities were presented as early as 1970 by Royce, with the introduction of the so-called waterfall model [17]. Although the waterfall model has been widely understood as an exemplary life-cycle model for software engineering, the author actually used it to illustrate a flawed approach for software development. Royce's model is presented in Figure 2.1. Although the waterfall model might be flawed, the activities of the model form the essential backbone of software development. The five RADIT activities present in the waterfall model can be found in almost any software development methodology in one form or another.

The requirements activity concentrates on the customer needs. The customer requirements are gathered into functional and qualitative requirements for the software. *The analysis activity* is not only about requirements analysis which means that the requirements are analyzed and completed if necessary. The analysis also refers to the high-level technical analysis where an architecture of the software system is constructed. The lower level software design is done during *the design activity*. The goal of the activity is to make sure that the overall architecture is followed and that the relationships between software components are working. *The implementation activity* includes the actual programming activities. This activity is

often also called *the integration activity* since the integration of the program code and software modules made by individual programmers is the greatest challenge of this activity. The last activity of the RADIT cycle is *the testing activity*. This activity concentrates again on the customer perspective, because testing activities are done mainly at the system level. This means that testing personnel makes sure that all the requirements are met and that the customer is satisfied with the final software.

The waterfall model is probably the most well-known software development method and the most used example of a software development model. Nowadays, the waterfall model also represents the traditional way of software development. It is used as a deterrent example of a too rigid software development methodology used back in "the old days" [1]. The myth is that before the waterfall model there were no formal processes. This is an erroneous impression. For example, Benington presented a production process for mainframe computer programs already in 1956 [18]. This process is presented in Figure 2.2.

Benington's model is clearly a plan-driven process which feels quite natural since mainframe programming did not leave much room for errors. Therefore the design of the program had to be done carefully before the actual coding. When Benington's model is examined more closely, other similarities to the waterfall model can be found: A design, a programming and an integration activity and even system-level testing and deployment activities can be found.

As said before, this dissertation assumes the RADIT activities to form the core of practically all software development processes. However, the interpretation of the RADIT cycle can be quite loose in terms of the chronological order, the actual contents and the exact boundaries of the activities. The essential idea is that the RADIT cycle contains the core activities that can be found in each software development project, but the actual processes and workflows can vary a lot depending on how plan-driven or how agile the underlying development methodology is. In addition, the individual methodologies can enhance the development process with activities that cannot be found in the RADIT cycle.

The software development process is not the only process in software companies. Closely related engineering activities like *software product development and software maintenance processes* surround the development process. These processes utilize the same kinds of techniques as the software development process, but are usually more heavily connected to the end-user interface. When the number of parallel software projects in a company increases, the need for *product and portfolio management processes* increases. Although these processes still have some engineering components, the emphasis is on the business and management activities.

Depending on the size of the software company, there are also several *non-engineering processes* related to the business operations and the administrative

[1] Which it actually was at the first place already in 1970.

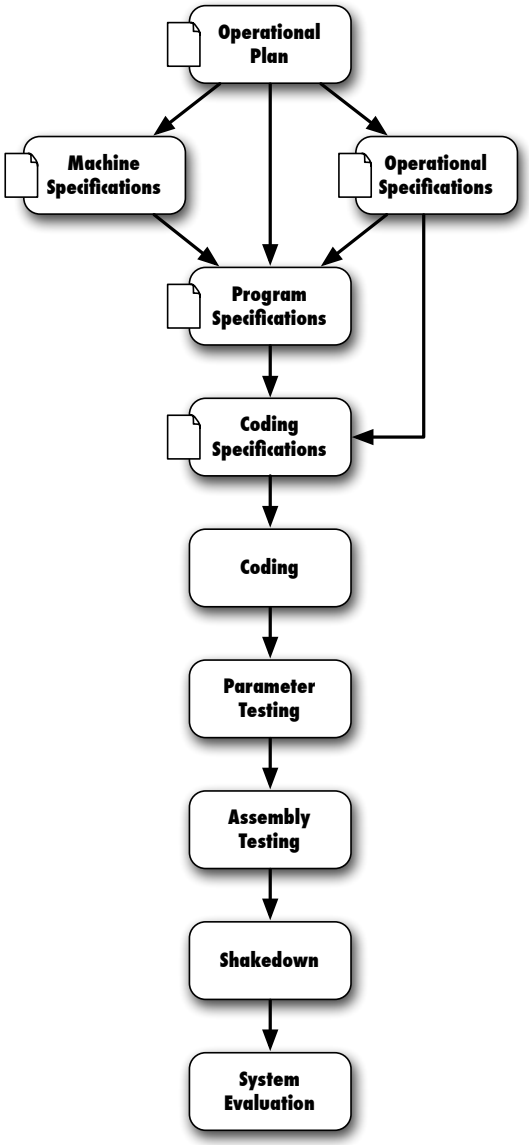

Figure 2.2: Benington's program production model for large mainframe computer programs. Although the paper was published in 1983, it is based on a presentation held back in 1956. [18]

functions of the company. It is also possible that the software is only one part of a larger system in development. In this case there are several other processes from different engineering and design disciplines. In all these cases, the question of integrating very different kinds of processes fluently together becomes important.

Plan-driven Processes

Before the late 90s most methodologies were still based on the concept of separate RADIT style activities, although the importance of the iterative and flexible processes was identified much earlier, e.g. by Royce [17] and by Boehm [19]. A common factor to all these methodologies is that they are based on extensive documentation and in-advance project planning. Hence, they are often called *document- or plan-driven processes*. These kinds of methodologies are often seen most suitable for the development of large, mission critical applications.

As mentioned before, not all plan-driven processes were strictly waterfall-like. For example, in 1988 Boehm presented the spiral model which was iterative and incremental [19]. The main idea of the model was to manage the risk caused by uncertain requirements with several consecutive prototyping cycles. Although the model had many elements that increased the flexibility of software development, it was still very rigid and complex and in the end included the classic waterfall cycle. The structure and the main activities of the spiral model are presented in Figure 2.3. These kinds of iterative and incremental processes can be seen as a transition models between the waterfall style processes and the agile methodologies, which are presented in the next section.

The variety of different plan-driven models and methodologies is huge. Shread has collected the major plan-driven standards into one diagram called the frameworks quagmire [20]. The quagmire is presented in Figure 2.4. The quagmire shows the sheer number of different methodologies which can include hundreds of pages of documentation *each*. This suggests that in order to effectively utilize these methodologies, a company has to have resources for an extensive pre-study even before starting the process implementation.

The concept of *software process improvement (SPI)* is also often connected to plan-driven methodologies. Although the software process improvement can generally mean all activities that target for making the operational software development some way better in an organization, the term SPI usually means formal process improvement activities that aim for implementing a well-defined quality standard (e.g. CMMI [6] or ISO15504 [7] and related methodologies). A good example of an industry-standard, highly disciplined SPI methodology is the IDEAL model [21]. The IDEAL model is directly connected with the CMMI process assessment and improvement method.

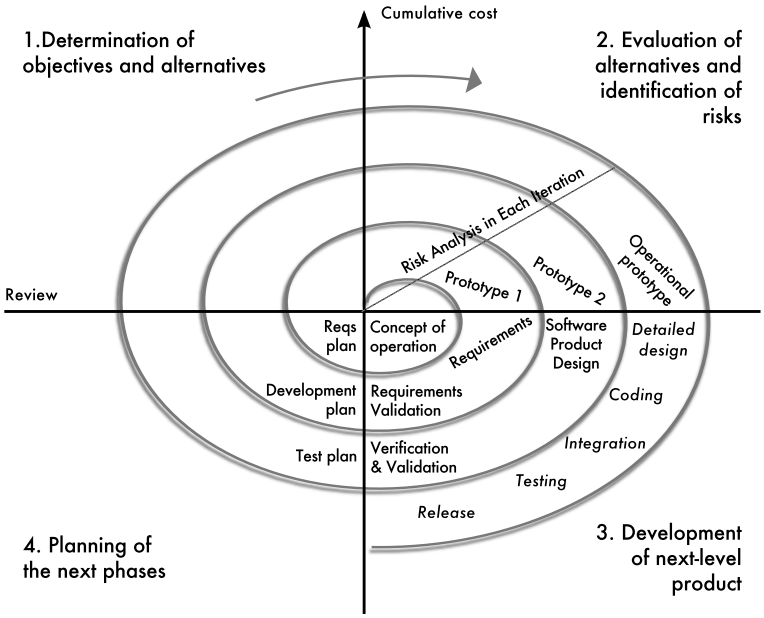

Figure 2.3: Boehm's spiral model of software development [19].

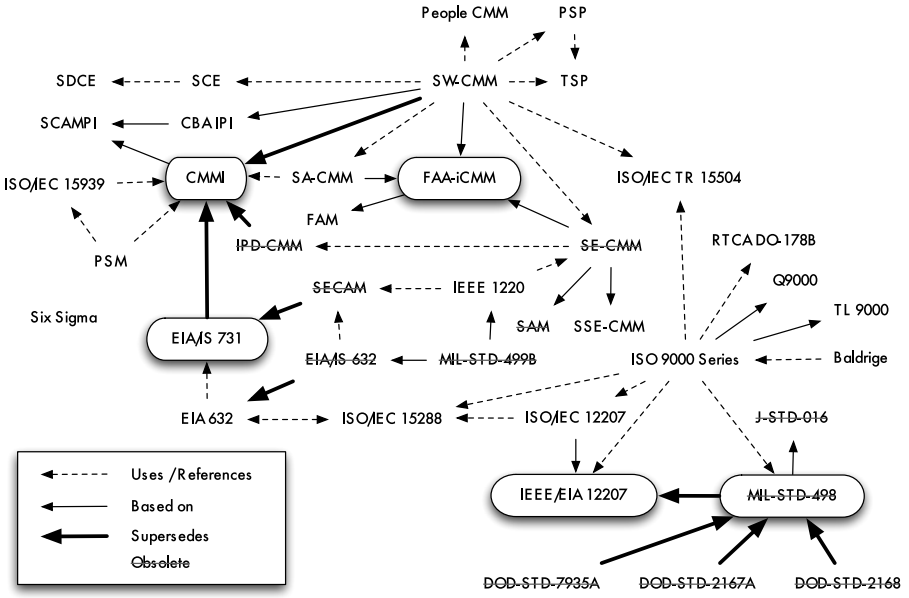

Figure 2.4: The frameworks quagmire. [20]

Agile Methodologies

The plan-driven approach was practically the only option for software development organizations until the 90s, when the movement of the agile software development methodologies begun. For example, Kent Beck developed a very practice-oriented Extreme Programming (XP) methodology [22] and Jeff Sutherland formalized the Scrum methodology with Ken Schwaber [23] at this time. In 2001, a group of software practitioners published the Agile Manifesto [13] that was a symbolical start for *the agile methodologies*.

The Agile Manifesto is clearly a critique against the plan-driven processes, especially the so called waterfall model. The manifesto emphasizes individuals, working software, customer collaboration, and responding to change over processes, comprehensive documentation, contract negotiation, and following the plan [13]. Basically, the agile methods focus on techniques that help the members of a software development team to develop software together rather than defining strict processes and documentation streams around the team and the whole software development organization.

Besides the Agile Manifesto there is no single, exact definition for the Agile methods. Consequently, there are many different methods and techniques which call themselves agile: Extreme Programming (XP) [22], Scrum [24], Feature Driven Development (FDD) [25], and Kanban Method [26] to name a few. An agile method is usually a collection of techniques which help the software developers to do their daily work and react to the change.

The Scrum method [24], which is one of the most popular agile methods, is briefly presented to clarify the agile approach in practice. If Scrum is analyzed from the classical process point of view, where the activities and the workflow

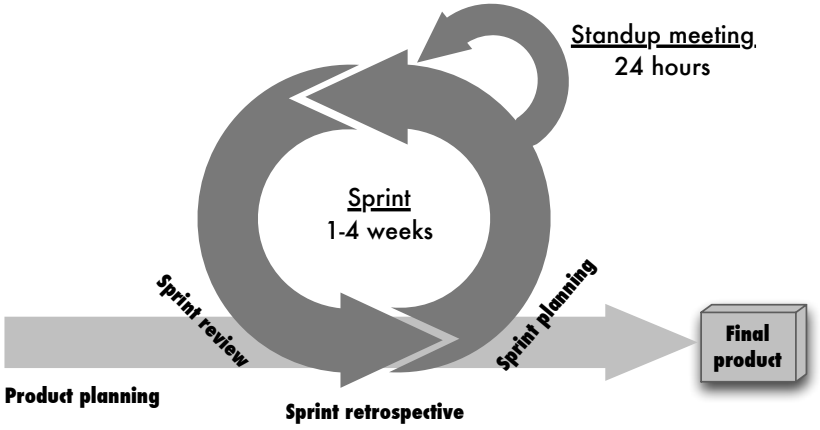

Figure 2.5: Sprint cycles and key activities of the Scrum method. [23]

between them are emphasized, the method is quite straightforward and simple. A Scrum-based project starts with *a product owner* defining *a product backlog*, which is essentially a prioritized list of the key product requirements. The development work is organized into two iterative cycles: The 1-4 weeks *sprints* divide the project into more manageable iterations and *daily standup meetings* help to resolve everyday problems and steer the project work on a regular basis. Each sprint starts with a sprint planning meeting, where the requirements for the particular sprint are transferred from the product backlog into *a sprint backlog*. At the end of each sprint there is *a sprint review*, where the working results of the sprint are presented to the product owner, and *a sprint retrospective*, where the problems in the work practices are analyzed and solved. The Scrum "process" i.e. sprint cycles and key activities are presented in Figure 2.5.

The Scrum example shows, and this applies to all agile methods, that the process and the activities are not the key parts in the methodology. Instead, the Scrum defines a relatively small set of methodological elements namely *roles* (Product owner, Scrum master, Team), *ceremonies* (Sprint planning, Sprint review, Daily Scrum meetings), and *artifacts* (Product backlog, Sprint backlog, Burndown chart). For each of these elements pragmatic, extensive and sometimes strict guidelines are given in order to form a compact and at the same time flexible ruleset for developing better software. The Scrum method, as almost all agile methods, concentrates on improving the work of a software development team instead of organizing the processes of the whole software development organization.

From software process modeling point of view, the main philosophical difference between the plan-driven and agile methodologies is related to the process-orientation of the methodologies. The plan-driven methodologies rely on well- and often extensively defined processes, which are tailored to fit the needs of the development organization. On the other hand, the agile methodologies define a set of very pragmatic techniques from which the best sub-set is selected to serve in the software project's implementation. The combined techniques define an implicit process which is strict enough to guide the development work.

What is said above means that process modeling techniques are a very natural mechanism for defining processes of plan-driven methodologies whereas for agile methodologies the process modeling is much less explicit and extensive. The most dedicated agilists sometimes even claim that agile methodologies and processes are an incompatible match, and the concept of process modeling contradicts with the Agile Manifesto. However, this is not a completely true statement. For example, a well-known Extreme Programming web site [27] uses extensive process modeling techniques to clarify the workflow of the Extreme Programming method and in that way transfer the knowledge to people who are not familiar with the method beforehand.

2.2 Software Development Process Modeling

The term *model* has two somewhat distinctive meanings in the context of software development. Model can mean an established software development method itself or *a definition* of such a method or other software development process. A development method is usually communicated to the software development community in natural language through a scientific article, a book or other media. Examples of these kinds of models are the waterfall model, the spiral model and so on. These kinds of models have been presented in Section 2.1. As mentioned before, model can also mean a definition of software development process which is done in at least a semiformal manner. When this latter viewpoint is taken, the interest is in modeling notations, languages, tools and metamodels. These more technical modeling topics are discussed in this section.

Although it is useful to understand these two meanings of the term software development model, it should be noted that in the end we are looking at the same concept. The only difference between these two meanings is the viewpoint: whether the main interest is in the model content or in the model representation.

Meta-process

In order to understand software process modeling and to analyze the usage of the models during software development, *the software development meta-process* has to be investigated. The meta-process presents the activities and entities related to the life-cycle of software processes, i.e. the meta-process defines the essential concepts of software process modeling. Analysis of these activities and entities tells us what are the actual applications of the process models. This information is important, because different modeling notations suit different needs and different modeling techniques are needed in different situations. Since many software process modeling notations try to be quite general, the meta-process helps to both evaluate the applicability of a certain notation in a certain situation and to analyze the strengths and the weaknesses of the notation.

Feiler and Humphrey present their interpretation of the software development meta-process in [28]. Their meta-process containing the essential process entities and actions is presented in Figure 2.6. Feiler and Humphrey define a process model as ”[a collection of] process elements at the architectural, design, and definitions level, whose abstraction captures those aspects of a process relevant to the modeling.” They continue: ”Any representation of the process is a process model.”

Therefore, the most important process modeling related entities in this meta-process are a process architecture, a process design, and a process definition. When the meta-process is analyzed further *tailoring process architectures, designs and definitions, developing more detailed process descriptions, evolving more general process descriptions, instantiating enactable process, planning the process, and analyzing the process execution via the control process* can be identified as the key

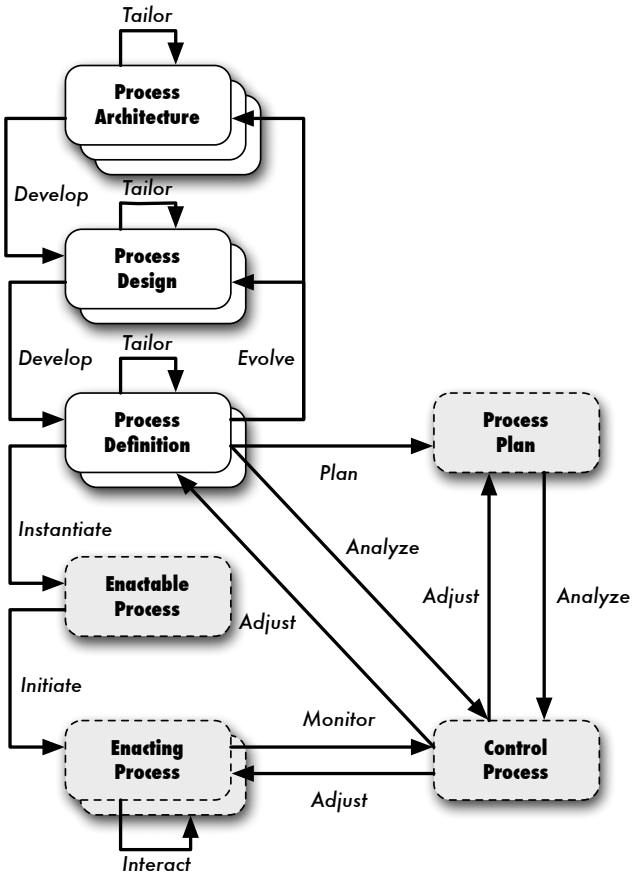

Figure 2.6: Feiler and Humphrey present essential meta-process entities and actions [28]. The most important entities and actions from the process modeling viewpoint are presented in this simplified diagram. Boxes in the diagram represent entities and arrows actions. The entities related to the process models are highlighted.

actions connected to the process models. Feiler and Humphrey [28] support this interpretation by discussing the usage of process models: "A process model can be analyzed, validated, and, if enactable, it can simulate the modeled process. Process models may be used to assist in process analysis, to aid in process understanding, or to predict process behavior."

While process models form only a small part of the Feiler and Humphrey's meta-process, Lonchamp presents his meta-process completely from the process modeling point of view [29]. Despite of a slightly different viewpoint, the key concepts between these two meta-processes are similar. Lonchamp's meta-process consists of the following activities: *process model design, process model cus-*

tomization, process model instantiation, process model enactment, enforcement, automation, simulation, validation, verification, monitoring, evolution, improvement, and feedback.

In Lonchamp's meta-process, the process model design, customization, instantiation and enactment describe the main phases of the process lifecycle that utilize the process models. During the model design, a general process model is defined based on the informal process model or requirements set for the process. The general process model is then customized for a certain domain or a development case. Lonchamp remarks that the distinction between a general process model and a customized process model can be vague, and in some sense all software process models contain both general and customized elements. The customized process model is instantiated by setting the actual parameters of the model from the real-life project setting. These parameters can include e.g. dates for the process activities, actors for the process roles and so on. The last phase in Lonchamp's metamodel is process model enactment, where the model is interpreted and used during the development project. The other activities in Lonchamp's meta-process, besides the main phases, define how the process models can be used in different situations during the process life-cycle.

The meta-process of Feiler and Humphrey and the meta-process of Lonchamp have much in common. As a synthesis of these two it can be said that the meta-process can be divided into two parts: 1) The planning related activities, where the process itself is designed and improved, and 2) The process execution related activities, where the meta-process guides the enactment of the process and the development work itself. Both meta-processes emphasize *the improvement* of processes through *the feedback* from the process execution and *the evolution* of clearly defined and documented processes.

Related to the software development meta-process concept, Armenise et al. list software process modeling objectives as a part of an assessment of the formal software process modeling languages [30]. They find that the software process models can be used for the following purposes: *Communicating, estimating and planning, managing and re-planning, measuring, configuring (tailoring), reusing, executing, and verifying*. Curtis et al. [31] have also listed five basic uses for software process models. Their list is similar to that of Armenise et al. in respect to *communication*, *process management* and *process execution support* use. In addition to these, Curtis et al. mention *automation of process guidance* and *support for process improvement*, which are to some extent implicitly included in the Armenise et al. list.

This use-based approach of Armenise et al and Curtis et al. fits quite well to the approach of this dissertation. Instead of analyzing the underlying software process improvement activities, the list focuses on different usages of the process models. In the end, the modeling purpose defines the key requirements for the modeling language.

For example, if the process model is done *for communicating* the overall process or certain parts of the process, the modeling language should support clear,

visual presentation and should allow the authors to attach some guidance information for model readers. On the other hand, if the process model is done *for reusing* the existing process information, the modeling language should support quick composition of existing, reusable process components and allow the attachment of performance data to the respective components. In addition, a specific tool for building process models from reusable process elements is necessary.

All the meta-level models presented above support each other quite well. The meta-process of Feiler and Humphrey defines the high-level framework for the software process improvement activities, where the process models are only one part. Lonchamp focuses on the low-level activities and defines the meta-process from the process modeling point of view. Armenise et al. and Curtis et al. list ways to use process models in an aspect-like manner within the meta-process.

Characteristics of Modeling Languages

In Section 2.2, the meta-process concept was presented. Next question is, what requirements the meta-process causes for the process modeling and what are the implications of the requirements for the process modeling languages.

The starting point for analyzing these requirements is to comprehend the nature of the software development work. Different approaches of software development work have been discussed in Section 2.1. Although the approaches are different, there are a few similarities in all software projects. Armenise et al. summarize these aspects well in [30]: ”Software production is a creative, intellectual activity, and, therefore, it is not completely formalizable.” In addition, the intangible nature of the software products affects greatly the way in which software is constructed. In other terms, software development is design emphasized work, which targets to create virtual (intangible) systems and services that are usually in direct contact with the end-users.

Based on the nature of software development work, Armenise et al. list requirements for the software process modeling languages in their survey [30]. There are many general requirements for all types of modeling languages, but also distinctive requirements for the software process modeling languages. These characteristic requirements are the following:

- Coexistence of both formally and informally described process parts
- Binding execution of process parts to human beings, computing devices, or other tools
- Controlled modifications during the process enactment
- Both technical and nontechnical activities
- Accompanied analysis tools, e.g. process and product measurement facilities, scheduling and planning tools

- Representation of the consequent actions when a certain activity fails

The requirements presented above are aspect-like for all software development process modeling applications. If a modeling language has to suit for the different modeling needs presented in Section 2.2, it has to support different modeling perspectives. Another option is to utilize different modeling languages for different uses. Curtis et al. [31] have identified the four most common modeling perspectives: 1) Functional, 2) Behavioral, 3) Organizational and 4) Informational modeling perspective. It is interesting that these perspectives are almost identical to the architectural layers of the well-known ARIS business process modeling platform [32]. In the ARIS framework, the perspectives are named: 1) Function, 2) Process, 3) Organizational, and 4) Data view [33, p.56].

In the 90s, the software process modeling was usually a synonym for the development workflow modeling, where the development work is seen as a long stream of consecutive activities. Classic workflow diagrams and more sophisticated languages like Petri nets and FUNSOFT are designed for this kind of modeling. There are also non-visual modeling languages resembling the programming languages that are used for this kind of ”algorithmic” software process modeling. Workflow modeling languages can be further divided into control-based and rule-based languages [34, Fig.1]. Also many modern business process modeling languages still utilize this paradigm.

Software Process Engineering Metamodel (SPEM)

The *Software and Systems Process Engineering Metamodel (SPEM)* [14] was selected as the main modeling language for the research because it was the most prominent modeling notation at the time this dissertation work begun. After a few years, it is also one of the few software process modeling notations that still have industrial support through the Eclipse Process Framework (EPF) and Rational Unified Process (RUP) community. For some reason most of the software process modeling languages presented in 90's have not gained much popularity and thus became obsolete.

The SPEM process modeling language utilizes a bit different paradigm than the workflow languages, which are frequently used in business process modeling. In SPEM the main concern is to model the essential process elements and their relationships. The workflow modeling is also included in SPEM but with a lower emphasis. SPEM version 2.0 does not actually define how the workflow models, i.e. the behavior models, should be constructed [14]. Instead, the standard provides mechanisms for utilizing the existing notations e.g. UML 2.0 [35], BPMN [36], or BPDM [37] for the behavior models.

The high level architecture of the SPEM metamodel is presented in Figure 2.7. The SPEM metamodel has been divided into seven packages which form a layered hierarchy by merging lower lever packages into more complex modeling structures

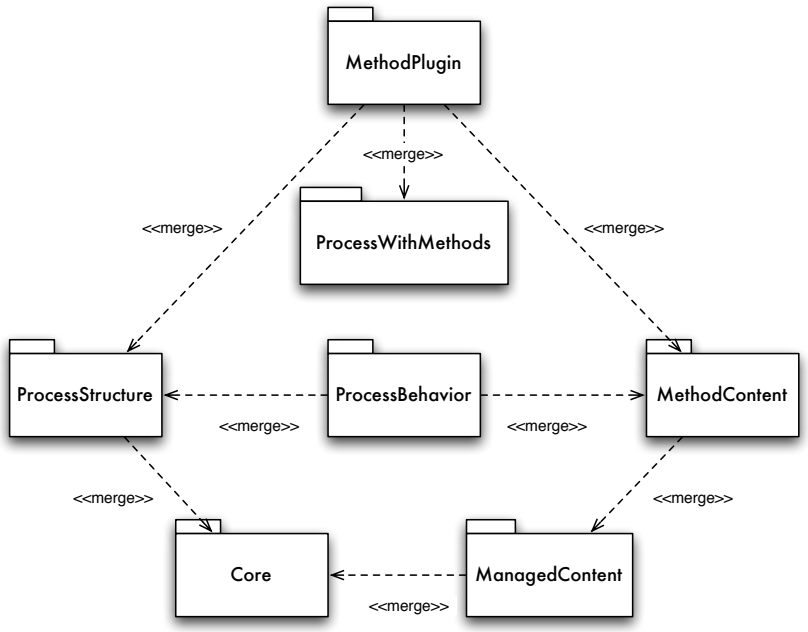

Figure 2.7: The high-level architecture and package structure of SPEM. [14, Fig.2.1]

[14, p. 2]. The goal of the architecture is to give the metamodel a logical structure, to allow extensions of the metamodel, and to limit model implementers' workload by allowing them to choose the most appropriate packages for their needs.

Core package contains all essential structures used by the rest of the packages. The core package is merged into *Managed Content package* which is consequently merged into *Method Content package*. These two content packages contain structures for storing the static process content, i.e. process element data, which is not dependent on the dynamic workflow of the process. The most important static elements are work products, roles and tasks. The static content defines the general relationships between the elements. *Process Structure package*, which also merges the core package, contains structures that are used to model the dynamic content of the process. The dynamic content is basically the work breakdown structures, i.e. the order and the phasing of different tasks that form the life-cycle of the development process. The dynamic content also defines the runtime relationships between the more static process elements. When the static and the dynamic content are compared, it can be seen that the static structures are far more re-usable than the dynamic content which can vary a lot between different process scenarios. Usually static content contains long textual descriptions whereas dynamic content relies more on temporal relationships between the process elements.

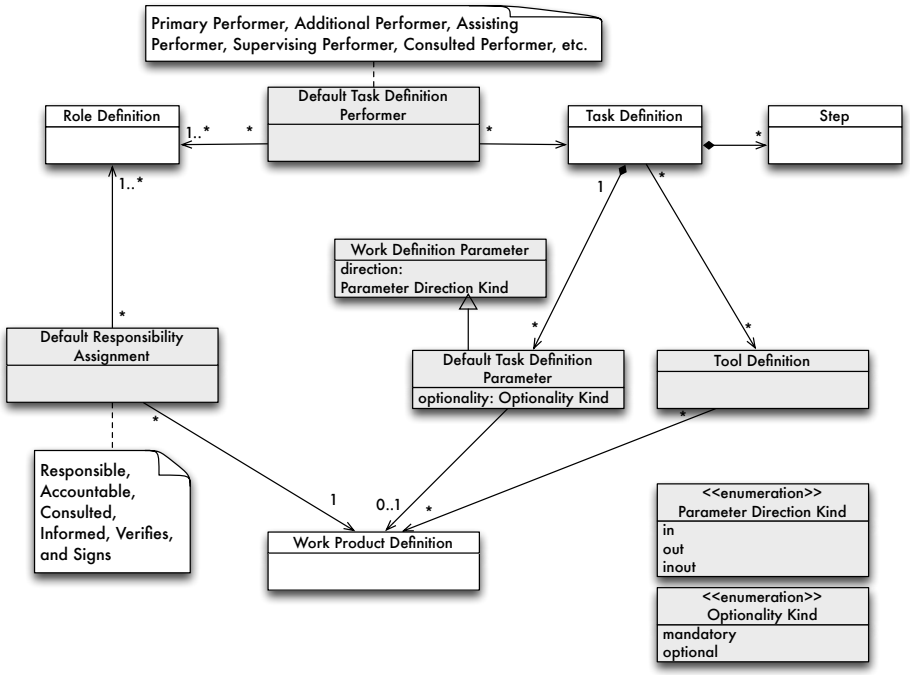

Figure 2.8: Key Method Content elements and their relationships in the SPEM metamodel. [14, Fig.12.3]

Process Behavior package contains mechanisms for using third party behavior metamodels, since SPEM does not define exact ways for behavior modeling, e.g. with swim lane diagrams or state diagrams.

The packages presented above concentrate on modeling the individual processes from different viewpoints. The remaining two packages introduce mechanisms for managing process content and process libraries. *Process with Methods package* contains structures for combining the dynamic and the static process content and therefore avoids the need for re-defining all the process elements for each dynamic process structure. *Method Plugin package* defines structures for building large-scale process libraries with a possibility for process re-use.

The entire SPEM metamodel is not presented in detail in this dissertation. However, some of the most essential concepts are discussed in order to illustrate how the software development methodologies are modeled with a modern process modeling notation. The concepts of work product (artifact), task (activity) and role form the key elements of the SPEM metamodel and to some extent any software process modeling notation. These three elements appear in almost every package in the SPEM metamodel. However, the clearest presentation of the relationships between these key elements can be found in the Method Content package, which

presents the Role Definition, Task Definition and Work Product Definition elements [14, p. 81]. This part of the metamodel is presented in Figure 2.8.

An analysis of the relationships between the work product, the task and the role elements makes it easier to understand their meaning and therefore their use in software process modeling. *Tasks* define the actual work during the software development. They form the backbone of the process. It should be noted that the tasks do not explicitly define the workflow of the process although they are the elementary elements when the work breakdown structures and workflows are constructed. *Roles* define the characteristics of different actors needed during the software development process. The roles do not necessarily have one-to-one mapping to the individuals involved in the process, but one individual can have multiple roles and many persons can work in one role. *Work products* define the things that are worked on during the process. Usually the work products are concrete documents and assets like different plans, source code, and executables, but sometimes the work products can be more abstract subjects like decisions and ideas.

In a nutshell, the three key elements, i.e. the work products, the tasks and the roles, define ”what is done”, ”how it is done” and ”who does it” in a software development process.

Role definitions are connected to Task definitions through *Default Task Definition Performer* relationships, which tell which roles are performing a particular task. A role can be involved in several tasks and a task can be connected to several roles. A role can also be responsible of several work products. A role's type as a task performer can be defined through *a kind attribute*. The kind can be, for example, primary performer, additional performer or supervising performer. Role definitions are also connected to Work product definitions through *Default Responsibility Assignment* relationships, which tell which roles are responsible of a particular work product. The relationship also forms many-to-many connections between the roles and the work products. The responsibility type can be defined using a kind attribute as with the Default Task Definition Performer relationship. The type can be e.g. responsible, accountable, consulted, informed, verifies, or signs. Consequently, Task definitions and Work product definitions are connected through *Default Task Definition Parameter* relationship. This relationship is tighter than previous ones and a particular task aggregates the connected Default Task Definition Parameters, i.e. the connected work products. The Task Definition Parameter relationship has two important parameters. Firstly, a work product that is connected to the task can be either mandatory or optional. Further, a connected work product can be an input work product for the task, an output work product, or both an input and an output work product.

It can be seen that the three key elements form a triangle where each vertex is partly defined by other vertices and therefore are highly dependent of the others.

The use of the key metamodel concepts is illustrated in Figure 2.9. Figure contains an example definition of a partial software process which is modeled using the SPEM metamodel. The process model defines how the planning and the execu-

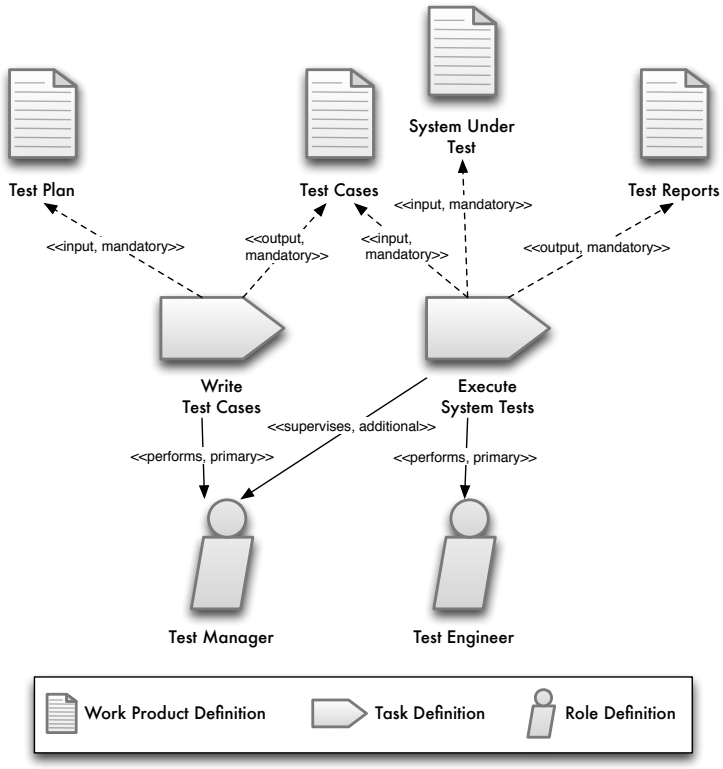

Figure 2.9: A part of a software testing process modeled using the SPEM metamodel.

tion of the test cases are handled during the system testing. The *Write Test Cases* and the *Execute System Tests* tasks are in the middle of the model. The model does not explicitly define the temporal order of the tasks, but this is implicitly expressed using the work products and the input and output stereotypes. The documents and other work products define the information flow between the tasks. In addition, the model defines the involved roles and which tasks these roles perform. It should be noted that the model does not reveal how many Test Managers and Test Engineers are needed to perform each task. Furthermore, the roles' responsibilities on the work products are not defined in this model for the sake of simplicity.

The example model shows that when the reader is familiar with the SPEM notation, it is easier and faster to understand the relationships between process entities than if the same process would have been described verbally. It can also be seen that even a small portion of a whole software development process can take a lot of space when modeled with SPEM. Therefore, in order to keep the process models readable and usable, there is a need to split them into smaller components.

Most of the SPEM-based modeling tools have solved the problem by presenting model entities in a hypertext structure.

More examples on the SPEM models can be found in the *Light-weight Approach for Software Process Modeling - A Case Study* article which is the fourth article in this dissertation. The modeling examples in the article focus on the workflows of certain agile development teams.

Software Process Modeling Tools

Major parts of the SPEM specification have been implemented in the EPF Composer tool. The tool is made by the Eclipse Process Framework (EPF) community [38]. The metamodel used in the EPF Composer is called Unified Method Architecture (UMA) [39, Appendix A], which has been designed in parallel with the SPEM metamodel. The metamodels resemble each other since UMA has been used as a basis for the SPEM 2.0 development work. The UMA metamodel emphasizes the reuse of model components and separation of the high-level method definitions and the process models tailored from those.

In addition to a state-of-the-art software process modeling tool, the EPF community also provides method content authored with the EPF tool using SPEM-based metamodel. The community has published process models [40] based on e.g. the OpenUP [41], the Scrum [23] and the Extreme Programming [22] development methods. These models are authored using the EPF wiki tool which is basically a wiki-style tool for browsing and commenting the models made with the EPF composer tool. The EPF wiki tool is also available as open source.

IBM has been highly involved in both development of the SPEM metamodel and activities of the EPF community. Therefore it is no surprise that IBM provides a commercial tool utilizing the SPEM metamodel and the EPF code base. Rational Method Composer (RMC) software [42] is a part of the Rational Unified Process (RUP) product family [9]. The main difference between the RMC tool and the EPF composer is that the former includes pre-defined RUP-compliant method content. Actually, the RUP methodology is the main product and the RMC is only a tool for organizing the extensive contents.

The concept of Application Lifecycle Management (ALM) [43] relates strongly to process modeling. The basic idea of the ALM is to provide holistic methods and tools for managing the whole lifecycle of a software product including governance, development and operational activities. In order to provide efficient tool support for the ALM, different tools have to be ”configured” to follow desired workflow using process models. For example, a process model could define the timeframe when a developer can submit changes to the main version of the software and when the main version is locked for deployment activities. The Jazz initiative of IBM [44] provides tools for supporting the ALM principles[2]. Also, Microsoft Team Foundation Server includes possibility to use process modeling techniques to customize

[2]Renamed as Collaborative Lifecycle Management (CLM).

process templates, which control the workflow of the team server and the development environment [45].

Chapter 3

Survey on Software Development Process Modeling

A literature survey of the academic publications on the software development process modeling is presented in this chapter. The goal of the chapter is to present the key publications that study the applications of the software process modeling and especially SPEM-based process modeling.

The key publications were selected by the author after a systematic, manual survey on the digital libraries of *IEEE*, *ACM* and *Springer*. These libraries were searched using the keywords *software development process modeling*. About top one hundred hits for each library were analyzed. The hardware and the computer science articles[1] were omitted based on the article title and the abstract of the article. The rest were read and their bibliographies were analyzed further to find more relevant works.

The selection procedure was rather strict, or even picky. The publications which discussed process modeling or its applications, but were mainly focused on some different topic, were omitted. The omitted articles can be coarsely divided into three categories: 1) Studies about technical details of process modeling languages e.g. [46] [34] [47], 2) studies reporting case study results without general findings e.g. [48], and 3) studies using process modeling as means to analyze some other topic like process improvement or process metrics e.g. [49]. Because of the strict selection procedure, the survey does not offer a complete listing of the publications written about the software process modeling, but it gives a good insight on how the applications of software process modeling are studied in the scientific literature.

The publications are presented in chronological order. Therefore, in addition to the presentation of the most influential process modeling papers, the survey also concretizes how the software process modeling research has evolved over time.

[1] Term process is used in both hardware design and computer science to describe an instance of computer program running in a computer system.

The chapter is divided into two sections. In the first section the publications are presented and in the second section a brief summary of the survey is given.

3.1 Key Publications

The key articles about software development process modeling are presented below in chronological order. The authors and the publication year are included in each article. In addition, other publications by the authors of the presented publication are discussed briefly, when they are related to the topic.

In [12] Osterweil (1987) presents an idea for constructing process models using the concepts used in programming. The famous presentation is more like a pamphlet than a true scientific report. Osterweil is concerned about the state of software development and sees that static, informal process definitions are a cause for the inefficient software development. As a solution, he proposes the use of the so called process programs which would better suit the dynamic nature of software development work. Process programs would be written with process modeling languages which resemble common programming languages and are executed during software development projects.

Osterweil's paper is probably the most influential paper in the software process modeling research. At the end of the paper he suggests that process modeling languages and tools should be further investigated. This thought guided the software process modeling research till the end of the 20th century.

In [31] Curtis et al. (1992) discuss the use of process modeling in the field of information systems. The rationale for the discussion is that the modeling of information systems had been very data-oriented. They do not concentrate on software process modeling but use it as an example application. Curtis et al. present four perspectives on process modeling: 1) functional, 2) behavioral, 3) organizational, and 4) informational. These perspectives are still in use in business process modeling [33]. The article also points out some issues of software process modeling, related to formality, granularity, precision, scriptiveness, and fitness of the models. In addition, some uses of the models are discussed.

As a conclusion, Curtis et al. mention that the research around process modeling is still young and, at the time the article was written, it was too early to summarize the characteristics of the research, although they saw that process modeling itself possessed promises.

In [29] Lonchamp (1993) defines a comprehensive terminological framework for software process engineering. Lonchamp lists almost all key concepts related to the topic and gives a detailed definition accompanied with informative comment to each key concept. He rationalizes this work by the confusing state of software process engineering at the time and the need for further discourse and communication between the people interested in the topic.

In [28] Feiler and Humphrey (1993) set key definitions for software process development and enactment. They try to form a general framework to describe the software process improvement activities and responsibilities. At the heart of their framework is a model of entities and activities that are involved in the process improvement activities. The model covers the whole life-cycle of the process modeling from the definition of high-level process architectures to the enactment of the process models. In addition, they compare the process modeling terms to the terminology of the operating system's computational processes.

In [30] Armenise et al. (1993) conduct a survey of the state of the process modeling formalism in 1993. As an introduction they present the main branches of the formal process modeling at the time and also try to list the key features for a perfect process modeling language. As the most significant process modeling notations they present: Adele, ALF, APPL/A, DesignNet, Entity, EPOS, FunSoft, HFSP, Marvel, Merlin, MVP-L, Oikos and SPADE. Most of the presented approaches use syntax similar to program languages. The main purpose of these kinds of models is to form machine readable model for automatic analysis and maybe for simulation.

The four articles presented above [31] [29] [28] [30] define the basic framework for software processes and software process modeling. Also, they provide the meta-process for managing and handling the process model. Therefore these articles should be taken into consideration whenever the process models are discussed. Their work is discussed further in the previous chapter in Sections 2.2 and 2.2.

In [50] Wolf and Rosenblum (1993) introduced a method for obtaining objective process data using simple process models and disciplined process event logging. The authors use a simple timeline based process modeling as a starting point for the process analysis. Basically project workers log manually ending and starting times of all essential process events like normal coding work, analysis work, external work and so on. The authors' hypothesis is that problems in the development process can be spotted by analyzing the intervals between different process events. The authors also present a case study conducted at AT&T. They feel that their approach leads to an objective analysis on the dynamic aspects of the development process. The key finding of the article from the process modeling point of view is that manual data logging of even these kinds of simple process events is very labor-intensive.

In [51] Morisio (1995) proposes a software process measurement method that is based on the formal software process models. He compares three alternatives of process definition, namely an undefined process, a process defined in a natural language and a (semi-)formal process model, and concludes that it is best to use at least semi-formal process models to avoid incoherent process interpretations when defining process metrics. Morisio uses OMT [52] as a modeling language for demonstrating his method. He defines OMT classes of *Process Item*, *Process Activity*, *Process Phase*, *External Product* and *Working Role* as parent elements for his process model, which resembles e.g. the SPEM modeling notation. In an actual

process model, these classes are inherited into more concrete process elements like project plan, verification/validation or g++ library. Morisio has reported about his work also in e.g. [53].

In [54] Ellmer and Merkl (1996) analyze the benefits of the process model re-use. They identify model classification, retrieval and tailoring as key activities to support model re-use in the organization. They also propose a technique for the model re-use which is based on the natural language keywords extracted from the models using a neural network.

In [55] Grundy et al. (1998) present four consequent tools for aiding the distributed software development work. As a starting point for their work, the authors have analyzed computer supported collaborative work (CSCW) systems, process-centered development environments (PCE), and workflow and project planning systems, but all these individual solutions have lacked features needed in distributed software teams. The biggest problem is that all these systems lack good coordination support. All the presented tools, C-SPE, SPE-Serendipity, JComposer, and Serendipity-II, provide collaborative software artifact editors. The last three include a process model editor which is used to define the work process for the distributed team. These tools also include graphical editors which allow the developers to define rules on e.g. how they are informed on the artifact changes and what kind of automatic events are triggered in certain project situations. An interesting feature about these rule models is that the tool allows developers to re-use the rule sets.

In [56] Becker-Kornstaedt (2001) presents a systematic framework for a full-scale process elicitation effort. The framework consists of *process familiarization* and *actual detailed elicitation* phases. Moreover, a schema or a template for defining building blocks for more detailed elicitation activities is defined. In addition to the solution proposal, the article analyses the software process elicitation process. An interesting remark is that most process models are constructed based on the interviews of the process actors. Therefore if the potential problems in the interviews are not addressed (e.g. gathering dispersed process knowledge and focusing on the key actors), the resulting model may become incomplete or even faulty.

In [57] Acuña and Juristo gather articles from the best software process and SPI experts in a book titled Software Process Modeling. The book investigates the process modeling from four perspectives, namely: 1) processes for open source software development, 2) behavioral processes, 3) socio-technico-organizational processes, and 4) software process analysis, definition and evaluation. Although the content is interesting, the book concentrates more on the process contents and the general SPI activities and less on the actual process modeling (e.g. use of modeling notation, activities of the meta-process). Therefore no single article from the book is presented here, but rather the whole book is listed as an example of the common viewpoints to the process modeling topic.

In [58] Porres and Valiente (2006) present an approach for modeling software development processes in a way that supports the principles of the model driven en-

gineering (MDE). In MDE software is constructed using high-level formal models that are transformed into working software through a chain of model transitions. The presented approach is two-folded. Firstly, a process is defined with a planning notation which focuses on workflows between activities, their inputs and outputs, and responsibilities of individual activities. Secondly, the model is executed during a software development process. The idea is that by defining the activities in fine detail, it is possible to follow which steps are already taken and which phases should be done next. The detailed model also makes it possible to automate the parts of the process which use tools to e.g. make transitions to the MDE-based models. The approach of Porres and Valiente can be used independently or as an extension of the SPEM notation.

In [59] Savolainen et al. (2007) propose a method where small companies can use light-weight modeling as a starting point for their software process improvement work. The method is demonstrated through a case study in a small Finnish software company. The key idea of the method is to iteratively build and analyze the process model in a focus group session, convert the resulted wall-chart model into digital format, let the participants of the focus group comment the results, and then repeat the process until the model is good enough. The authors see that the light-weight process modeling is needed even in the small companies to initiate SPI work and establish a base for the following modeling activities.

In [60] Turetken and Demirörs (2007) present an iterative method for decentralized software process modeling called the Plural method. In context definition phase, different roles involved in the processes are identified and responsibilities for different process areas are guided to these roles. In the next phase, the description and conflict resolution phase, role-based modeling is utilized to construct a complete process model in a decentralized manner. A specific coordination team supervises the work and helps to resolve conflicts between individual process parts. In the last phase, the integration and change phase, the individual process parts are merged together. Extended Event Driven Process Chain (eEPC) is used as a modeling notation in the Plural method.

The method was applied for two cases. The authors conclude that the method increases communication between process participants but the lack of proper modeling tools hinders the full potential of the method.

The authors have presented the same idea also in [61]. The concept of the decentralized modeling is based on the Ph.D. work of Demirörs [62] and the related articles [63][64] where the Horizontal Change Approach (HOC-A) method is described.

In [65] Mäkinen and Varkoi (2008) present a method for supporting process assessment with process modeling. Their method consists of doing a preliminary process modeling when preparing for the assessment, constructing a descriptive process model based on the current development process and designing a prescriptive process model based on the descriptive process model enhanced with improvement suggestions from e.g. the SPICE process library. These modeling activities

are further connected to other assessment activities. The authors find that process modeling combined with systematic process assessment activities increase the accuracy of final process models and, on the other hand, make assessment results easier to access and use. The research group has continued their work. For example, Mäkinen investigates the topic further in his PhD thesis [66].

3.2 Summary of Survey

The survey shows that software development process modeling is not a major topic in the software engineering research discipline. Currently the number of publications on the topic is rather low compared to more popular software engineering topics. Also, based on this survey the research is not very interconnected between the research groups. This does not mean that the software process modeling research is irrelevant but it is usually done in parallel with other software process improvement research.

The temporal analysis of the publications shows that the software process modeling research saw its peak in the nineties. There were several research groups investigating the topic and they wrote the same kinds of papers at approximately the same time. This shows that there was some kind of dialogue between the research groups. In the nineties, the software process modeling research seemed to fade away for a few years to return around 2005. While the software process research of the nineties was focused on analyzing the process modeling languages, the current research is more scattered and focused on more specific process modeling issues.

Amongst the publications on the topic, there seems to be only one article that is frequently cited by the following studies. It is the *'Software processes are software too'* paper that Osterweil published in 1987 [12]. Osterweil approaches the software process modeling in a very formal way and this might be one explanation to why the modeling research in the nineties was very focused on investigating the modeling formalism instead of its actual use and applications.

The research work done in the *Method Engineering* [67] community is very close to the topic of this dissertation. However, there are only a few references to the method engineering research. The reason for this might be that the method engineering community concentrates on building re-usable process modules, which was not considered pragmatic in the context of this dissertation (see the first dissertation article [68]). On the other hand, the method engineering community has a slightly different viewpoint to the topic: They see process modeling as a complex framework for managing the building of usually big and complex information systems, whereas this dissertation analyzed how the individual development teams could benefit from the process modeling techniques.

Another research community that is close to the topic, but is not widely covered in this dissertation, is the *Model Driven Engineering* community. They see

that models can be used to control and to guide the software development from the beginning to the end. Therefore process models could be used to plan the development efforts and synchronize the automated work done through models and the work done by developers (see e.g. [58]). The author of the dissertation sees the model driven engineering approach as one possible development method amongst others and therefore has not especially concentrated on that topic.

Chapter 4

Research Method

4.1 Software Engineering Research Methods

The natural way in the context of this dissertation is to approach the research by constructing a high-level process model of the common activities in the software engineering research. Software engineering can be analyzed from a few different viewpoints, e.g. high-level program structures are analyzed by the computer scientists and the software engineering as an organizational function is investigated by the information systems community. From the perspective of the engineering discipline, the target is to challenge the old methodologies and tools by trying to develop improvements in a systematic way.

The author followed the high-level research process, presented below as a list of phases from a very early stage of the dissertation work. This process is based on the author's own observations and experiences of the software engineering research.

1. Analysis of the current situation
2. Position paper on the current situation and its problems
3. Development of the improvements
4. Small-scale proof-of-concept
5. Technical report of the improvement and initial results
6. Pilot projects in the industrial context
7. Analysis of the pilot projects
8. Article on the pilot project results
9. Possible industrial adoption of the improvements

10. Case studies on several industrial adopters

11. Articles on the success of the improvements in the industrial setting

The process-based approach for software engineering is not the author's own invention. In [69] Basili discusses various research paradigms related to the software engineering research. He presents three experimental models: the scientific method, the engineering method and the empirical method. The latter two are the most interesting ones in context of software engineering research, from the viewpoint of the engineering discipline.

The engineering method is seen as an evolutionary method; it targets to improve existing technical solutions. The engineering method consists of the following steps: "Observe existing solutions, propose better solutions, build / develop, measure and analyze, and repeat the process until no more improvements appear possible". The empirical method is seen as a revolutionary method; it targets to bring a new method into an organization and to analyze its effects. The empirical method consists of the following steps: "Propose a model, develop statistical / qualitative methods, apply to case studies, measure and analyze, validate the model, and repeat the procedure". [69]

A very coarse generalization is to say that the engineering method suits better for studying software engineering as an engineering discipline whereas the empirical method suits best for the information systems research. The reality is that mixed methodologies are constantly used, and software engineers, computer scientists and information systems researchers attend the same conferences and investigate similar issues. Sometimes researchers with even more diverse background (e.g. economics, social sciences) work on topics related to software engineering.

When the Basili's research paradigms and the author's ideas about the engineering research are compared, similarities between them can be found. The steps 1 to 5 of the author's process are almost identical to the engineering method of Basili. The steps 6 to 11 are more pragmatic and may be simpler than in Basili's models but the essence is the same as in the empirical method of Basili.

It is very natural to combine the engineering and the empirical approaches in software engineering research. Software engineering methods and techniques can be improved with the evolutionary engineering method in more closed university or single company settings. When the method is good enough, a wider adoption can be sought and the revolutionary effects in the software industry can be analyzed with the empirical method through case studies. Reasoning behind this approach is that testing of new software engineering techniques can be quite time-consuming and expensive. Therefore it is difficult to get companies to adopt too immature or minor improvements. Via the engineering method confidence about the new method can be gained in order to start wider pilot projects. This idea and both the engineering and the empirical methods are presented in Figure 4.1.

It should be noted that especially in the engineering discipline the academic research partially overlaps with the research and development work done in the

industry. Similar investigation methods and techniques are used e.g. to analyze the efficiency of a company's development processes as in research projects focusing on SPI issues. The main difference is that the academic research has higher risk potential and longer time-span. The academic research usually does wider analysis whereas the industrial R&D activities target to benefit mainly the company doing the research.

It can also be argued that the ultimate goal for the scientific research should be to formulate general theories based on unbiased findings and observations. Building general and widely applicable theories can be difficult in the software engineering research, because in many cases the research is based on separate case studies. Theory building in this kind of setting requires ability to identify the scope of each individual research effort and patience to collect sufficient evidence from multiple sources.

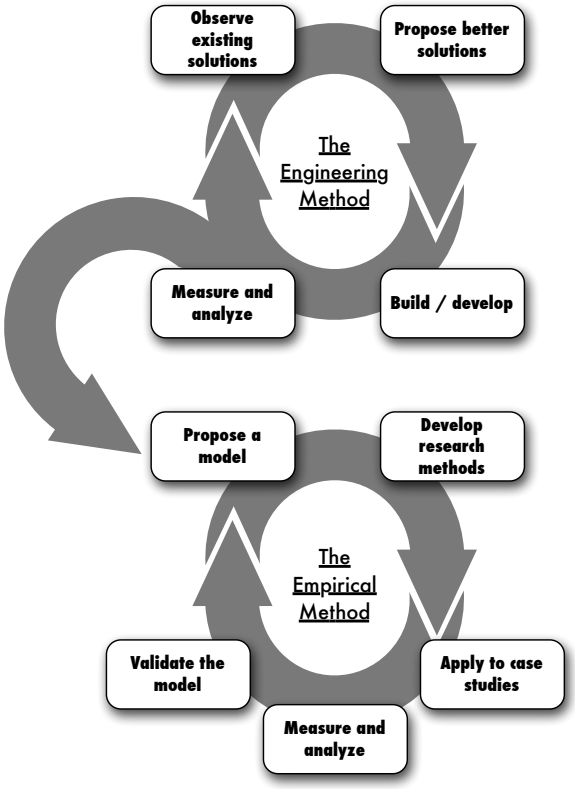

Figure 4.1: Research paradigms presented by Basili. [69]

4.2 Methods of the Dissertation Study

As mentioned before, the dissertation research followed the author's own perception of the software engineering research process from very early on. The activities of this process were presented in the previous section. It was later discovered that the outline of this process could be well aligned with Basili's engineering and empirical research paradigms [69].

When the research articles presented in this work are compared to the research process phases presented above, it is seen that the dissertation work has followed the intended research process quite well.

In *Observations on Modeling Software Processes with SPEM Process Components* the current situation of the process modeling languages is analyzed. The article resembles a position paper where problems in forming re-usable process components are investigated. The work for the article covers Steps 1-2 of the presented research process. The article discusses Research Question *Q1* which was presented in Section 1.2.

In *Spemmet — A Tool for Modeling Software Processes with SPEM* the starting point of the research was the lack of a proper process modeling tool that would utilize the latest SPEM standard. Therefore such a tool was developed as a proof of concept. At the same time difficulties in the implementation of the metamodel were analyzed. The work for the article covers Steps 3-5 of the presented research process. The article analyzes Research Question *Q2*.

In *Changing Role of SPI — Opportunities and Challenges of Process Modeling* the research group had gained enough experience to analyze the challenges and the opportunities of the new modeling languages from the software process improvements point of view. The article is clearly a position paper that articulates the effects of these changes. The work for the article also covers Steps 1-2 but from a wider perspective than the first article. The article discusses Research Question *Q3*.

In *Light-weight Approach for Software Process Modeling — A Case Study* some ideas presented in the previous article were tested in a real-life pilot environment. The research work mixed work techniques from both the engineering and the empirical disciplines. The article presents the pilot case results. The work for the article covers Steps 6-8 of the presented research process. The article examines Research Question *Q4*.

In *Three Approaches Towards Teaching Game Production* the process modeling techniques were used to communicate process level issues to other researchers in a specific software development domain. It can be seen that the work done for the article covers Steps 7-8 of the presented research process. The article acts as an example case which clarifies Research Question *Q5*.

In *Survey of Practitioners' Attitudes To Software Process Modeling* the experiences of a wider industrial audience were gathered. The article presents how the process modeling has succeeded in the Finnish industry. The work for the article

covers the final Steps 10-11 of the presented research process. Step 9 was omitted i.e. the researchers did not actively push the new techniques to the industry. However, the next steps were possible because the industry was already adopting the modeling techniques by itself. The last article of the dissertation discusses Research Question *Q6*.

The dissertation research utilized several different research techniques. As the research utilized mostly the qualitative research techniques, only a very simple statistical analysis was done in a couple of cases. The used qualitative methods were literature surveys, case studies and a qualitative survey. Also a proof of concept (PoC) was constructed. The significance of the PoC as a purely scientific research method can be questioned. However, it is an important phase in the engineering research paradigm and can reveal new perspectives on the investigated topic.

The main sources of research theory for this work are [10, 70, 71]. In [70] and [71] the research methods used in empirical software engineering research are presented in general and good references for detailed research method articles are provided. In [10] a detailed introduction on software metrics is given. Although this work uses mostly qualitative methods, the book gives excellent guidelines on how to interpret non-quantitative results and what are the limits of numeric techniques when the samples are small.

Chapter 5

Articles

In this chapter the articles which form this dissertation are presented in chronological order. It will be shown how the articles form a consistent story answering the research questions of the dissertation. For each article the general goals of the article, the used research methodology, contribution to the research questions, and other significant findings are presented. Similar work found in the literature, after the article was published, will also be presented with each article.

5.1 Observations on Modeling Software Processes with SPEM Process Components

The study presented in this paper begun the research group's work with the software development process models and modeling languages. The purpose of the study was to analyze how the SPEM 1.1 version [72] supports component based process modeling. For example, would it be possible to publish commercial process components that could be integrated to a company-specific, SPEM-based process model.

During the study the SPEM 1.1 documentation was analyzed in order to find out how well the language supported process components. Parts of two real-life process frameworks, namely CMMI [6] and RUP [9], were modeled with SPEM as process components and these components were compared to each other.

The results of the study reveal several technical shortcomings in SPEM 1.1 standard that make the construction of process components difficult or even impossible in practice. Most of the problems relate to the fact that the process component mechanism in SPEM 1.1 is not very well documented.

In addition to the findings connected to the SPEM language, the study also points out more fundamental difficulties in constructing arbitrary, re-usable process components. In order to connect the modeled components to each other without extensive tailoring, the component interfaces should be compatible. If the components were modeled from the same process framework, this could be feasible. Alas,

fitting together two components modeled from two different frameworks is considerably harder, because of the foundational differences on how the frameworks are organized.

The article acts as a kick-off for the software process modeling research. The article can be seen as a pre-study survey where the essential literature was reviewed and the possibilities and limits of the current technology were investigated.

The study answers *Research Question Q1* of the dissertation: "What are the prerequisites for the process model components that could be re-used in several development scenarios?" As mentioned above two issues have to be resolved. Firstly, the metamodel used for modeling should provide sufficient support for component-based modeling. Secondly, the process components should follow the same general process framework. If these two prerequisites are not met, the modeling will become considerably harder and require much more effort and expertise.

The author did the study together with Antero Järvi. The author investigated the technical details of the modeling standard and did most of the modeling work, whereas Järvi fitted the findings to the context of process modeling and lead the writing of the article.

Some work related to the process model notations and especially model re-use has been presented in Chapter 3, e.g. [54].

5.2 Spemmet — A Tool for Modeling Software Processes with SPEM

The article presents an effort for making a working modeling tool for constructing SPEM 1.1 compliant models. The main reason for the effort was that no publicly available SPEM modeling tools existed at the time and the research group required such a tool to continue the analysis of the SPEM modeling mechanisms. There was a commercial SPEM tool from Osellus [73] but there was no evaluation version of it. The Softeam SPEM modeling tool was also evaluated but it was in beta stage at that time and did not work properly. Also, both tools focused on the process enactment through pre-defined process content and implemented the SPEM specification only partially.

The article presents the implementation project and the main architecture of the Spemmet tool. The tool was using a client-server-architecture in web platform. The idea was that multiple users were able to edit the same model simultaneously.

The implementation effort showed that the SPEM metamodel can be used as a basis for a real-life modeling tool. The implementation was quite straightforward and the tool was up and running within a couple of months. It also became clear that the SPEM metamodel was partially ambiguous and therefore hard to actually implement.

The Eclipse project published the first version of the Eclipse Process Framework (EPF) Composer modeling tool just when the second version of the Spemmet

tool was under construction. Because the original purpose for the Spemmet tool was just to enable further research activities, the research team decided to cease further development activities regarding the Spemmet tool and focus on utilizing the new EPF Composer as the main modeling tool during the research activities.

Although the life span of the Spemmet tool was quite short, it helped the research team to do some initial SPEM modeling by e.g. modeling Microsoft Solutions Framework (MFS) [74] with SPEM. This was later transferred to an EPF Composer model. The implementation effort of the Spemmet tool also revealed several implementation related shortcomings in the SPEM standard, which would not have been necessarily noticed with a third party modeling software.

The article discusses *Research Question Q2* of the dissertation: "What issues are important when a process metamodel based tool is implemented?" Here the devil is in the details. The tool vendor perspective is not always present in the metamodel definitions, which will lead to a situation where the tool vendors have to interpret the metamodel definition when they are implementing the process modeling tools. If the metamodel definition is too ambiguous, as the SPEM 1.1 definition partially is, different modeling tools could produce incompatible process models.

The author implemented the Spemmet tool alone, and started the implementation of the second version. The author also did most of the writing work for the article, whereas Antero Järvi acted as a supervisor and mentor for the work.

5.3 Changing Role of SPI — Opportunities and Challenges of Process Modeling

The article analyses the opportunities and the challenges the new process modeling techniques impose on traditional Software Process Improvement (SPI) activities. The main motivation for the article was the finding that the SPEM modeling language made new kinds of modeling techniques possible and therefore critical analysis of the existing modeling applications was necessary.

The main research method for the article was a literature survey combined with the authors' analysis work. The authors held several workshops where modeling ideas from the various sources were put together and contrasted against the current situation of the software process modeling.

The authors found five key concepts of SPI that are mostly affected by the new process modeling techniques. The five key concepts of the SPI are: 1) Business, project and process coherence, 2) process frameworks, 3) process definitions, 4) SPI cycle, and 5) organization's capability. It is analyzed what kind of opportunities and, on the other hand, challenges modern modeling techniques impose on these key concepts.

It is hard to devise one single conclusion or summary about the article's results, because the article tries to make a broad analysis on the modeling techniques'

impact onto the SPI. Instead, all the key concepts contain findings that are valid even today, several years after the original publication:

Business, project and process coherence. It is evident that there is a connection between organization's business context and both the processes and the projects. Stable business factors directly guide the organization's SPI activities and process development. The link between projects and processes has traditionally been more indirect, since process engineers have to gather typical project factors and take them into consideration during the SPI activities. New process modeling techniques enable quicker model changes and therefore more direct connection between projects and organization's processes.

Process frameworks. Each process framework makes dominant assumptions about the organization's way of doing software. In order to make most out of the framework, each organization should selfishly cherry-pick the frameworks that suit best for each task at hand. Process modeling would help this activity and allow the organization to manage several parallel frameworks.

Process definitions. Process definitions make traditional SPI activities possible, since the process definitions make the process visible and act as a medium for designing the improvements. The SPEM standard defines the actual notation in very great detail but does not guide how the notation should be used to construct the process definitions. Therefore it is important to make general guidelines on how the process definitions are made with SPEM in different situations and how these definitions support the SPI work.

SPI cycle. The SPI cycle (or SPI process) contains several steps which are presented in the same manner in almost every SPI framework. The new modeling techniques would allow the practitioners to speed up the SPI cycle and that way reach for example better business, project and process coherence that were mentioned above.

Organization's capability. Stakeholders in a software development organization use the process models differently. Even within a stakeholder group, e.g. process engineers, project managers or developers, the skill set of individual persons may vary. Therefore it is important that a modeling language supports different views into the process models, which support each stakeholder's work in the best possible way.

The article can be categorized as a position paper which tries to identify the main problems with the software process modeling, but also the opportunities after the problems are addressed. The article gives the mind set or "philosophical framework" for the rest of the dissertation research and is therefore very important. Especially influential for the rest of the dissertation work are the ideas about business, project and process coherence.

The article comprehensively answers *Research Question Q3* of the dissertation: "How do the changes in process modeling languages impact the process improvement and therefore software development work?" The main message is that modern software process modeling notations, like SPEM, can make the gap between

process modeling and actual software development work narrower. However, this requires good tool support and tested practices for modeling processes in a manner that supports both the goals of process engineering and daily software development work.

The author of the dissertation participated the workshops and the writing process with equal contribution as the other authors. However, it should be noted that Järvi and Hakonen as senior researchers provided valuable insights to the article and therefore the author was very lucky to be a part in such a team.

5.4 Light-weight Approach for Software Process Modeling — A Case Study

The research team wanted to experiment with the actual modeling after the release of the EPF Composer modeling tool. In the literature, there are only a few guidelines written on how the software process modeling should be done, whereas on the technical notation details there exists plenty of documentation (see Chapter 3). In the business process modeling there exists studies on how the actual modeling is done, but these modeling efforts are usually very extensive, lasting months or even years.

During the study, the research team examined if it is possible to get sane models with very light-weight modeling techniques. Based on existing knowledge on the modeling, the team developed a very straightforward and simple modeling process, which was then tested in the GAUDI software factory of Åbo Akademi university.

The testing was done by first modeling an initial process model based on the written documentation on the GAUDI work methods. Then several GAUDI team members and projects managers were asked to make corrections to the initial model and this way the model was constructed in an iterative manner.

The results show that achieving light-weight process modeling is possible and the resulting models are usable. In this study the final model had a dual purpose. Firstly it showed that although the process was very well documented in the GAUDI software factory and the process guidelines were well followed, there were actually two different implementations of the process in use, based on whether there was an actual customer involved in the project or the team coach acted as a customer proxy. The team was unaware of this kind of difference. Secondly, the model was good enough to act as an initial model for more serious modeling efforts.

The study showed that process modeling techniques can be used close to the day-to-day development work context. Using this kind of light-weight modeling method, a software team can easily analyze their process and compare it with other teams within the organization. Of course, an experienced process engineer is required to do the actual modeling work, because the training would take some time.

The paper was done in co-operation by Luka Milovanov, Antero Järvi and the author. The author was primarily responsible for designing the study, planning the modeling process and writing the article. Luka Milovanov from the GAUDI software factory offered great help with the GAUDI work methods and provided an insider's view to the process.

The paper answers *Research Question Q4* of the dissertation: "How can the process modeling techniques be applied when the resources for the modeling efforts are low?" The paper presents a method for conducting light-weight process modeling with minimal human resources. The resulting model can be used as a starting point for a larger modeling work or as a snapshot of current work processes. The downside of the method is that it requires a skilled modeler who is familiar with the modeling notation. There are also limitations to the size of the modeled process and the organization using the process, since the modeling method relies on the direct communication and interaction between the modeling team and the people working in the modeled process organization.

The same kind of ideas about light-weight process modeling exist in literature. Some of them were published after the study was done.

Becker-Kornstaedt presents a systematic framework for a full-scale process elicitation effort [56]. The framework consists of *process familiarization* and *actual detailed elicitation* phases which both may require several iterations. This framework is too extensive for the light-weight modeling purposes although some analogies may be drawn between the light-weight modeling and the process familiarization phase. Becker-Kornstaedt also discusses the problems that may arise during the elicitation, which can be applied to light-weight modeling situations.

Other light-weight modeling approaches have been proposed using other modeling notations or techniques. Conradi et al. propose the use of *a coarse process model* as an initial point for their method to support project-wide process evolution [75]. However, they do not cover the actual modeling techniques in detail. Savolainen et al. propose a method where small companies can use light-weight modeling as a starting point for their software process improvement work [59].

It should be noted that *process mining techniques* [76] could also be utilized for generating initial models for the light-weight modeling. This naturally requires existing data from the previous projects. The generated models could also be so inaccurate it could be easier to model the process from scratch. In these cases the generated models can be used to validate the light-weight models quickly.

5.5 Three Approaches Towards Teaching Game Production

There has been an active game algorithms research group at the University of Turku for several years. The research has led to game development teaching activities which the author of the dissertation has also participated. Therefore it was quite

natural to combine the knowledge of the process modeling techniques with the lessons learned during the game development courses. In the article *Three Approaches Towards Teaching Game Production* the authors disseminate the knowledge gained in the game development courses to the game teaching community using the process modeling techniques.

The article analyzes game development on two levels: On general game development process level and on the level of individual development courses. Firstly, the general game development process was investigated in order to situate the pedagogical contents into right context. This investigation was done by comparing several game development processes found in the literature with a general software product development process. For each of these processes a high level model was done in order to make the comparison easier.

Secondly three different types of game development courses were analyzed. A light-weight process model was made for each course type. These three models visualized the main teaching activities and the key responsibilities of teachers and students. The models showed the similarities and the differences between the various course types. Together with the general game development process analysis it was easy to point out which parts of the game development cycle were taught during each course.

An interesting result was that the game development process actually matches the software product development process rather than the technical software development process. This fact was revealed by modeling the game development and the software product development systematically and comparing the models. The paper shows that light-weight process modeling techniques can be used to disseminate process models even in a multi-disciplinary setting.

The article acts as a proof of concept in using the process modeling techniques to share process knowledge to a wide audience in a very visual manner. The light-weight modeling techniques presented in the previous paper were successfully used to model various types of game development related processes. This shows that even the light-weight process models can reveal new details about the process under investigation.

The study indirectly answers to *Research Question Q5* of the dissertation: ”How is process modeling used to share the knowledge of a certain soft- ware development domain?” This is done by providing a case example where the process modeling is used in the domain of game development and teaching of it. The study shows that the process modeling techniques can be applied to analyze process-related problems and to share the process-related choices with a wider audience. In the study a few high level game development processes are compared with a software product development process. The model comparison shows that the game development processes usually define the whole product development life cycle rather than just development methods. Also, three different methods for teaching game development are defined using process modeling techniques. This allows

university teachers to discuss different approaches to teach game development and possibly apply new ideas to their teaching.

The main contributors to the paper were the author and Harri Hakonen. Hakonen provided the pedagogical background for the study while the author did most of the modeling and related process analysis. Extended version of the book chapter was published as a technical report with a more detailed theoretical analysis [77].

5.6 Survey of Practitioners' Attitudes To Software Process Modeling

Most of the software process modeling research concentrates on studying the technical aspects of the modeling languages and notations. It is unclear how the people in software development organizations see the process modeling as a tool and a technique for making better software. This is an important research issue because opinions of the software practitioners a) reveal problems in existing modeling techniques, b) tell which are the most important pragmatic use scenarios of the process modeling and c) hint the future modeling needs of the industry.

In the article *Survey of Practitioners' Attitudes To Software Process Modeling*, a qualitative survey of the software practitioners opinions on the process modeling techniques was conducted. The web-based survey was sent to several software development companies in Finland through the researchers' partner channels. Twenty practitioners from fifteen different Finnish software companies answered. The survey was divided into three parts: 1) Background questions, 2) Opinions on the current process methodologies and modeling, and 3) Expert analysis on the upcoming modeling trends. The main focus of the questions was to analyze how the practitioners see and feel the modeling techniques in their everyday work.

As upcoming modeling trends the following modeling methods were presented and opinions of the practitioners were asked: 1) Distributed process modeling, 2) Light-weight process modeling and 3) Decrease of the process-project gap. The last trend means techniques that bring process models closer to the everyday project world and enable the usage of the same models in both process and project planning.

As mentioned above, the survey was conducted in a qualitative manner. This means that most questions in the questionnaire were of open form. Naturally, the length and depth of the answers varied and therefore most of the work was done after all the answers were gathered by identifying key words and common themes from the answers. Although the qualitative study is more prone to a subjective bias than pure statistical studies, the approach was suitable for this initial survey since the researchers got more rationalized answers. The qualitative approach was also more suitable than the quantitative because the sample of the study was quite small and therefore statistical relevance would not have been met.

The results of the study show that most of the companies utilize some kind of development methodology. The respondents saw that the methodologies cause some overhead and may be inflexible, but at the same time bring the needed structure to the software development work. The use of process modeling was quite rare and only a few of the respondents knew that their company used some kind of standard modeling notation. The most common uses for the process models were to access the document templates and check some individual parts of the process methodology. Only a few of the respondents used the process models to enact and improve the development processes.

An interesting finding was that the respondents' opinions towards the emerging modeling trends were quite positive. The conclusion might be that although the respondents are not completely satisfied with current modeling techniques they see potential in process modeling if it would really support their own work.

The survey is an appropriate conclusion to the dissertation research since it binds the theoretical research to the practice. The previous studies tried to understand the modeling field as a whole and developed modeling methods that would improve the existing situation. In this study most of the findings in the previous studies were implicitly exposed to the public debate by the survey respondents. The fact that the survey results did not reveal any big surprises tells that the premises set in the previous studies were somewhat correct also from the software practitioners' point of view.

The study answers to *Research Question Q6* of the dissertation: "What are the opinions of the software practitioners towards the software process modeling techniques?" As mentioned above, the most surprising finding was that the practitioners were interested in the advances in the field of process modeling and saw the applications of the process modeling promising. However, the current situation of the process modeling techniques and tools was not perfect in many ways.

If the other reported studies are evaluated, it is found that the findings of the study are mostly in line with them. Turetken and Demirors found out in their decentralized modeling study that the lack of proper tools was one of the greatest obstacles for the approach [60].

5.7 Further Work

The dissertation research concludes with the small qualitative study on the practitioners' opinions of the current state of process modeling. The next natural step would be to conduct a much larger statistical study on the subject based on the findings of this study. The target would be to analyze whether the findings done in some Finnish companies could be generalized also to software companies e.g. in other European countries. As a side product the extent of the modeling techniques used in software companies could be evaluated in greater accuracy.

Also, the Eclipse Process Framework community [38] continues its work. For example, the process composer software and the wiki-based process content are constantly updating. Although the discussion on the project web forum is not most active, there still seems to be hundreds of readers in each discussion thread.

Chapter 6

Conclusions

This dissertation investigates how modern software process modeling notations and techniques could be used to improve and support daily software development activities. This is accomplished by conducting a series of studies which investigate the software process modeling from several different viewpoints. The goal of this approach is to form wide understanding of the relationship between software process modeling and operational software development work.

In the beginning of the dissertation there is a summary of the history and the current state of software development methodologies. Both the plan-driven processes and the agile methodologies are discussed. After that, theoretical background for software development process modeling is presented concentrating especially on the Software and Systems Process Engineering Metamodel (SPEM) [14]. Also, the most significant scientific articles on software process modeling and its application techniques are briefly presented.

Most of the research effort done for the dissertation is documented in the six accompanied research articles. Each article analyzes one of the research questions presented in Section 1.2. The following conclusions can be drawn from each individual article in contrast to the research questions:

Q1. What are the prerequisites for the process model components that could be re-used in several development scenarios?

> The first article asked whether the SPEM notation can be used to build re-usable process components for different development scenarios. It was found that the SPEM 1.1 was not complete in this respect[1] but the greater obstacle for re-usable components was that the different development methodologies have different terminologies, different interfaces, different pace etc. *This means that different methodologies are not compatible with each other so that interchangeable components could be made.*

[1] SPEM 2.0 is much better when process re-use is considered.

Q2. What issues are important when a process metamodel based tool is implemented?

The second article analyzed whether a modeling tool based on the SPEM 1.1 metamodel could be made. The successful implementation of the tool and later the publication of EPF process tools proved that this can be done. However, the case also showed that *the SPEM 1.1 specification was not completely unambiguous and therefore two SPEM-based tools would probably become slightly different*.

Q3. How do the changes in process modeling languages impact the process improvement and therefore software development work?

The third article analyzed the challenges of contemporary process improvement activities and how the process modeling could be used to improve the situation. The main finding from the development point of view was that *there is still a gap between processes and development work; the modeling cycles are too long and the developers have different views to the processes than the process engineers*. The process modeling was seen as a promising tool for solving these challenges by providing support for dynamic but yet managed changes in the enacted work process.

Q4. How can the process modeling techniques be applied when the resources for the modeling efforts are low?

The fourth article investigated whether fast process modeling is possible and speculated the possible benefits of such modeling approach. This was possible but the model could be used just as a starting point for further process improvement activities. In addition, *an experienced modeler was needed to successfully execute a quick modeling effort*.

Q5. How is process modeling used to share the knowledge of a certain software development domain?

The fifth article gave an example case of how the process modeling techniques are used as a tool for analyzing the game development processes from the pedagogical perspective. *Process models helped to understand that the game development process compares to the product development process, not the actual software development work*. It was also found out that the division of the responsibilities between teachers and students is the key issue that differentiates various game development courses.

Q6. What are the opinions of the software practitioners towards the software process modeling techniques?

The last article investigated opinions of Finnish software practitioners to-

wards the process modeling techniques. *The practitioners saw high potential in modeling techniques, but currently the process modeling did not play a big role in their daily work.*

The wider research theme, to which this work contributes, was set through two general questions: *1) Can the software process modeling and models help software developers in their daily work?* and *2) What are the main mechanisms for utilizing the modeling techniques in the development work?* Each article provided an answer to a specific research question but they all also contributed to the research theme by examining the software process modeling from the development work perspective. By combining the results of the individual studies, the following statements can be made, which helps to understand the set research context better and partially answer the two general questions.

Articles 1, 2, and 4 investigated detailed technical or methodological aspects of the SPEM-based process modeling. The result of all of these articles indicate that *there are ways to use SPEM in a way that also supports the development work directly*. Use of process modeling reduces the feedback cycle and makes the process use more flexible from the development point of view. The fifth article strengthens this conclusion by providing an example of using the modeling techniques to communicate techniques used in a specific branch of software development.

Articles 3 and 6 investigate the process modeling on a more general level. The results of these two studies support each other. In the third article the researchers identify several problems in software process improvement and develop ideas on how to address these challenges by using process modeling techniques. In the sixth article, done a couple of years later, the existence of the challenges and the feasibility of process modeling, a solution is verified from the software practitioners.

As a synthesis of the main research questions of the dissertation, it can be said that there are indeed several ways to use software process modeling techniques to help software developers in their work *in general*. The applications can usually be found in the interface of the process improvement and the software development. Whether the modeling can help the developers in their *daily* work is a more complex question. *The efficient use of modeling would require good skills in process modeling and it is not practical for each software engineer to acquire these skills*. Therefore the process engineers are still required to aid software developers to utilize process modeling techniques even though the techniques are connected more strongly to daily development work.

In principle, *through the process modeling, a connection between the process improvement and the software development could be established*, i.e. the process modeling could act as "glue" between the improvement and the development activities. However, *this would require much better tool support than currently exists*. The process modeling tools should take different use scenarios better into account and provide different views e.g. for process engineers, software developers and managers.

Most of the research work for this dissertation was done between the years 2005 and 2009. During this time period, the agile methodologies have become mainstream in the software development field and have replaced the plan-driven processes in many software companies. Therefore the dissertation concludes with a few remarks on the relevance of the research results.

It can be said that *the raising popularity of the agile development methodologies have reduced the importance of the process modeling notations*, which were originally designed for modeling plan-driven software development processes. The reason for this is that the practitioners of the agile methodologies do not rely on heavy process structures on their daily work. Therefore, they require no extensive process modeling support during the operational development activities, i.e. the need for dynamic process models which guide the work is mild. What is said does not mean that the process modeling cannot be applied in agile context. Beneath the surface many agile methodologies are quite complex and contain many process-like elements like role, work products and different activities. Hence, the process models can be used to illustrate the main principles of a certain methodology and compare the methodologies with each other.

The agile trend will probably continue and other processes around the software development will simplify and become more streamlined. Naturally, the process modeling techniques are still needed for analyzing and improving the high-level processes, and especially the process interfaces, and for disseminating the methodological innovations to other practitioners.

Bibliography

[1] OECD information technology outlook 2010. Technical report, OECD Publishing, 2010.

[2] Tuomo Nikulainen, Jyrki Ali-Yrkkö, and Timo Seppälä. *Softaa koneisiin! Ohjelmisto-osaaminen suomalaisen teollisuuden uudistajana*. The Federation of Finnish Technology Industries, 2011. Executive Summary available in English.

[3] Edsger W. Dijkstra. The humble programmer. In *ACM Turing award lectures*, page 1972. ACM, 2007.

[4] Chaos. The standish group report, Standish Group, 1994.

[5] J. Laurenz Eveleens and Chris Verhoef. The rise and fall of the chaos report figures. *IEEE Software*, 27(1):30–36, 2010.

[6] CMMI for development, version 1.3. Technical report, Software Engineering Institute (SEI), November 2010.

[7] ISO/IEC 15504-1:2004. Technical report, International Organization of Standardization (ISO), 2004.

[8] Rational unified process - best practices for software development teams. Technical report, Rational Software, 1998.

[9] Rational unified process (RUP) homepage. http://www.ibm.com/software/awdtools/rup/. Accessed on September 28th 2012.

[10] Norman E. Fenton and Shari Lawrence Pfleeger. *Software Metrics, A Rigorous & Practical Approach, Second Edition*. PWS Publishing Company, 1997.

[11] Watts S. Humphrey. *Managing the software process*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[12] Leon Osterweil. Software processes are software too. In *Proceedings of the 9th international conference on Software Engineering*, ICSE '87, pages 2–13, Los Alamitos, CA, USA, 1987. IEEE Computer Society Press.

[13] Manifesto for agile software development. http://agilemanifesto.org/. Accessed on September 28th 2012.

[14] Software & systems process engineering meta-model specification, version 2.0. Technical report, Object Management Group OMG, April 2008.

[15] IEEE standard glossary of software engineering terminology. http://dx.doi.org/10.1109/IEEESTD.1990.101064, 1990.

[16] Alain Abran, Pierre Bourque, Robert Dupuis, James Moore, and Leonard Tripp, editors. *Guide to the Software Engineering Body of Knowledge – SWEBOK*. IEEE Press, 2004.

[17] Winston W. Royce. Managing the development of large software systems: Concepts and techniques. In *Proceedings of the IEEE WESTCON (1970)*, 1970.

[18] Herbert D. Benington. Production of Large Computer Programs. *Annals of the History of Computing*, 5(4):350–361, October 1983.

[19] Barry Boehm. A spiral model of software development and enhancement. *SIGSOFT Software Engineering Notes*, 11(4):14–24, 1986.

[20] Sarah A. Sheard. Evolution of the frameworks quagmire. *Computer*, 34(7):96–98, 2001.

[21] Robert McFeeley. IDEAL: A users guide for software process improvement. Cmu/sei-96-hb-001, Software Engineering Institute, 1996.

[22] Kent Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional, 1999.

[23] Jeff Sutherland and Ken Schwaber, editors. *The Scrum Papers: Nuts, Bolts, and Origins of an Agile Process*. 2007. Draft 10/14/2007.

[24] Ken Schwaber. SCRUM development process. In *Proceedings of the 10th Annual ACM Conference on Object Oriented Programming Systems, Languages, and Applications (OOPSLA)*, pages 117–134, 1995.

[25] Stephen R. Palmer and John M. Felsing. *A Practical Guide to Feature-Driven Development*. Prentice Hall, 2002.

[26] David J. Anderson. *Kanban: Successful Evolutionary Change for Your Technology Business*. Blue Hole Press, April 2010.

[27] Don Wells. Extreme programming: A gentle introduction web page. http://www.extremeprogramming.org/. Accessed on September 28th 2012.

[28] Peter H. Feiler and Watts S. Humphrey. Software process development and enactment: Concepts and definitions. In *Second International Conference on the Software Process*, pages 28–40, Berlin, Germany, 1993.

[29] Jacques Lonchamp. A structured conceptual and terminological framework for software process engineering. In *Proceedings of the Second International Conference on the Continuous Software Process Improvement*, pages 41—53, 1993.

[30] Pasquale Armenise, Sergio Bandinelli, Sergio B, Carlo Ghezzi, Angelo Morzenti, and Cefriel Politecnico Di Milano. A survey and assessment of software process representation formalisms. *International Journal of Software Engineering and Knowledge Engineering*, 3:401—426, 1993.

[31] Bill Curtis, Marc I. Kellner, and Jim Over. Process modeling. *Communications of the ACM*, 35:75–90, September 1992.

[32] ARIS platform homepage. http://www.softwareag.com/corporate/products/aris_platform/. Accessed on September 28th 2012.

[33] Jörg Becker, Martin Kugeler, and Michael Rosemann, editors. *Process Management*. Springer, 2003.

[34] Darren C. Atkinson, Daniel C. Weeks, and John Noll. Tool support for iterative software process modeling. *Information and Software Technology*, 49(5):493–514, May 2007.

[35] Unified modeling language: Infrastructure, version 2.0. Technical report, Object Management Group (OMG), May 2005.

[36] Business process model and notation (BPMN), version 2.0. Technical report, Object Management Group (OMG), January 2011.

[37] Business process definition metamodel. Technical report, Object Management Group (OMG), November 2008.

[38] EPF project homepage. http://www.eclipse.org/epf/. Accessed on September 28th 2012.

[39] Ueli Wahli, Majid Irani, Matthew Magee, Ana Negrello, Celio Palma, and Jason Smith. *Rational Business Driven Development for Compliance*. IBM, November 2006.

[40] EPF wiki. http://epf.eclipse.org/. Accessed on September 28th 2012.

[41] Ricardo Balduino. Introduction to OpenUP (open unified process). Technical report, Eclipse Process Framework Project, August 2007.

[42] Rational method composer homepage. http://www.ibm.com/software/awdtools/rmc/. Accessed on September 28th 2012.

[43] David Chappell. What is application lifecycle management. Technical report, Chappell & Associates, December 2008.

[44] Jazz community site. https://jazz.net/. Accessed on September 28th 2012.

[45] Visual studio 2012 manual: Customize process templates. http://msdn.microsoft.com/en-us/library/ms243782.aspx, 2012. Accessed on September 28th 2012.

[46] Martín Soto and Jürgen Münch. Focused identification of process model changes. In *Software Process Dynamics and Agility*, pages 182–194. 2007.

[47] Wil van der Aalst, Vladimir Rubin, Eric Verbeek, Boudewijn van Dongen, Ekkart Kindler, and Christian Günther. Process mining: a two-step approach to balance between underfitting and overfitting. *Software and Systems Modeling*, 9(1):87–111, January 2010.

[48] Chris Jensen and Walt Scacchi. Process modeling across the web information infrastructure. *Software Process: Improvement and Practice*, 10(3):255–272, 2005.

[49] Sergio Bandinelli, Alfonso Fuggetta, Luigi Lavazza, Maurizio Loi, and Gian Pietro Picco. Modeling and improving an industrial software process. *IEEE Transactions on Software Engineering*, 21(5):440–454, 1995.

[50] Alexander L. Wolf and David S. Rosenblum. A study in software process data capture and analysis. In *Proceedings of Second International Conference on the Software Process*, pages 115–124, Berlin, Germany, 1993.

[51] Maurizio Morisio. A methodology to measure the software process. In *In Proceeding of 7th Annual Oregon Workshop on Software Metrics*, Silver Falls, 1995.

[52] James R. Rumbaugh, Michael R. Blaha, William Lorensen, Frederick Eddy, and William Premerlani. *Object-Oriented Modeling and Design*. Prentice-Hall, 1st edition, October 1990.

[53] Maurizio Morisio. Measurement processes are software, too. *Journal of Systems and Software*, 49(1):17–31, December 1999.

[54] Ernst Ellmer and Dieter Merkl. Considerations for an organizational memory in software development. In *Proceedings of the 10th International Software Process Workshop*, pages 60 – 62, Dijon, France, June 1996.

[55] John C. Grundy, John G. Hosking, and Warwick B. Mugridge. Coordinating distributed software development projects with integrated process modelling and enactment environments. In *Proceedings of the 7th Workshop on Enabling Technologies (WETICE '98)*, pages 39–44, Stanford, CA, USA, June 1998.

[56] Ulrike Becker-Kornstaedt. Towards systematic knowledge elicitation for descriptive software process modeling. In *Product Focused Software Process Improvement*, pages 312–325. 2001.

[57] Silvia T. Acuña and Natalia Juristo, editors. *Software Process Modeling*, volume 10. Springer-Verlag, New York, 2005.

[58] Ivan Porres and María Valiente. Process definition and project tracking in model driven engineering. In *Product-Focused Software Process Improvement*, pages 127–141. 2006.

[59] Paula Savolainen, Hanna-Miina Sihvonen, and Jarmo Ahonen. SPI with lightweight software process modeling in a small software company. In *Software Process Improvement*, pages 71–81. 2007.

[60] Oktay Turetken and Onur Demirörs. An approach for decentralized process modeling. In *Software Process Dynamics and Agility*, pages 195–207. 2007.

[61] Oktay Turetken and Onur Demirörs. Process modeling by process owners: A decentralized approach. *Software Process: Improvement and Practice*, 13(1):75–87, 2008.

[62] Onur Demirörs. *A Horizontal Reflective Process Modeling Approach for Managing Change in Software Development Organizations*. PhD thesis, School of Engineering and Applied Science, Southern Methodist University, 1995.

[63] Onur Demirörs and Dennis J. Frailey. A horizontal approach for software process improvement. In *Proceedings of Nineteenth Annual International Computer Software and Applications Conference (COMPSAC 95)*, pages 326 – 331, Dallas, TX, USA, 1995.

[64] A. Gunes Koru and Onur Demirörs. *Advances in Computer and Information Sciences '98: Proceedings of the 13th International Symposium on Computer and Information Sciences*, chapter Decentralized Process Modeling Notations for Horizontal Change Approach, pages 535 – 542. IOS Press, 1998.

[65] Timo Mäkinen and Timo Varkoi. Assessment driven process modeling for software process improvement. In *Proceedings of Portland International Conference on Management of Engineering & Technology (PICMET 2008)*, pages 1570–1575, Cape Town, South Africa, 2008.

[66] Timo Mäkinen. *Towards Assessment Driven Software Process Modeling*. PhD thesis, Tampere University of Technology, Finland, 2010.

[67] Sjaak Brinkkemper. Method engineering: Engineering of information systems development methods and tools. *Information and Software Technology*, 38(4):275–280, 1996.

[68] Antero Järvi and Tuomas Mäkilä. Observations on modeling software processes with SPEM process components. In *Proceedings of The 9th Symposium on Programming Languages and Software Tools*, 2005.

[69] Victor R Basili. The experimental paradigm in software engineering. In *Proceedings of International Workshop of Experimental Software Engineering Issues: Critical Assessment and Future Directions*, volume 706 of *Lecture Notes in Computer Science*, pages 3—12. Springer, 1992.

[70] Wilhelm Hasselbring and Simon Giesecke, editors. *Research Methods in Software Engineering*. GITO Verlag, 2006.

[71] Forrest Shull, Janice Singer, and Dag I.K. Sjøberg, editors. *Guide to Advanced Empirical Software Engineering*. Springer-Verlag London Limited, 2008.

[72] Software process engineering metamodel specification – version 1.1. Technical Report formal/05-01-06, Object Management Group (OMG), January 5th 2005.

[73] Osellus IRIS process author homepage. http://www.osellus.com/IRIS-PA, 2012. Accessed on September 28th 2012.

[74] MSF Team. MSF process model v. 3.1. Technical report, Microsoft Inc., June 2002.

[75] Reidar Conradi, Minh Ngoc Nguyen, Alf Inge Wang, and Chunnian Liu. Planning support to software process evolution. In *Proceedings of Tenth International Conference on Software Engineering and Knowledge Engineering (SEKE'98)*, pages 17—20, 1998.

[76] Vladimir Rubin, Christian Günther, Wil van der Aalst, Ekkart Kindler, Boudewijn van Dongen, and Wilhelm Schäfer. Process mining framework for software processes. In *Software Process Dynamics and Agility*, pages 169–181. 2007.

[77] Harri Hakonen, Tuomas Mäkilä, Jouni Smed, and Andy Best. Learning to make computer games: An academic approach. Technical Report TUCS Technical Reports 899, Turku Centre for Computer Science, 2008.

Antero Järvi and Tuomas Mäkilä

Observations on Modeling Software Processes with SPEM Process Components

In Proceedings of The 9th Symposium on Programming Languages and Software Tools, pages 59–69. University of Tartu, 2005.

Observations on Modeling Software Processes with SPEM Process Components

Antero Järvi and Tuomas Mäkilä

University of Turku,
Department of Information Technology,
FI-20014 University of Turku, Finland

Abstract. OMG's standard for software process modeling (SPEM) contains an element, ProcessComponent, that could be used as a reusable element to assemble end-to-end software processes. However the composition mechanism and the nature of process components is not well defined in SPEM. In addition the organization of process components is not straightforward. In this paper we describe an experiment of using CMMI Process Areas and Rational Unified Process disciplines as a basis for structuring process components. The two approaches produced process components that could not be used together. We conclude that in order to achieve reusable process components, the components must be defined using a common process framework. Further, we give propositions for organizing process components that facilitate component reuse and composition.

1 Introduction

All organizations involved in developing software need to organize, manage and support the development work. Organization's *software development process* ties together all activities and practices addressing this need. Recent years have shown a clear trend of growing emphasis on the software process – the process is thought to be a key factor in ensuring high product quality, achieving accurate time and cost estimates, providing cost efficiency in software development and coordinating large development efforts. In other words, software process has and increasingly important role in achieving and maintaining competitiveness in software business.

Software processes can be defined in many levels of detail, ranging from processes defined implicitly in project management, tools and work practices up to high-ceremony explicitly defined and documented processes. Explicit process description opens the route for *software process engineering* (SPE), consisting of modeling, authoring, tailoring and enacting processes. A natural part of SPE is *software process improvement* (SPI), a deliberate effort to document and modify organizations software processes to increase its competitive strength. However, manual process authoring and tailoring may be impractical. SPI that involves constant change of software processes becomes prohibitively expensive. The overhead cost of SPE and SPI can be brought down by tool support and process automation, both enabled by a process modeling language.

Recently, OMG published a standard for software process modeling, *Software Process Engineering Metamodel* (SPEM) [1], that provides the required formalism for process engineering tools. This paper investigates the suitability and use of SPEM for modeling reusable process components. Section 2 briefly introduces SPEM and related concepts of organizations process engineering. In section 3 we describe the modeling experiment that was carried out. We focus on two frameworks: Capability Maturity Model Integration (CMMI) [2] and Rational Unified Process (RUP) [3]. CMMI is a maturity model, which provides a roadmap for practicing SPI in an organization, RUP is a *de facto* process standard, which defines a software process and its roles, activities, and work products at detailed level. In section 4 the problems of SPEM in this context are summarized and a proposition of process component organization to alleviate the problems is made. This paper is based on work done in the ReProCo-project[1] aiming at constructing reusable process components for various software projects.

2 Background

The background of software process modeling and related technologies are comprehensively reviewed in [4]. Since then the research has somewhat focused around SPEM.

2.1 Software Process Engineering Metamodel (SPEM) overview

The SPEM specification is used for describing a concrete software development process or a family of related software development processes [1]. SPEM is structured as an UML profile and provides a complete OMG Meta-Object Facility (MOF) metamodel. SPEM also defines a notation for its concepts, defined as UML stereotype icons. The main idea of SPEM based process representation is the interplay of three basic elements: *ProcessRoles* that are responsible for and execute *Activities* that consume and produce *WorkProducts*. ProcessRoles, WorkProducts and Activities are all *process definition elements*. Relationships between these basic elements are illustarted in Figure 1.

SPEM defines *LifeCycle*, *Phase* and *Iteration* that are used for dynamic structuring of the process. A LifeCycle defines the ordering of Phases, which in turn can contain Iterations. A Process must have one LifeCycle.

SPEM also defines elements that are meant for organizing other process elements from the viewpoint of process authoring, assembly and reuse. *Packages* are concerned with dividing one or more process descriptions into self-containing parts. These parts can then be placed under configuration and version management and used in assembling and tailoring software development processes. *ProcessComponents* are specializations of Packages. A ProcessComponent is an internally consistent and self-contained chunk of process description that may be

[1] Part of E!3320 project, in co-operation with Genestia Group Inc. - Neoxen Systems and Devera Software Development Center.

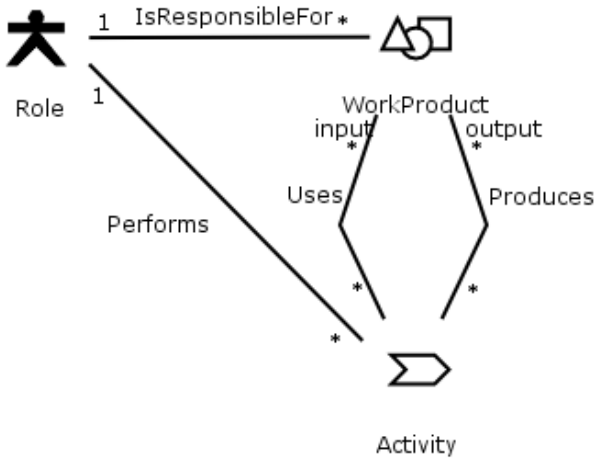

Fig. 1. The core elements of SPEM and their relationships. [1]

reused with other ProcessComponents to assemble a complete process. ProcessComponents can import a non-arbitrary set of process definition elements. The *Discipline* is a specialization of ProcessComponent and is used for representing activities in a common area (corresponding to e.g. Core Workflows in the Unified Process). *Process* finally is a also a specialization of ProcessComponent that is intended to stand alone as a complete end-to-end process.

The assembly of processes is done by composition of ProcessComponents. This requires *unification* of the ProcessComponents. Corresponding output and input WorkProducts must be unified, as well as ProcessRoles and possibly other elements that are used in more than one ProcessComponent. The details of unification are not defined in SPEM.

2.2 OPA, OSSP and software process frameworks

Capability Maturity Model Integration (CMMI) [2] introduces concepts of *Organization's Process Assets* (OPA) library and *Organization's Set of Standard Processes* (OSSP). OPA library is a loosely tied collection of "process assets that are potentially useful to those who are defining, implementing, and managing processes in the organization". OSSP defines a set of processes that "guide all activities in an organization" and "cover the fundamental process elements". OSSP forms a base for organization's project-level tailoring whereas more detailed definitions for process elements can be found from the OPA library. Both OPA Library and OSSP are highly organization specific, depending on the business goals and characteristics of the organization. Similar tailoring mechanisms are also suggested by other major process standards (ISO 12207 [5], RUP [3]).

OSSP is not normally developed from scratch nor simply put together from existing and past projects – instead, so called software process frameworks should be used as guidelines. There are numerous process frameworks available, each focusing on certain types of software development organizations. A partial overview of the software process frameworks and their categorization can be found from the Frameworks Quagmire [6] [7].

2.3 Levels of process modeling

The process modeling language must support modeling at the framework level, at the OSSP level, and at the project level. In Figure 2 these different levels of process modeling are shown with respect to the four-layered organization of SPEM [1]. Layer M2 defines the SPEM language as OMG MOF model. Layer M1 contains process models created with SPEM language, and layer M0 is the enacted software process. Different levels of process modeling are not directly supported by the layering, specifically layer M1 must be divided into two sub-layers: 1) general process modeling and description that models elements of process frameworks, and 2) OSSP that structures modeled process elements so that reusability of these assets is achieved. In 'Project-level tailoring', level M0 performing processes are enacted by selection, composition and tailoring of reusable process components of OSSP M1 sub-layer. These components in turn contain modeled elements from ProcessFramework M1 sub-layer and are tailored to meet organizations need for different types of processes.

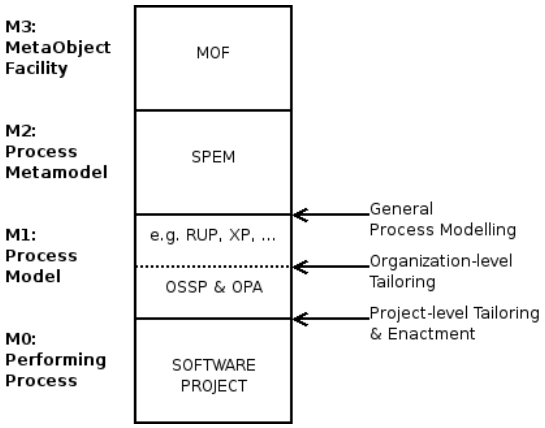

Fig. 2. Process modeling levels contrasted with the OMG SPEM language organization as an UML profile.

Support for this kind of sub-layer organization of organizations process assets is not well included in SPEM standard. The basic SPEM construct for struc-

turing process elements, ProcessComponent, is simply a self-contained Package. SPEM does not define how the underlying process or process framework should be decomposed into components. However, this decomposition largely determines how reusable and how pluggable the process components become. Most process frameworks define a natural decomposition, e.g. Process Areas in CMMI or Disciplines in Unified Process. SPEM standard mentions e.g. Disciplines of Rational Unified Process as possible candidates for ProcessComponents. It appears, however, that this proposed framework decomposition does not yield very reusable process components. The whole issue of framework decomposition to define conceptual organization of the OSSP is far more complicated than we and apparently the SPEM standard anticipated. We will get back to this issue in Section 4.

3 The modeling experiment

3.1 Modeling CMMI Process Areas as process components

The original plan was to model CMMI *Requirement Management* (REQM) and *Requirements Development* (RD) Process Areas [2] as two process components. CMMI was chosen because it structures software process in widely adopted and accepted way. The reasoning was that with this kind of approach, reusable process component 'stubs' would form the OSSP. The conceptual organization of the OSSP would follow CMMI Process Areas, thus CMMI compliance of the tailored process would be straightforward to show. The contents for these 'stubs', the OPA of the organization, would be obtained from other software process frameworks and the components would be mutually compatible as long as they comply to corresponding requirements of CMMI. This would have enabled assembling a software process from several different process frameworks and using most suitable parts of each.

The visible interface of a process component in SPEM is defined by components inputs and outputs. Thus, in order to model Process Areas as process components, we analyzed what process elements were consumed or produced by a particular Process Area, i.e. its inputs and outputs. Modeling was done at the CMMI Specific Practice level. Specific Practices are brief statements on what is required from a software process in order be compliant with regard of a particular Process Area. Special Practices essentially form the backbone of each Process Area. It became apparent that Specific Practices, despite their activity-like nature, describe only the minimum set of Process Areas requirements. In order to construct a process component that could be used in a real software process, Process Areas requirements must be complemented with a software process framework that defines the missing detailed content of the process model, in particular the exact set of inputs and outputs of the Process Area. This suggests that process components can not be modeled based solely on the CMMI Specific Practices. The textual guidelines and work product recommendations of CMMI provide some of the missing content of the process component, but fall short on providing detailed interface descriptions. The interface detail must

be obtained from another software process framework. This hints that CMMI Process Area based components will not have high reusability and can not be used to form an OSSP.

3.2 Integrating RUP and CMMI process components

To set the underlying software process methodology and provide detailed content for CMMI Process Area based process components, we chose commercial *Rational Unified Process* framework (RUP) [3]. The reason for this choice was that the framework is widely adopted, there is enough reference material about the framework available and it is based on the more academic Unified Process framework [8]. Also RUP involves many of the generally accepted properties of modern software process: iterative, incremental, use-case driven and architecture centric to mention a few [8].

We began the modeling experiment by mapping relevant parts of RUP methodology into CMMI Process Area based process components. Because RUP and CMMI are structured differently and CMMI generally speaking operates at a higher level of abstraction, process element level correspondences between these two frameworks had to be established. This required interpretation of the elements roles in the two frameworks, since directly corresponding elements could not be always found. In CMMI, Specific Practices were chosen as elements that were mapped into RUP activities and work products. Figure 3 illustrates a mapping of CMMI REQM Process Area Specific Practices into RUP. The mapping shows that the practices of the CMMI REQM Process Area do not map into a single RUP discipline, even though only the Requirements discipline should contain requirements related activities in RUP. It should be noted that only Special Practices 1.1. and 1.2. are shown in Figure 3 The remaining three Special Practices of REQM are similarly scattered throughout the RUP disciplines. Inverse mapping from the RUP Requirements discipline to REQM and RD Process Areas strengthened this finding: only part of the practices in RUP Requirements discipline could be mapped back into CMMI REQM or RD.

4 Findings

In our experiment we found two different impediments of full scale use of SPEM process components as a basis for an OSSP: 1) shortcomings of SPEM standard regarding process components and 2) lack of established practices of organizing process components at conceptual level.

4.1 SPEM process component shortcomings

SPEM language is well-designed to model basic process definition elements. However the definition of ProcessComponent is ambiguous: requirement of self-contained components means that there must be no 'RefersTo' dependencies

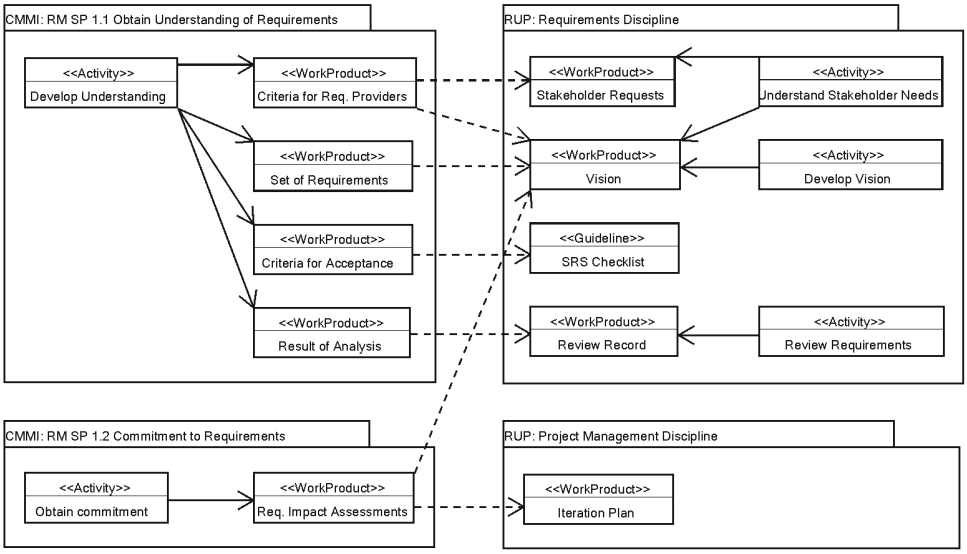

Fig. 3. An exemplar mapping of CMMI practices to RUP activities illustrate how these two frameworks structure the requirements management differently.

from within the component to elements not within the component. Other dependency types, e.g. 'Import' are allowed. The semantics of 'RefersTo' and 'Import' can be interpreted in many ways. Figure 4 shows one possible use of the dependencies. [1] The absence of any examples in the standard illustrating the intended use of these dependencies will lead to different ways of composing process models from process components in different organizations.

Composition of process components is done by *unification*. SPEM states that at least output work products of component P1 and input work products of component P2 must be made identical in order to combine P1 and P2. In addition, according to SPEM other elements may possibly be also unified, such as ProcessRoles, Templates, and so on. SPEM suggests that composition of components could only be fully automated if they originate from a common family, so that unification could be automated. Otherwise the unification would involve human intervention consisting re-writing of the elements [1]. However, SPEM does not provide any explicit support for component composition and unification – the issue is left open.

From the viewpoint of process component reusability, the assembly mechanism may allow too much variation and jeopardize component portability between two OSSPs. Also tool independency is compromised as tool vendors may interpret SPEM standard composition mechanisms differently.

4.2 Conceptual organization of process components

Process components could be formed from process frameworks by using their existing organization to identify components. As an example, making process components out of RUP disciplines is certainly a natural decomposition. However, looking closer at how one discipline in RUP interacts with other disciplines in RUP, we clearly see that the composition interface of the component will become very complex. The work carried out in a discipline, expressed as an activity flow, is connected to many other disciplines in all four phases of RUP. Further, phases contain iterations, and many disciplines are active during one iteration. Roles, tools, guidances and templates are shared between disciplines, and continuous, integral workflows of a single worker continue seamlessly from one discipline to another. Expressing such a complicated and versatile relationship between process components, with the simple support that SPEM offers, is challenging.

It is questionable whether these kind of large process components are useful at all in an OSSP. Achieving reusability in general requires that the reused component has a well defined task or responsibility, and has a simple interface - otherwise it will not be pluggable without considerable manual tailoring. It may very well be the case that these problems originate from the foundational level, not specifically from SPEM. Process components that are formed from RUP Disciplines or CMMI Process Areas appear not to be reusable or even usable. The only situation where the composition is gets manageable is a simple waterfall type sequencing where process components represent the phases of the life-cycle and are connected via major mile-stone work products. Note that in this case disciplines (e.g. requirements, design, testing) in fact coincide with the phases.

The other possible approach to create process components is to begin by defining process elements of a process framework: work products, roles, guidances etc., and model the dependencies between these basic elements. These elements have high reusability and process independence, and can thus be used to form the OPA. SPEM Package is a natural choice to model this library of stable process elements. The OSSP has the role of modeling the dynamic part of the process. It consists of rather small process components, packaging together a cohesive flow of activities, the participating roles, guidances and tool mentors, templates, and related work products. Most of this content could be taken from the OPA using 'Import' dependency. Elements that remain internal to only one process component need not be added to OPA. The main reason for this arrangement is the unification of process elements shared by several process components.

'RefersTo' dependency is used in the OPA to indicate which elements must be imported together due to dependencies. The situation is illustrated in Figure 4. For instance, importing 'Use Case Model' necessitates also importing 'Use Case'. 'Impacts' dependency means that changing one element has a potential effect on the other. All of these dependencies should be modeled in the OPA only if they are valid in all possible process components where they will be used,

e.g. 'Software Requirements Specification' will impact 'Software Architecture' in any imaginable process, thus the dependency is reusable.

4.3 The proposed organisation of an OSSP based on process components

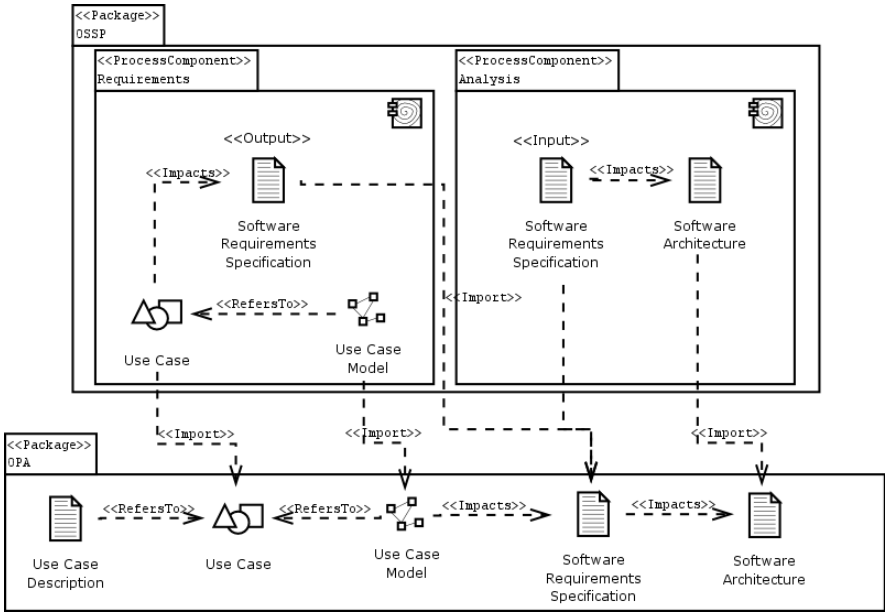

Fig. 4. The proposed organization process assets into OPA that contains low level reusable process elements, and OSSP that is the target of process authoring and contains goal-oriented small sized process components that package together the flow of activities and all related process elements.

Some of the process components in OSSP are going to be reusable and some not. Examples of reusable components are 'DefineSystemScope' or 'CreateCandidateArchitecture'. These components could have different versions for different types of projects, e.g. a light version of 'DefineSystemScope' for projects in familiar domain and a more detailed and more comprehensive one for new domains or multi stakeholder situations. Note that we propose to create process components based on clearly identified goals during the development work, rather than based on the structure of the process framework. As stated these components have variable degree of reusability. However this approach facilitates process authoring intermediate goals of development are a very natural view in process authoring.

Process authoring would then consist of dividing the project into clear intermediate goals, looking for reusable components for each identified goal, selecting the most suitable component depending on the particular project, tailoring or modifying the selected components, creating new components using elements from OPA where none could be found in the OSSP, and finally controlling the input and output work products to make sure that the components fit together. All of this involves a lot of manual work, but it seems that when practical issues are taken into account, more automation in process authoring is very difficult to achieve. The OPA of course gives high degree of reusability of process elements.

5 Discussion

In this ongoing work we aim at understanding the many faceted problem of decomposing software process models into reusable and easily pluggable components. Our original idea, inspired by guidance of SPEM standard, of creating large components using the structure of existing process frameworks seems to yield unmanageable process authoring task and no real reuse. Thus we started looking at smaller scale components and new ways of delineating process components. Some promising ideas have emerged, as reported above. However, there are many forces involved that must be taken into account. Processes with different life-cycle models and other process attributes probably require different modeling philosophy, e.g. document-driven methodologies using the traditional waterfall life-cycle model can be modeled using SPEM type language and process components quite easily while evolutionary methodologies with iterative activity flows are more demanding. Agile methodologies probably would require something very different because they usually lack clear intermediate work products and therefore process component interfaces could not be formed. The control relies heavily on inter-person communication and seamless integration of many activities by the developer into one whole. Establishing artificial interfaces and work products to implement the interfaces would be catastrophic in this environment. We believe that an agile process can not be decomposed at all into process components.

The traditional goals for OSSP are not easily achieved. The issue must be addressed from many viewpoints: process analysis, design, assessment, process component reuse, process improvement, process authoring, publishing, enactment and authoring. The issues are organization dependent; at least organizations size and business context affect the OSSP.

Common understanding on these issues must be achieved: academic community, tool vendors and standardization work all have a role here. SPEM standard version 1.1. has not gained widespread acceptance, possibly due to insufficient guidance on how it should be put into use. The standardization work for SPEM version 2.0. [9] is ongoing and hopefully succeeds better on these issues that have immense practical influence; is process modeling only going to be used for describing graphically the processes of the organization, or will we be creating a component oriented OSSP that is the target of SPI and can yield a tailor made

process for each project, possibly supporting third party process component markets.

References

1. Object Management Group. *Software Process Engineering Metamodel Specification - Version 1.1*, January 5 2005. formal/05-01-06.
2. CMMI Product Team. Cmmi for systems engineering and software engineering (cmmi-se/sw, v1.1) - staged representation. Technical Report CMU/SEI-2002-TR-002, Software Engineering Institute, Pittsburgh, PA, USA, December 2001.
3. IBM. Ibm rational unified process. Software Product, 2005.
4. D. Wastell J.C. Derniame, B.A. Kaba, editor. *Software Process, Lecture Notes in Computer Science*. Springer-Verlag, 1999.
5. ISO/IEC, Geneva, Switzerland. *ISO/IEC 12207, Information technology - Software life cycle processes*, August 1 1995. ISO/IEC 12207:1995.
6. The frameworks quagmire. http://www.software.org/quagmire/. Accessed on May 30 2005.
7. Sarah A. Sheard. The framework quaqmire, a brief look. *Crosstalk*, September 1997.
8. Ivar Jacobson, Grady Booch, and James Rumbaugh. *The Unified Software Development Process*. Addison-Wesley Professional, February 4 1999.
9. Object Management Group. *Software Process Engineering Metamodel (SPEM) 2.0 - Request For Proposal*, November 4 2004. ad/2004-11-04.

Tuomas Mäkilä and Antero Järvi

Spemmet – A Tool for Modeling Software Processes with SPEM

In Proceedings of the 9th International Conference on Information Systems Implementation and Modelling, ISIM 06, pages 87–94. MARQ, 2006.

Spemmet - A Tool for Modeling Software Processes with SPEM

Tuomas Mäkilä *
tuomas.makila@it.utu.fi

Antero Järvi*
antero.jarvi@it.utu.fi

Abstract: The software development process has many unique attributes and therefore the modeling language should be designed for this particular task. The emerging Software Process Engineering Metamodel (SPEM) modeling standard from the Object Management Group (OMG) offers one solution of modeling software processes. In this paper the Software Process Engineering Metamodel is presented in brief and challenges on implementing a SPEM based modeling tool are discussed. Especially the Spemmet process modeling tool implemented by the authors is presented and analyzed.

Keywords: software process modeling, SPEM

1 Introduction

Business process modeling is a tool in information system development for understanding the dynamic behavior of the system context. Similarly, the dynamic complexity of information system development is managed by modeling the underlying development processes. However, these processes have different characteristics than business processes and thus can not be modeled well with the present business process modeling languages e.g. traditional flow chart notation, *Business Process Modeling Notation (BPMN)* [1], *Integrated Definition Methods (IDEF)* [2], or *Event-Process Chains (EPC)* [3].

The development of information systems is a highly complicated task, involving coordination of actions of many people working under uncertainty, managing constant change, and at the same time using resources efficiently to meet tight deadlines. Software processes need to balance predictability and efficiency with flexibility of operation and creativity. The stable elements in software development are the work products; how created value is captured to work products during the project is less volatile than the activities and their order of execution. Therefore well-known business process modeling languages based on activity oriented modeling fail to express the required flexibility in modeling processes characterized by uncertainty and creativity.

Process modeling languages that are built around the work products, so called entity-based modeling languages, are more suitable for modeling software development processes. One such language is *the Software Process Engineering Metamodel (SPEM)* from *the Object Management Group (OMG)* [4]. The SPEM is based on *the OMG Meta Object Facility (MOF)* [5]. The purpose of the SPEM is to become the standard software process modeling language. The current version of the SPEM is 1.1 which was released in January 2005. In a few years there will be a major revision of the standard [6].

The research of software process specific modeling languages has been mild during recent few years. Some work on the topic has been done. García et al. have developed metrics for

* University of Turku, Department of Information Technology, FI-20014 Turku, Finland

evaluating maintainability of the software process models [7][8]. They have used the SPEM as a primary modeling language. Franch and Ribó have investigated ways to enhance reuse with appropriate modeling conventions [9]. They have developed own UML-based modeling language PROMENADE to support the reuse.

Besides the academic research there are some commercial tools, that support software process modeling techniques. Key practitioners in this field are IBM and Osellus. IBM provides a software process modeling tool that supports their Rational Unified Process framework [10]. Osellus have developed the IRIS product family which is meant for process authoring and enactment [11]. The IRIS product family has been built on the SPEM modeling language.

One interesting project related to the topic is the freshly started Eclipse Process Framework Project (EPF) [12]. The goal of the project is to provide process authoring tool framework and process content for different development needs. The work is based on the open Eclipse platform.

This report is based on the ongoing ReProCo research project, that investigates how software development should be supported by process modeling. The central issues are the type of modeled process content, its organization into process components, and the mechanisms of process content reuse. The aim is to provide flexible and customizable process models that can meet the high variety of the process needs, which are typically found in software companies. In addition these process models should be light to define and maintain. The concepts of the process content organization into reusable process libraries are presented in another paper [13].

This paper will focus on describing technological elements of software process modeling: the modeling language and a modeling tool based on the language. The context of software process modeling should still be kept in mind. The process models should be organized into reusable process libraries, which can be selected, tailored, and enacted by software companies based on the organizational and the project needs.

2 Software Process Engineering Metamodel (SPEM)

2.1 SPEM standard

SPEM standard is closely related to *the UML* standard, since they both are based on the MOF specification. The SPEM 1.1 standard is actually built onto the SPEM Foundations package which is a subset of the UML 1.4 [14]. Basic element, relationship, and package structures are defined in this Foundations package. All other SPEM elements are defined in relation to the elements of the Foundations package through inheritance. Because there is a close connection between the SPEM and the UML, it is natural that the SPEM is also defined as a UML profile in the SPEM documentation [4, Chapter 11].

Although the SPEM notation resembles the UML, it has otherwise quite different structure. At the conceptual level, the main idea of SPEM based process is the interplay of three basic elements: *Process Roles* that are responsible for and execute *Activities* that consume and produce *Work Products*. Roles, Work Products, and Activities are all process definition elements. This is illustrated in Figure 1.

SPEM defines elements *Life Cycle*, *Phase* and *Iteration* that are used to model the dynamic structure of the process. A Life Cycle defines the order of Phases, which in turn can contain Iterations. A Process must have exactly one Life Cycle.

SPEM also defines elements that are meant for organizing other process elements from the viewpoint of process authoring, assembly and reuse. The purpose of *Packages* is to divide

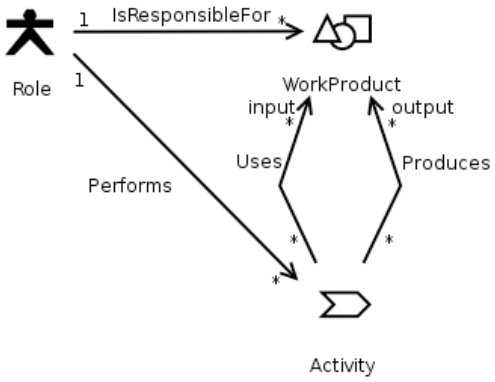

Fig. 1: The core concept of SPEM process modeling is a triangle formed by Role, Work Product and Activity.

process descriptions into self-containing parts. These parts can then be placed under configuration and version management and used for assembling and tailoring software development processes. *Process Components* are specializations of Packages. A Process Component is an internally consistent and self-contained chunk of process descriptions that may be reused with other Process Components to assemble complete processes. Process Components can import a non-arbitrary set of process definition elements. The *Discipline* is a specialization of the Process Component and is used to represent activities within a common process area, such as design, implementation or testing (corresponding to e.g. Core Work-flows in the Unified Process [15]). *Process* itself is also a specialization of the Process Component, that is intended to stand alone as a complete end-to-end process.

The assembly of processes is done by composition of Process Components. This requires unification of the Process Components. Corresponding output and input Work Products must be unified, as well as Process Roles and possibly other elements that are used in more than one Process Component. The details of unification are not defined in SPEM.

2.2 The scope of process modeling

Process modeling can be used in many different scales; at its simplest, an existing software process in a company would be modeled for clearer presentation and easier delivery. On the other extreme, a large company might manage a library of software process content and new end-to-end processes would be created by combining and customizing process components, either internally developed or third party components. It is evident, that the more complex applications of process modeling and reuse require more sophisticated mechanisms for organizing process content. Unfortunately SPEM 1.1 cannot provide these mechanisms. However, the core ideas of SPEM modeling are unlikely to change. For this reason we are currently targeting for simple applications of process modeling; having only internally developed process content, a clear separation of static and dynamic process content, and simple mechanisms of process assembly [13]. As the SPEM standard evolves, these basic modeling concepts should remain valid.

3 Spemmet Process Modeling Tool

In this section we present a SPEM based modeling tool called *Spemmet*. The development of the tool begun at the early stage of the ReProCo project when we had to form a prototype process component model using the SPEM modeling language.

Before the decision of the own SPEM modeling tool was made, several readily available tools were evaluated. The evaluated tools can be divided into four categories: Basic drawing tools, UML tools, XML tools, and SPEM specific editors. Drawing tools, UML tools, and SPEM tools concentrated on diagram drawing, whereas we needed more efficient means to bind textual data (e.g. tailoring guidelines) to process elements. These tools also had limited modeling support for our needs. XML tools were evaluated in order to exploit the SPEM XMI DTD (XML Metadata Interchange Document Type Definition) provided by OMG [16] for manual data input. It soon become apparent that the DTD was meant only for exchanging information between modeling programs and could not be used as a base for forms for manual data entry [1].

When the Spemmet development started we made a couple of key design decisions. Firstly, we wanted to make sure that the tool would not restrict our modeling options. Of course, the tool should offer all necessary elements for modeling software processes and support the modeling efforts in order to make the modeling as easy as possible. It is equally important that the tool is flexible and allows the user to use different modeling techniques, because modeling needs change depending on the modeled process. Modeling tool must also allow user to experiment with different modeling techniques since there are no established software process modeling guidelines available.

Second design decision was to omit graphical representation of the process model and allow user to input and output data into the model only in textual form. It was apparent that the textual list of the attributes of a process element was easy to comprehend. Actually it would have been difficult to clearly express all attributes of one element in a graphical diagram. After the process element data is entered into machine-readable form, it is relatively easy to generate various graphical and textual views from this data in future version of the tool. This is an important design aspect, because in practice different stakeholders have different purposes of use for the same process model, and logical way of taking these varying and currently unanticipated requirements into consideration is to be able to offer multiple views to the model.

The implementation of the Spemmet tool was quite straightforward. It was developed on a web platform and used a shared data storage. The idea was that the tool would be available on any workstation without installation and multiple users could use it simultaneous. The overview of the architecture of the tool can been seen in Figure 2.

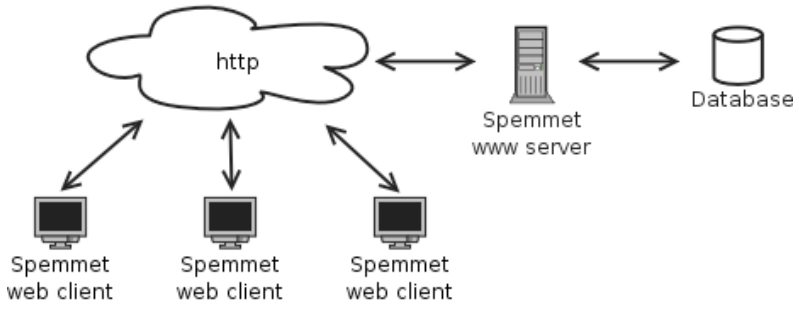

Fig. 2: Overview of the Spemmet architecture. Spemmet web clients communicate with the Spemmet server through the http protocol. The model data is stored in the database.

The most difficult task during the implementation was to transform the conceptual model of

[1] This was an anticipated outcome, because the original purpose of the XMI definitions is to enable metadata interchange between modeling tools.

the SPEM to the actual data structure definitions used by the tool. This was a consequence of the complexity of the SPEM standard specification. Some parts of the standard were ambiguous or simply too complex to be feasible in this kind of tool prototype. Good example of the latter is the state machine described in the SPEM Foundations package, which might be good for the UML 1.4 but is too complex for the SPEM.

Luckily the UML Profile definition of the SPEM was easier to understand and helped resolving some of the metamodel ambiguities. Some shortcuts were made with the most complex parts of the SPEM metamodel. However, the overall data structure of the Spemmet tool was kept as compatible with the SPEM standard as possible.

The most important classes of the Spemmet tool data model are presented in Figure 3. Only structures used for modeling static process elements were implemented in this first version of the Spemmet tool. Therefore the basic conceptual triangle of the SPEM presented in Figure 1 formed the heart of the class structure of the Spemmet tool. Guidance and Package classes were adopted to support the process reuse: The Guidance class enabled documentation and tailoring guidelines integration into the process models and the Package class made possible to logically group process model elements. Figure 3 also shows that all of the SPEM classes are inherited from the Model Element class.

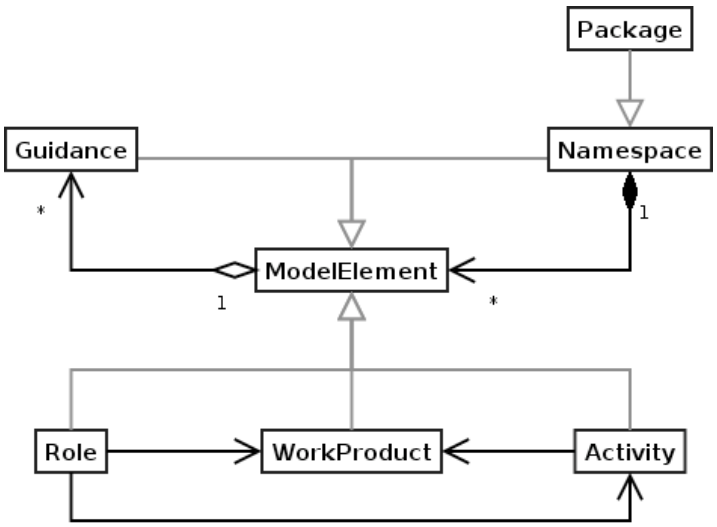

Fig. 3: Core classes from the SPEM implemented in the Spemmet tool.

The first version of the Spemmet tool was successfully implemented and included all the features that were needed in our modeling project. The basic architecture works fine and it is possible to use modeling tool with a web browser through any workstation in our intranet. Multiple users are able to model with the tool simultaneously, although there are still some scalability issues and no user management in this first version.

The most important feature for flexible and practical process modeling styles is the versatile linking between the process elements. All basic elements can be linked together using multiple relationship types. Association, dependency, and inheritance, which are familiar from the UML standard, are all supported. The user can to browse elements by following these relationship links. Together with the basic package structure the user can navigate through the process model easily and in appropriate manner.

The user interface of the Spemmet tool is naturally quite simple, because in the SPEM all process elements are inherited from the same parent class (i.e. Model Element class). Therefore the user can browse and edit all process elements in consistent one-screen view. This simple design allowed us also to implement an html exporter, which is a great help when we have to share snapshots of the model with other stakeholders. A screen shot of the Spemmet user interface can be seen in Figure 4.

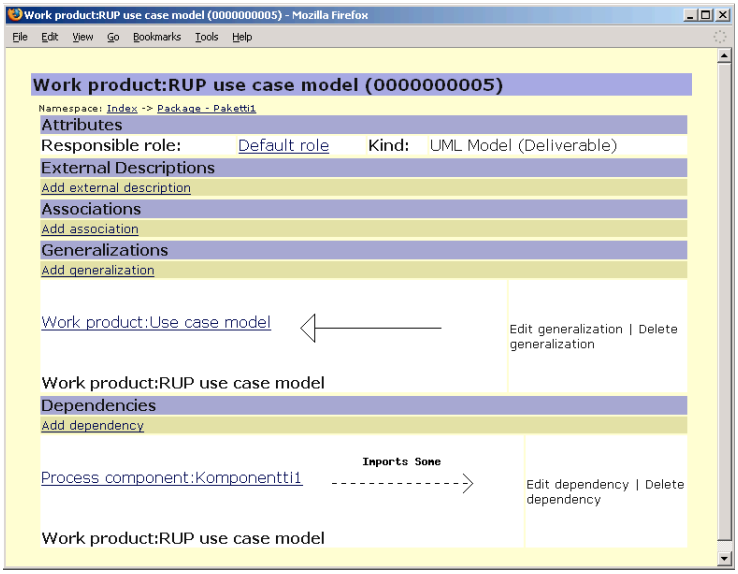

Fig. 4: Screen shot of an element view of the Spemmet tool.

4 Further Work

The Spemmet tool is our first attempt to get practical understanding of the SPEM based process modeling. At the time being the Spemmet tool includes only static process elements of the SPEM standard. The tool must be further developed so that the process dynamics mentioned at the introduction section can be modeled. Especially elements in Process Life Cycle package (i.e. Phase, Life Cycle, Iteration) are important, because these elements are used to express temporal relationships between activities. Modeling of the dynamic structures can however be based on elements that are already included in the tool. For example an activity can be automatically connected to the compatible activities using the input and output work products and their states.

Our work continues on two mutually supporting tracks: first we are carrying out an empirical investigation on the needs and applications of process modeling in software companies. Together with theoretical research on software process modeling, the empirical study results guide the second track, the development of Spemmet tool. The goal is to be able to meet the practical modeling needs in a wide range of companies, and support them with process modeling tools.

The standardization work of SPEM is ongoing; the version 2.0 [6] will be released with in a few years. This of course introduces some uncertainty, but on the other hand, it gives us time to understand the potential and sound ways of applying process modeling in companies.

5 Conclusions

Our experiment with the Spemmet tool shows that it is possible to develop a working process model tool in a relatively short time using the SPEM process modeling standard. The SPEM standard is certainly hard to follow in places and partially too complex. Therefore some shortcuts have to be made at least for this kind of lightweight modeling tool.

The tool was successfully used to model parts of a process framework during the ReProCo project. The modeling was quicker than with tools that were not designed for process modeling, because the tool supported directly SPEM metamodel concepts. The communication about the model itself was also easier with the other stakeholders of the project, because all necessary information of a process element was collected into one place.

The clear benefit from the SPEM specific modeling tool was that the tool makes modeling easier and enables quick modifications. Models made with the software process specific tools seem to be more informative than models "hacked" with other modeling tools.

The greatest challenge with the SPEM modeling standard is to promote its use both in the academic world and in the software industry. Also, the current SPEM 1.1 version is immature, and this obstacle will probably be removed by the forthcoming SPEM 2.0. The only way we see to increase the adoption of SPEM is to develop working SPEM based modeling tools that will help learning modeling techniques and experimenting with the SPEM.

This article is based on work done during the ReProCo research project (Sub-project of the E!3320 project) in co-operation with Genestia Group Inc. - Neoxen Systems and Devera Software Development Center.

Bibliography

1. *Business Process Modeling Notation Specification - Final Adopted Specification.* Object Management Group, 2006. dtc/06-02-01.
2. *Integrated Definition Methods Home Page.* `http://www.idef.com/`, Knowledge Based Systems Inc. Accessed on March 16 2006.
3. Becker et al.: *Process Management - A Guide for the Design of Business Processes.* Springer, 2003.
4. *Software Process Engineering Metamodel Specification - Version 1.1.* Object Management Group, 2005. formal/05-01-06.
5. *Meta Object Facility (MOF) Specification - Version 1.3.* Object Management Group, 2000. formal/00-04-03.
6. *Software Process Engineering Metamodel (SPEM) 2.0 - Request For Proposal.* Object Management Group, 2004. ad/2004-11-04.
7. García et al.: *Integrated Measurement for the Evaluation and Improvement of Software Processes.* Lecture Notes in Computer Science, Volume 2786, Springer, 2003. pp 94 – 111.
8. García et al.: *Definition and Empirical Validation of Metrics for Software Process Models.* Lecture Notes in Computer Science, Volume 3009, Springer, 2004. pp 146 – 158.
9. Franch and Ribó: *A UML-Based Approach to Enhance Reuse within Process Technology.* Lecture Notes in Computer Science, Volume 2786, Springer, 2003. pp 74 – 93.
10. *Rational Unified Process Home Page.* `http://www.ibm.com/software/awdtools/rup/`, IBM. Accessed on March 16 2006.
11. *Osellus Home Page.* `http://www.osellus.com/`. Accessed on March 16 2006.
12. *Eclipse Process Framework Project Home Page.* `http://www.eclipse.org/epf/`, The Eclipse Foundation. Accessed on March 16 2006.

13. Antero Järvi and Tuomas Mäkilä: *Observations on Modeling Software Processes with SPEM Process Components.* Proceedings of The 9th Symposium on Programming Languages and Software Tools, Tartu, Estonia, 2005.
14. *OMG Unified Modeling Language Specification - Version 1.4.* Object Management Group, 2001. formal/01-09-67.
15. Jacobson et al.: *The Unified Software Development Process.* Addison-Wesley Professional, 1999.
16. *XMI DTD for SPEM 1.0.* Object Management Group, 2003. formal/02-11-14.

Antero Järvi, Tuomas Mäkilä and Harri Hakonen

Changing Role of SPI – Opportunities and Challenges of Process Modeling

In Proceedings of the 13th European Conference on Software Process Improvement (EuroSPI), pages 135–146. Springer-Verlag, 2006.

Tuomas Mäkilä, Antero Järvi and Luka Milovanov

Light-weight Approach for Software Process Modeling – A Case Study

In Proceedings of New Exploratory Technologies 2007, pages 12–16. Korea Electronic Forum, 2007.

Editorial note: This dissertation contains a slightly adapted version of the original article where a broken layout of the proceedings is fixed.

Light-weight Approach for Software Process Modeling – A Case Study

Tuomas Mäkilä[1], Antero Järvi[1] and Luka Milovanov[2]

[1]University of Turku, Department of Information Technology, Turku, Finland
[2]Plenware Oy Ltd., Embedded Systems, Turku, Finland
tuomas.makila@utu.fi, antero.jarvi@utu.fi, luka.milovanov@plenware.fi

Abstract

Formal software process modeling languages like Software Process Engineering Metamodel (SPEM) together with proper tool support, provide an efficient way to maintain and modify process definition content. In order to adopt a process modeling technology, an organization has to model its current processes. This is not a straightforward task since existing process definitions may be nonexistent, incomplete or incompatible with concepts of a modeling language, and the actual process may vary considerably from the documented one. In this paper, we describe an approach for conducting the as-is process modeling work. As a proof of concept we present a case study on process modeling carried out in the Gaudí Software Factory.

Keywords

Software Process Modeling, SPEM, EPF, Agile Methods

INTRODUCTION

Software process modeling technology is advancing rapidly and gaining wider acceptance in the industry [1,2]. The Software Process Engineering Metamodel (SPEM) standard of the OMG has defined a unified way to model software development processes. Current SPEM standard is at version 1.1 [3], version 2.0 [2] is not released at the time of writing this paper, but has reached final adopted specification phase and will be released in the very near future. The success and penetration of a standard is crucially dependent on the availability of mature tools. In this research we have used an open-source process authoring tool provided by the Eclipse Process Framework (EPF) project [4]. The EPF Composer tool supports all essential SPEM 2.0 modeling mechanisms, although it is not fully SPEM compliant. Also commercial SPEM based process modeling tools are available.

Formal process modeling with suitable tools enhances existing process practices by reducing process management costs and increasing process flexibility, and creates new innovative ways of utilizing processes for supporting development work and project management [5]. A common requirement for realizing these benefits is bringing processes closer to the developers' world and thus enabling to narrow the gab between the defined process and the actual process followed by the developers.

Depending on how the process models will be used in the organization, the required accuracy, level of detail and frequency of modeling work varies. However, the more often we refine or supplement the models, the better match we can sustain between the models and the actual process. Thus modeling has to be seen as an ongoing activity, not as a single project which creates an everlasting process definition. There is a need for a modeling approach that is lightweight and can be executed in parallel with the development work.

In this article we propose an approach for modeling the software development processes using the SPEM language. Prerequisite for the modeling is the existence of explicit or implicit process documentation which can be incomplete, outdated or incompatible with the SPEM language.

MODELING APPROACH

Although there is active research on process modeling, the focus is on the models. Research on the methods of creating the models is rare. Schwegmann and Laske present a procedure for as-is business modeling in [6]. The procedure includes following high-level activities: preparation of as-is

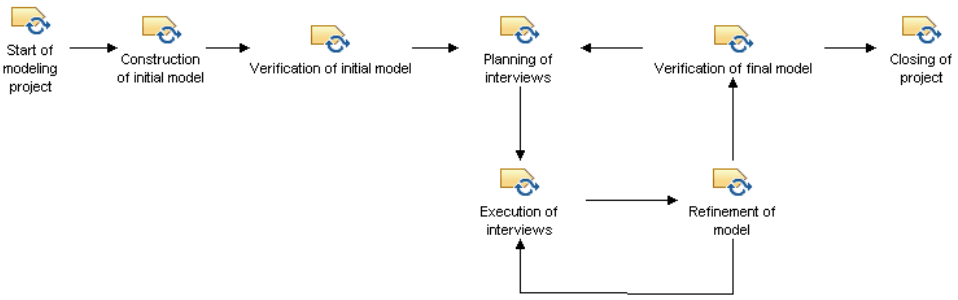

Figure 1 Illustration of the modeling approach

modeling, identification and prioritizing of problem areas, collection and documentation of as-is models and consolidation of as-is models. Kellner and Hansen [7] describe a process modeling project for SPI purpose. Although their focus is on the contents of produced models, some useful information on how the actual modeling should be done can also be found.

Requirements for Approach

To define the concept of light-weight modeling, we set the following requirements for our proposed modeling approach:

1. *Fast modeling*: the development processes shall be modeled effectively so that the focus can be kept on the use of the models rather than on the definition of the models.
2. *Low interference with development*: the modeling shall not affect the normal development activities more than is absolutely necessary. This makes it possible to model the software development process in parallel with the on-going development projects.
3. *No need for extensive consultation*: the modeling shall be possible to execute by following the written modeling approach description and general modeling guidelines. This makes industrial adoption more plausible and takes also small organizations into consideration.
4. *Technology independence*: modeling shall be possible for all kinds of software development processes, regardless of the methods they are based on.

The modeling of processes can be a complex task and consume a large amount of resources if done thoroughly [6,7]. Therefore, we must also accept some process simplifications in order to speed up the modeling and reduce the interference with development projects. On the other hand, based on the findings of Kellner and Hansen, a too low number of interview iterations will reduce the accuracy of the final process model [7][p.23]. Their findings also suggest that processes which have been executed repeatedly are easier to model.

Our modeling approach suites well for the cases where the process is well-established and a consensus of work practices exists within the development organization. In this case, the number of costly interview iterations can be kept minimal while still yielding sufficient accuracy of the model.

Modeling Process

Our modeling process consists of seven steps which are executed sequentially. The high-level overview of the approach is illustrated in Figure 1.

Start of modeling project. Plan the modeling project and communicate the plan to all participants. There should be one lead modeler who is responsible for the modeling work. The lead modeler should have experience in software process modeling. Also the purpose of the final model should be specified, because it defines the level of detail of the model.

Construction of initial model. The initial process model is constructed based on the existing process documentation. The initial model acts as a starting point for modeling the current process. It can also work as a reference when comparing the differences between the documented and actual processes. If no explicit process documentation exists, implicit process documentation, i.e. guidelines and templates of the development organization, should be used.

Verification of initial model (Optional). Authors of the existing documentation, namely *process experts*, should verify that the model is in line with the existing process documentation and that the modelers have understood the documentation right. The model should be refined during the verification meeting, so that no additional meetings are needed. Time-consuming, detailed changes can be done after the verification.

Planning of interviews. The interviews serve several purposes. Firstly, to clarify unclear parts of the process model. Secondly, to find contradictions between the process documentation and the actual process. Thirdly, to find differences on how different process stakeholders use the process. The interviewees should be chosen so that they represent different process stakeholders in as broad way as possible.

Execution of interviews. Minor modifications to the model should be made interactively with the interviewees to illustrate and verify the model changes immediately. If possible, notes should be made from the comments of the interviewee.

Refinement of model. Model should be refined right after or even during the interviews based on the interviewee comments. A proper modeling tool is essential since it allows fast, interactive and flexible modeling.

Verification of final model. The purpose of the step is to verify that the modelers have understood the interviewees correctly and validate that the model is accurate enough for indented use. If there are major contradictions between participants, the interviews should be re-planned and repeated.

Closing of modeling project. The lead modeler should create a baseline of the model and make sure that it is stored properly and distributed to all stakeholders.

CASE STUDY AND RESULTS

Modeling Environment

The Gaudí Software Factory [8] is a software construction unit at the department of Information Technologies at Åbo Akademi University. The goal of the Gaudí factory is to produce software for the needs of various university research projects. Software process in the Gaudí factory is based on agile methods, particularly on Extreme Programming [9]. It is characterized by short (one or two weeks) iteration cycles followed by small releases, and close in-

volvement of a customer [10] or customer proxy [11] in software projects. The Gaudí process is defined as a collection of so-called software best practices focusing on product quality and project agility [12]. Some of the practices were directly borrowed from Extreme Programming, while the most of them were adapted to suit the university environment [13].

Modeling Steps

Besides the description of the software best practices which comprise the Gaudí process, there is no other description of the process in a conventional way. However, a need for a clearer description of the Gaudí process has been arising, mainly because of two reasons. Firstly, due to the high staff turn-around [8] in Gaudí, there is a need for clear process description to teach the process to new programmers, customers and coaches. Secondly, the Gaudí process is supposed to be very flexible and allow tailoring – an accurate process model would be a great help in this task. Therefore, the Gaudí factory makes a good pilot study for testing our approach for software process modeling.

The initial model was created based on the technical report on the Gaudí Software Factory [12]. After completing the initial model, it was verified by interviewing the process expert who was one of the original authors of the Gaudí process description. Also the key stakeholders of the process were identified and interviews with them and modeling team were scheduled. The key stakeholders, a developer and a coach from different projects, were interviewed about the actual hands-on execution of the process during the development projects. The interviews revealed new details about the process, problems in the initial work flow, and elements that were over-emphasized in the original process description. The interviews also complemented each other. The developer had more information about actual development tasks while the coach knew better about the managerial issues. This is of course quite obvious finding, but shows that making a comprehensive model with low effort is possible only if the interviewees are selected carefully at the beginning of the modeling project. An initial model was modified during the interview session based on the comments of the interviewees. Notes were also made from the interviewees' feedback to be used in subsequent steps.

After the interviews, the draft model was modified based on the interview notes. This model was again verified by the process expert and the relevance of the model was evaluated. Some minor adjustments were made, but in general the model was satisfying. The final model is presented in the next section.

Final Model

Interviews revealed that there are two kinds of projects in the Gaudí factory. Therefore the main iteration was divided into two different models: customer driven and technical coach driven. The former process model is used in projects with clear customer role. In this case separate acceptance testing is done and there is need for release builds. This model can be seen in Figure 2. The latter process model is used when the coach of the project also works as a customer proxy. In this case the role of the coach resembles more of the traditional project manager role. Because the customer proxy i.e. the coach works closely in the project, separate acceptance testing is not needed. Instead, the coach evaluates the functionality of the software constantly by doing exploratory testing. This model is illustrated in Figure 3.

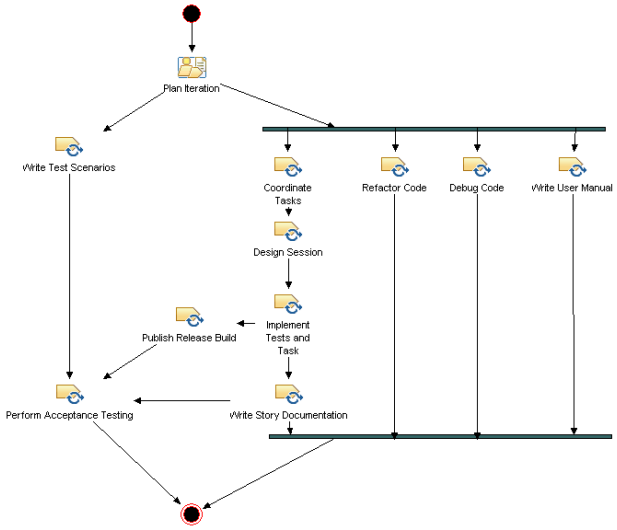

Figure 2 Modeling step 4 a: Final model for the customer driven projects.

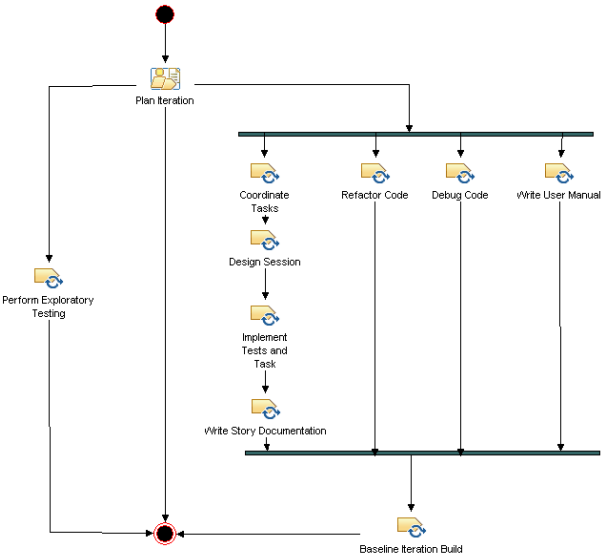

Figure 3 Modeling step 4 b: Final model for the technical coach driven projects.

Table 1 Estimated effort of the modeling participants in hours. Legend: LM = Lead modeler, AM = Assistant modeler, PE = Process Expert, C = Coach, D = Developer.

Step	LM	AM	PE	C	D	Total
Project Start	1	1	1	-	-	**3**
Initial Modeling	15	-	-	-	-	**15**
Verification of Initial Model	1.5	1.5	1.5	-	-	**4.5**
Interviews Planning	0.5	-	0.5	-	-	**1**
Interviews Execution	3	3	-	1.5	1.5	**9**
Model Refinement	3	3	-	-	-	**6**
Verification of Final Model	1.5	1.5	1.5	-	-	**4.5**
Project Closing	1	-	-	-	-	**1**
Total	**26.5**	**10**	**4.5**	**1.5**	**1.5**	**44**

Analysis of the Modeling Experiment

The model of the Gaudí software process presented in the case study turned out to be an accurate enough for the selected purpose. The SPEM diagrams of the model give a clear overview of the Gaudí process to process owners and software engineers. The model captures most of the process activities.

Finally, it is significantly faster to capture the basics of the Gaudí process with the presented diagrams, rather than reading the existing 27-page Gaudí document [12].

Since the proposed modeling approach is straightforward there were neither major delays nor problems during the modeling. We were able to follow the proposed modeling approach quite faithfully. The total work effort was approximately one man-week distributed into one month's period. The details on the modeling effort are presented in Table 1.

CONCLUSIONS

In this paper we have presented an approach for conducting as-is process modeling. The presented approach is a fast and light-weight way to model the existing software process of an organization, when some process documentation exist.

As a proof of concept we have also presented a case study on process modeling carried out in the Gaudí Software Factory. As a result of this case study, we have seen that the proposed approach works in practice. Furthermore, the approach turned out to satisfy our preset requirements as it is fast, has low interference with development, does not require extensive consultation and is methodology independent. In addition, the resulting process model showed directions for the process improvement in the Gaudí factory.

The modeling approach proposed in this paper resulted in two final models for Gaudi factory: the *customer-driven* and *coach-driven* models (see Section 3.3). These two types of the process have been established in the Gaudí factory since its start, but were never implicitly documented anywhere. This discovery speaks for the accuracy and objectivity of the proposed modeling approach in the presented case study and shows the ability of bottom-up modeling to reveal existing, undocumented high-level development strategies.

However, the final model is not complete. For example, the model does not include full work-product and role descriptions. In fact, we do not see the need to model comprehensive work-product flow because of the agile nature of the Gaudí process. Nevertheless, the model is accurate and we believe that complemented with essential work-product templates and role descriptions, the model would serve as good process documentation in the Gaudí factory.

While Gaudí factory offers a fine, controlled environment to experiment with the process modeling techniques, we will also test the proposed modeling approach in purely industrial settings. The problems with process modeling also apply to the purely commercial software development, namely how process documentation can be supported by modeling techniques and how models can be flexibly tailored to fit different development needs. The forthcoming industrial cases will represent different development methods, so that technological independence of the modeling approach and its applicability in different situations and organizations can be verified.

REFERENCES

[1] Peter Haumer. Second revised spem 2.0 submission. OMG Meeting, 2006.

[2] Object Management Group. Software Process Engineering Metamodel Specification, v2.0, March 2007. ptc/07-03-03.

[3] Object Management Group. Software Process Engineering Metamodel Specification – Version 1.1, January 5 2005. formal/05-01-06.

[4] Eclipse process framework project homepage. http://www.eclipse.org/epf/. Accessed on March 16 2007.

[5] Antero Järvi, Tuomas Mäkilä, and Harri Hakonen. Changing role of spi - opportunities and challenges of process modeling. In Proceedings of the 13th European Conference on Software Process Improvement (EuroSPI 2006), volume 4257 of Lecture Notes in Computer Science, pages 135 – 146. Springer Berlin / Heidelberg, October 2006.

[6] Ansgar Schwegmann and Michael Laske. Process Management - A Guide for the Design of Business Processes, chapter As-is Modeling and Process Analysis, pages 107 – 133. Springer, 2003.

[7] Marc I. Kellner and Gregory A. Hansen. Software process modeling. Technical report, Software Engineering Institute, May 1988.

[8] Ralph-Johan Back, Luka Milovanov, and Ivan Porres. Software development and experimentation in an academic environment: The gaudí factory. Journal of Systems and Software, 2007. To appear.

[9] Kent Beck. Extreme Programming Explained: Embrace Change. Addison-Wesley, 1999.

[10] Ralph-Johan Back, Piia Hirkman, and Luka Milovanov. Evaluating the XP Customer Model and Design by Contract. In Proceedings of the 30th EUROMICRO Conference. IEEE Computer Society, 2004.

[11] Piia Hirkman and Luka Milovanov. Introducing a Customer Representative to High Requirement Uncertainties. A Case Study. In Proceedings of the International Conference on Agile Manufacturing, 2005.

[12] Ralph-Johan Back, Luka Milovanov, and Ivan Porres. Software development and experimentation in an academic environment: The gaudi experience. Technical Report 641, TUCS, Nov 2004.

[13] Luka Milovanov. Agile Software Development in an Academic Environment. PhD thesis, TUCS, Dec 2006.

Tuomas Mäkilä, Harri Hakonen, Jouni Smed and
Andy Best

Three Approaches Towards Teaching Game Production

M. Kankaanranta, P. Neittaanmäki (Eds.), Design and Use of Serious Games, Intelligent Systems, Control, and Automation: Science and Engineering, pages 3–18. Springer Netherlands, 2009.

Tuomas Mäkilä and Henrik Terävä

Survey of Practitioners' Attitudes to Software Process Modeling

In Industrial Proceedings of the 16th European Conference on Software Process Improvement (EuroSPI), pages 12.25–12.33. Delta, 2009.

Survey of Practitioners' Attitudes to Software Process Modeling

Tuomas Mäkilä, University of Turku, Finland
Henrik Terävä, Digia Plc., Finland

Abstract

Software process modeling has evolved fast during the past few years. New dedicated modeling standards and process-based tools have been introduced. Emerging trends of the process modeling could bring even more radical changes. These changes have affected the work of common software industry practitioners. Results of a qualitative survey, which was conducted amongst Finnish software practitioners, are presented in this paper. The goal of the survey was to map the attitudes of the practitioners towards the use of software methodologies and software process modeling in their own work. In addition the practitioners provide expert analysis on the emerging modeling trends. The answers of the practitioners are analyzed in this paper and conclusions are given on how the practitioners see the current state and the near future of the software process modeling.

Keywords

Software Process Modeling

Introduction

Software process modeling and software process models are essential techniques in making understandable process descriptions. Efficient modeling and clear, up-to-date process models are essential part of many software process improvement (SPI) activities. Several different techniques have been used to construct the models: Work-flow diagrams, more advanced business process modeling languages, and also dedicated software process modeling languages. Development in the field of software process modeling has enabled new applications for the models and brought the models closer to the every-day project work in the software industry.

Many practical applications of the process modeling require strong tool support. For example a simple task like keeping a process model up to date can become laborious with basic office tools. Existing modeling language standards simplify and in many cases enable the implementation of modeling tools. Modeling languages standards are also essential for model reuse and model interchange between different organizations. There has been swift development in both modeling standards and tool support during the past five years. This development has increased the software industry's interest towards the software process modeling.

Second version of the Software Process Engineering Metamodel (SPEM) modeling language standard [1] was released in 2007. The SPEM modeling language has provided a foundation for development of new generation of software process modeling tools. IBM has constantly published improved versions of its Rational Method Composer (RMC) [2] modeling tool which is indirectly based on the SPEM standard. Partly based on the RMC code, almost identical tool is freely available through the Eclipse Process Framework (EPF) project [3]. The EPF project also distributes models of several popular process methodologies. Latest advancement is the IBM's Team Concert tool [4] which uses simple process models as a mean to configure the project tools (e.g. user rights, version tracking rules and communication). Microsoft has also same kind of ideas in their Visual Team System product [5].

This paper reports the results of a qualitative survey of software practitioners' attitudes on software process modeling. There were three goals for the survey. First goal was to get objective information about the current state of the process modeling in the software industry. This was done by obtaining information about practitioners' personal attitudes towards process modeling and also about the underlying work methodologies. Second goal was to get expert analyses on the upcoming trends of the process modeling. Third goal was to evaluate the research team's own beliefs of the process modeling which were based on the literature and the team's own experiences.

The research was conducted as a two-part qualitative survey. In the first part the respondents answered the web-based questionnaire at their own pace. The questionnaire included mainly open questions grouped in three categories: 1) Influence of work processes in respondents' own work, 2) influence of process models in respondents' own work, and 3) analysis on the upcoming trends of process modeling. In the second part of the survey the answers of the selected respondents were supplemented during personal interviews. The interviews were done after the preliminary analysis of the answers of the first part questionnaire.

The survey results were analyzed by our research team. The team consisted of both university researchers and industry practitioners. The idea was to use a scientific method to get relevant results, which would also serve industrial needs. During the analysis the research team tried to find common elements and also contradictions in the answers to form meaningful conclusions. In addition to the qualitative analysis, the team used straightforward quantitative methods to analyze the respondents' answers to certain, individual questions. It should be noted that the percentage values presented in this paper can not be directly generalized outside this study.

The structure of the paper is following. In Section 0 the respondents' opinions on the impact of the software methodologies, processes and models to their every-day work are presented. In Section 0 respondents' views on the selected software process modeling trends and the future of the software process modeling are analyzed. Section 0 presents the research team's plans on continuing and extending the research of practitioners' opinions on the software process modeling. Finally in Section 0 the paper is concluded.

Current State of Modeling

The questionnaire was send to several software companies from the research team's partner network in Finland during spring 2009. Twenty practitioners (N=20) from fifteen different companies participated the survey. The total number of people who actually received the invitation to the survey is hard to determine, but about one fourth of those who opened the survey web page actually filled the survey.

Respondents' company sizes varied quite evenly from micro companies to large companies. The roles of the survey respondents varied from developer and project manager to process engineer and company executives. When the more detailed work descriptions of the respondents were analyzed there was usual variation between the work descriptions. No particular sector of the ICT industry was overemphasized. Most of the respondents worked in the various software development projects, but there were also respondents working in e.g. ERP system development, methodology engineering and maintenance projects. The exact distributions of company sizes and respondent work roles are presented in Table 1.

Company Size	Respondents
1-9 employees	10 %
10-99 employees	10 %
100-499 employees	30 %
500-3000 employees	35 %
over 3000 employees	15 %

Role	Respondents
Project worker	40 %
Project manager	30 %
Process engineer	20 %
Business management	10 %

Table 1 Company sizes and roles of the respondents

The respondents were inquired about the methodologies and process frameworks their employees had implemented. This was done to better understand the use of process modeling and modeling needs in respondents' everyday work. Based on the detailed answers, it can be said that most of the respondents were quite well aware of the methodologies used in their companies and the maturity of the used methodologies.

It was found out that 85 % of the respondents recognized at least one methodology to be used in their company. The methodologies used in the respondents' companies varied and there was not one clearly dominant methodology. For example ISO standards, ITIL, RUP or variant, CMM(I) and Agile methodologies were mentioned by multiple respondents. Half of the companies used more than one methodology. Usually a standard methodology was accompanied by company's own process methodology or guidelines. The maturity of the used methodologies also varied: 40 % of the companies had used a methodology for several years and 40 % were at the beginning of the methodology adoption. It could be seen from the answers that almost all of the companies were constantly developing their processes and evaluating underlying methodologies.

The information presented above is mainly background information. The respondents' views on the benefits and drawbacks of the methodologies and process modeling are analyzed next.

By analyzing the overall attitudes of respondents it can be concluded that 65 % of the respondents found the process frameworks beneficial for their work, 15 % had negative experiences, and 20 % had neutral attitude. Negative experiences seemed to be result of poorly defined or too inflexible processes. Although many respondents had positive overall experience, most of them also found some negative aspects in the use of the methodologies. In addition many respondents emphasized that the use of the guidelines and the methodologies has to be adapted case by case in order to get the most out of them.

More detailed analysis of attitudes revealed that the many respondents saw the methodologies as a kind of foundation for either the development work itself or the improvement activities. Many of them also said that methodologies enable reuse of practices which in turn saves time in different phases of project work. On the other hand the reuse could lead to repeating old mistakes. Other more negative attributes connected to the methodologies were inflexibility and unnecessary overhead caused by a methodology.

Although only 20 % of the respondents worked mainly with methodology improvement issues, almost

all respondents had participated in the process improvement activities: 55 % directly, 35 % indirectly e.g. by giving feedback about the process, and 10 % had not participated at all.

The use of the software process modeling was investigated by a set of yes / no claims about the state of the process descriptions and modeling in the respondents' companies. The answers are collected in Table 2. It can be said that while the use of standard methodologies was unexpectedly high in the respondents' companies the use of process modeling seemed to be more normative. The table shows that the advanced process modeling techniques like dedicated modeling tools, formal process models and use of modern process modeling language were still rare in the companies. It is also notable finding that number of the *"unknown"* answers increased when more technical issues were inquired.

Claim	Yes	No	Unknown
Implicit documentation	85 %	10 %	5 %
Several fragmented documents	65 %	25 %	10 %
Centralized documentation	45 %	55 %	0 %
Hypertext-based documentation	55 %	35 %	10 %
Dedicated process documentation tool	20 %	65 %	15 %
Processes described mainly as natural language	60 %	30 %	10 %
Processes described as natural language and models	50 %	30 %	20 %
Processes described as formal process models	15 %	65 %	20 %
Formal models done with drawing tools	50 %	25 %	25 %
Formal models done with process modeling tools	25 %	50 %	25 %
Modeling done with self-made language	10 %	65 %	25 %
Modeling done with work-flow diagrams	55 %	25 %	20 %
Modeling done with dedicated language	20 %	60 %	20 %

Table 2 How the work processes are documented in respondents' companies

The process modeling tools and languages the respondents use in their work was also inquired. 35 % of the respondents used dedicated process modeling tools, 25 % used some other tools like drawing software while 40 % did not use any process modeling tool or language. SPEM was mostly used modeling language while Business Process Modeling Notation (BPMN) and Universal Modeling Language (UML) were also in use.

The concrete use of the process models revealed that about half of the respondents used models just to access common document templates or checklist. Only one fourth of respondents mentioned process models as a tool for process tailoring and software process improvement. The question about the use of process models also showed that many people used the word *"model"* as a synonym for the word *"methodology"*. This was confusing because the research team used the word *"model"* to represent a result of modeling efforts.

The respondents would develop the use of the process models and modeling in their company in various ways. Only 75 % of all respondents actually answered to the model development question. All of them saw improvement possibilities and were able to specify clear development suggestions. Many respondents mentioned that the modeling tools should be improved and the use of dedicated modeling language increased. One respondent noted that this could take longer than expected: *"Formal modeling is our next step but this step is bigger than we first thought"*.

Some respondents wanted to increase flexibility of the models. One rationale behind this was to enable tailoring of the models for different situations. There were also suggestions about making a library of more detailed process models for different small scale situations. This approach resembles the emerging practice-based process modeling [6] [7] which is probably still quite unknown amongst the practitioners. Rest of the respondents wanted to increase the use of models by making them clearer, easier to read, and more comprehensive. Also training for using the models was needed.

Answers about the process modeling seemed to suggest that the maturity of the software process modeling was lower than the maturity of software methodologies which were used in companies. 85 % of the respondents knew that their company utilizes at least one development methodology but only 60 % used some kind of modeling tool. It also seems that the respondents were more familiar with methodologies and their use than meaning and the use of the software process modeling.

Emerging Trends

The second goal of the survey was to get practitioners' analysis on the vitality of emerging software process modeling trends. Three most interesting modeling trends were selected based on the research team's previous research [8] and the recent advancements in the modeling techniques. The trends that were presented to the respondents were:

- **Distributed process modeling.** Distributed process modeling is an approach which emphasizes bottom-up modeling practices. Portions of the process models are done in the projects where the process is used. This approach is opposite to the top-down, process engineer led process modeling. In practice, both aspects have to be taken into consideration. Techniques for distributed process modeling are proposed e.g. in the paper [9].
- **Light-weight process modeling.** Light-weight process modeling means focusing only on the most important elements during the process modeling. Idea is to quickly form a starting point for longer modeling efforts or to quickly illustrate the current state of the process. The approach is opposite to the traditional business modeling techniques where target is to generate very accurate models. Techniques for light-weight process modeling are described in the papers [10] [11].
- **Decrease of project-process-gap.** There's always overhead when process description and methodologies are enacted into an actual project organization. The gap can lead to process deviations and make measuring the project difficult. The gap can be decreased with modern process modeling techniques for example by configuring the project tools using the actual process descriptions. Ivar Jacobson has discussed about the project-process gap and developed a practice-based method to deal with the issue [6]. Process modeling techniques for reducing the gap are presented e.g. in the paper [12].

The respondents were asked to evaluate whether the techniques described in the trends would be applicable to their own work and if they would benefit from these trends. They were also inquired to analyze if some of the techniques were already applied in their companies.

Surprisingly many of the respondents had hands-on experiences on *the distributed process modeling*: 30 % of them had at least tested the principles related to this trend. Half of the respondents had positive attitudes towards this trend while others were neutral about the trend. The respondents liked the idea that the process users can directly affect the process models. They also saw that it is efficient to define process where it is used. This way there would be less process deviations in the project level and the overall process model would better resemble the reality.

The respondents found also many possible pitfalls in the distributed process modeling. The largest concern was the integration of the distributed models into one company-wide model. Many respondents mentioned that strict distributed modeling would not work, but traditional top-down techniques would still be needed to accompany the distributed modeling. There were worries about extra workload and insufficient skill levels of the project workers who would have to participate more actively in process modeling. As a solution the respondents offered a modeling facilitator who would do the actual modeling in co-operation with the project team, and take care of the integration and other technical issues.

The light-weight process modeling was a little bit more unfamiliar concept: Only two of the respondents had tried the techniques related to this trend. Controversially to the previous trend even 70 % of the respondents had positive attitudes towards this trend. The respondents liked the idea to model only necessary elements and reduce unnecessary overhead. The trend was connected to the principles of agile methods by several respondents. Iterative process development was also mentioned.

The largest problem with light-weight modeling was how to identify the most important process elements and define the detail level of the modeling. Solutions for this problem were not found. Some respondents also identified that the light-weight process modeling has very focused applicability: It works best for sketching and piloting new methodological ideas, and for forming a starting point for longer lasting process modeling efforts.

The decrease of the project-process gap with process modeling techniques was clearly the most abstract concept for the respondents. Although 65 % of the respondents had positive attitudes towards

this trend, the analysis was not as detailed as with the previous trends. The main message was that it is hard to see how the project-process gap is actually decreased because the tools and modeling standards do not yet fully support this approach.

The survey was concluded with the question about the respondents' own opinions on the future trends of the software process modeling. Almost all respondents saw that meaning of the process modeling and process methodologies will generally increase in the near future. The respondents emphasized the importance of the development of the both process and project tools, and their interoperability. It seemed that there are methodologies, modeling languages and tools available already but their maturity is still quite low. Full potential of the modeling technologies is still to be reclaimed. Optimistically, the respondents believed that this will eventually happen.

Further Work

As mentioned before, this was a qualitative survey and therefore percentages presented in the paper give only hints on which issues were more important and which were less important for the respondents as a whole. Because of the small sample (N=20) the percentage values themselves are not statistically significant.

This study acts as a starting point for a longer research on the practitioners' attitudes and expectations towards the software process modeling. The qualitative analysis presented in the paper was a necessary step to form understanding of the modeling issues that are important for the practitioners. Next the research team is planning to conduct a statistical survey that will provide more comprehensive information on tighter formulated set of hypotheses.

In the study presented in the paper the population and the sample was limited to the Finnish software practitioners. Therefore conclusions can be drawn only about Finnish software industry. In the following study the survey will be conducted in several other countries as well. There might be regional differences on the attitudes since the software methodology culture seems to vary geographically. In the following study the research team expects to deepen understanding on the differences of the attitudes of different employee groups by using more formal statistical analysis.

It will also be interesting to follow how the field of the software process modeling will evolve in the near future. As the study continues the research team will observe how well the expert analyses of the respondents realize.

Conclusions

There should not be big surprises in the overall results of the survey for those who have followed the recent development of the software process modeling concepts, methods, languages, and tools. Practitioners welcome, with healthy criticality, new methodologies that will improve their ability to do their every-day work better. Naturally the methodological frameworks do not offer a silver bullet, but some kind of structures and guidelines are clearly needed in the software work.

The software process modeling concepts seem to be still a little bit unfamiliar for the practitioners, although the project and development methodologies are quite well known. Reason for this might be the immaturity of the modeling languages and the tools. It should also be noted that only portion of the practitioners actually modify the process models, and therefore work directly with the modeling tools and languages. For others it is sufficient to understand the models and maybe give constructive feedback about them. This situation might however be changing because of the trends presented in this paper.

The unity of the practitioners' answers for the survey was interesting. Despite the fact that the respondents represented many different work roles and different-sized companies, they all looked quite positively at the methodological issues and changes in the process modeling field. Maybe the people who are interested in these kinds of issues became selected as the respondents and this somehow biased the results. However, it can be said that there are people in the software industry who are

open-mindedly willing to adopt new methodologies, but at the same time they expect to see direct improvements in their work environment.

Acknowledgements: The authors wish to thank all practitioners who participated the survey from Digia, Codebakers, Ericsson, RP5 Software, Samlink, Sesca and several other software companies.

Literature

[1] Object Management Group. Software & systems process engineering meta-model specification - version 2.0, April 2008.

[2] Rational method composer (IBM) homepage. http://www-01.ibm.com/software/awdtools/rmc/. Accessed on May 14th 2009.

[3] Eclipse process framework (EPF) project homepage. http://www.eclipse.org/epf/tool component/tool index.php. Accessed on May 14th.

[4] Rational team concert homepage. http://www-01.ibm.com/software/awdtools/rtc/. Accessed on May 14th 2009.

[5] Team system (Microsoft) homepage. http://msdn.microsoft.com/enus/teamsystem/default.aspx. Accessed on May 14th 2009.

[6] Ivar Jacobson, Pan Wei Ng, and Ian Spence. Enough of processes: Let's do practices. Journal of Object Technology, 6(6):pp. 41-66, August 2007.

[7] IBM practices homepage. http://www.ibm.com/developerworks/rational/practices/. Accessed on May 14th 2009.

[8] Antero Järvi, Tuomas Mäkilä, and Harri Hakonen. Changing Role of SPI Opportunities and Challenges of Process Modeling, volume 4257/2006 of Lecture Notes in Computer Science, pages 135-146. Springer Berlin / Heidelberg, 2006.

[9] Oktay Turetken and Onur Demirors. Process modeling by process owners: A decentralized approach. Software Process: Improvement and Practice, 13(1):75-87, 2008.

[10] Tuomas Mäkilä, Antero Järvi, and Luka Milovanov. Light-weight approach for software process modeling - a case study. In Proceedings of New Exploratory Technologies 2007, October 2007.

[11] Paula Savolainen, Hanna-Miina Sihvonen, and Jarmo Ahonen. SPI with Lightweight Software Process Modeling in a Small Software Company, pages 71-81. 2007.

[12] Soojin Park, Hoyoung Na, Sooyong Park, and Vijayan Sugumaran. A semiautomated ltering technique for software process tailoring using neural network. Expert Systems with Applications, 30(2):179-189, February 2006.

Author CVs

Tuomas Mäkilä (tuomas.makila@utu.fi)

M.Sc.(in technology) Tuomas Mäkilä works as a teacher and a resercher at the Department of Information Technology of the University of Turku. He has researched software process modeling for five years and is currently finishing his doctoral thesis on the topic.

Henrik Terävä (henrik.terava@digia.com)

M.Sc.(in technology) Henrik Terävä works as a project manager at Digia Plc., a worldwide information and technology solutions provider. He is an expert on modern software process modeling technologies and develops Digia's Open Method software development methodology product.

Turku Centre for Computer Science
TUCS Dissertations

116. **Siamak Taati**, Conservation Laws in Cellular Automata
117. **Vladimir Rogojin**, Gene Assembly in Stichotrichous Ciliates: Elementary Operations, Parallelism and Computation
118. **Alexey Dudkov**, Chip and Signature Interleaving in DS CDMA Systems
119. **Janne Savela**, Role of Selected Spectral Attributes in the Perception of Synthetic Vowels
120. **Kristian Nybom**, Low-Density Parity-Check Codes for Wireless Datacast Networks
121. **Johanna Tuominen**, Formal Power Analysis of Systems-on-Chip
122. **Teijo Lehtonen**, On Fault Tolerance Methods for Networks-on-Chip
123. **Eeva Suvitie**, On Inner Products Involving Holomorphic Cusp Forms and Maass Forms
124. **Linda Mannila**, Teaching Mathematics and Programming – New Approaches with Empirical Evaluation
125. **Hanna Suominen**, Machine Learning and Clinical Text: Supporting Health Information Flow
126. **Tuomo Saarni**, Segmental Durations of Speech
127. **Johannes Eriksson**, Tool-Supported Invariant-Based Programming
128. **Tero Jokela**, Design and Analysis of Forward Error Control Coding and Signaling for Guaranteeing QoS in Wireless Broadcast Systems
129. **Ville Lukkarila**, On Undecidable Dynamical Properties of Reversible One-Dimensional Cellular Automata
130. **Qaisar Ahmad Malik**, Combining Model-Based Testing and Stepwise Formal Development
131. **Mikko-Jussi Laakso**, Promoting Programming Learning: Engagement, Automatic Assessment with Immediate Feedback in Visualizations
132. **Riikka Vuokko**, A Practice Perspective on Organizational Implementation of Information Technology
133. **Jeanette Heidenberg**, Towards Increased Productivity and Quality in Software Development Using Agile, Lean and Collaborative Approaches
134. **Yong Liu**, Solving the Puzzle of Mobile Learning Adoption
135. **Stina Ojala**, Towards an Integrative Information Society: Studies on Individuality in Speech and Sign
136. **Matteo Brunelli**, Some Advances in Mathematical Models for Preference Relations
137. **Ville Junnila**, On Identifying and Locating-Dominating Codes
138. **Andrzej Mizera**, Methods for Construction and Analysis of Computational Models in Systems Biology. Applications to the Modelling of the Heat Shock Response and the Self-Assembly of Intermediate Filaments.
139. **Csaba Ráduly-Baka**, Algorithmic Solutions for Combinatorial Problems in Resource Management of Manufacturing Environments
140. **Jari Kyngäs**, Solving Challenging Real-World Scheduling Problems
141. **Arho Suominen**, Notes on Emerging Technologies
142. **József Mezei**, A Quantitative View on Fuzzy Numbers
143. **Marta Olszewska**, On the Impact of Rigorous Approaches on the Quality of Development
144. **Antti Airola**, Kernel-Based Ranking: Methods for Learning and Performace Estimation
145. **Aleksi Saarela**, Word Equations and Related Topics: Independence, Decidability and Characterizations
146. **Lasse Bergroth**, Kahden merkkijonon pisimmän yhteisen alijonon ongelma ja sen ratkaiseminen
147. **Thomas Canhao Xu**, Hardware/Software Co-Design for Multicore Architectures
148. **Tuomas Mäkilä**, Software Development Process Modeling – Developers Perspective to Contemporary Modeling Techniques

Turku Centre *for* Computer Science

Joukahaisenkatu 3-5 B, 20520 Turku, Finland | www. tucs.fi

University of Turku
Faculty of Mathematics and Natural Sciences
- Department of Information Technology
- Department of Mathematics and Statistics

Turku School of Economics
- Institute of Information Systems Science

Åbo Akademi University
Division for Natural Sciences and Technology
- Department of Information Technologies

ISBN 978-952-12-2790-5
ISSN 1239-1883

Tuomas Mäkilä

Tuomas Mäkilä

Tuomas Mäkilä

Software Development Process Modeling

Software Development Process Modeling

Software Development Process Modeling