SUPPORTING PROCESS IMPROVEMENT USING METHOD INCREMENTS

KEVIN VLAANDEREN

SIKS Dissertation Series No. 2014-43
The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

Supporting Process Improvement using Method Increments
ISBN: 978-90-393-6227-3

Supporting Process Improvement using Method Increments

Ondersteunen van Procesverbeteringen met Behulp van Methode-Incrementen
(met een samenvatting in het Nederlands)

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Universiteit Utrecht op gezag van de rector magnificus, prof.dr. G.J. van der Zwaan, ingevolge het besluit van het college voor promoties in het openbaar te verdedigen op maandag 27 oktober 2014 des middags te 2.30 uur

door

Kevin Vlaanderen

geboren op 12 augustus 1986, te Hilversum

Promotor:	Prof.dr. S. Brinkkemper
Copromotoren:	Dr. G.C. van de Weerd
	Dr. F. Dalpiaz

ACKNOWLEDGEMENTS

Writing the acknowledgements for a dissertation such as the one you have in front of you requires its own form of research. I've spent almost a third of my life studying and doing research at the university, and I've met many inspiring, friendly and helpful people during that time. It takes some effort to recollect the memories of all of them. For three, however, I do not have to look very far. My promotor, Sjaak Brinkkemper, and my copromotors, Inge van de Weerd and Fabiano Dalpiaz, deserve my grattitude first and foremost. Without you, there wouldn't have been a dissertation.

They are accompanied by the staff, past and present, of the Organisation & Information group at Utrecht University, many of which I was able to call professor first, and colleague later. I've enjoyed working with Slinger, Remko, Marco, Sandra, Ronald, Johan, Rik, Jan Martijn, and my fellow Ph.D. students, Jaap (whom I can still count among my best friends after four years of sharing an office), Erik, Ravi, Amir, Michiel, Wienand, Eko, Ivonne, Willem, Henk, and all the others I didn't mention. Buy them a beer when you see them. They deserve it. And buy an extra one for Sandor, who had to deal with all the uncertainties that accompanied my research, and my mood.

Having experienced it first hand, it is easy to imagine that every researcher is prone to fall victim to a paradox of some sort. The more a researcher becomes familiar with his domain and the more he discovers about it, the smaller his own world can become. On various occasions, I have used the excellent Illustrated Guide to a Ph.D. by Matt Might[1] to put scientific research in perspective (for my students, but perhaps even more for myself). Most research is perfomed at a tiny fragment on the boundary of a huge body of knowledge, and it can make your work seem insignificant. But the dents are important. The world is big and complex, and every bit of knowledge makes it that more beautiful. Don't forget the bigger picture; keep pushing.

I am grateful to have had the opportunity to view that beauty throughout every part of my studies. One of the best choices I've ever made was to obtain a second bachelor degree in Spanish Language & Culture. The value of having (or rather,

1 Check out the illustration at http://matt.might.net/articles/phd-school-in-pictures/

obtaining) degrees in both science as well as the arts is something I never fail to emphasize to whomever wants to hear it. The people at Utrecht University's Department of Languages, Literature and Communication have taught me a lot, and so have all the people with whom I've spent my years there.

A warm thank you goes out to the people who allowed me to spend two periods of my student life abroad, and two of them in particular. Oscar Pastor at the University of Valencia, who charms every person that is lucky enough to cross his path, and Jolita Ralyté at the University of Geneva, who has paved the way for much of my own research. My stay with you has changed me for the better.

I am fortunate enough to have so many friends so that I will not be able to mention all of them here, and I will not make a choice. You're the ones I've travelled with, laughed with, had a beer (or four) with, lived with. You are awesome. Almost as awesome as my parents, whose proud smile has reminded me of the bigger picture many times over.

Finally, a big thank you goes out to all the people and organizations that were willing to invest some of their time to participate in the various case studies and evaluations that are an important aspect of this dissertation. You have been patient and helpful, and you have shown that science and industry go hand in hand.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

AIM	Areas of Improvement Matrix
BPM	Business Process Management
BPML	Business Process Modeling Language
BSC	Balanced Score Card
CAME	Computer-Aided Method Engineering
CASE	Computer-Aided Software Engineering
CCP	Current Capability Profile
CM	Change Management
CMM	Capability Maturity Model
CMMI	Capability Maturity Model Integration
CO	Controller Object
DAMM	Dynamic Architecture Maturity Matrix
DAO	Data Access Object
DSB	Development Sprint Nacklog
DSDM	Dynamic Systems Development Method
FDD	Feature Driven Development
GOPRR	Graph, Object, Property, Relationship, Role
GQM	Goal/Question/Metric
IS	Information Systems

JCR	Java Content Repository
KM	Knowledge Management
MaaS	Method-as-a-Service
ME	Method Engineering
MF	Method Fragment
MOA	Method-Oriented Architecture
MOF	Meta-Object Facility
MVC	Model-View-Controller
OC	Organizational Capability
OCP	Optimal Capability Profile
OME	Online Method Engine
OSGi	Open Service Gateway Initiative
PB	Product Backlog
PDD	Process-Deliverable Diagram
PDDL	Planning Domain Definition Language
PDO	Persistent Domain Object
PMSB	Product Management Sprint Backlog
PSKI	Product Software Knowledge Infrastructure
QIP	Quality Improvement Paradigm
QUPER	Quality Performance
SAM	Situational Assessment Method
SDRP	Systems Development Research Process
SE	Software Engineering

SF	Situational Factor
SFE	Situational Factors Effect
SME	Situational Method Engineering
SOA	Service-Oriented Architecture
SPI	Software Process Improvement
SPICE	Software Process Improvement and Capability dEtermination
SPM	Software Product Management
UML	Unified Modeling Language
VO	Value Object
WCB	Web Component Bundle
XP	eXtreme Programming

Part I

INTRODUCTION

1

INTRODUCTION

1.1 THE CHALLENGE OF CHANGE WITHIN (SOFTWARE) ORGANIZATIONS

In the Schumpeterian world of software producing organizations, "where entry and exit barriers are low, marginal costs of production are minimal, product innovation occurs rapidly and disruptively, and firms' competencies and strategies are critical for competitive advantage"—as cited by Giarratana and Fosfuri (2007) and Schmalensee (2000)—there is a constant influx of new requirements and an ever-changing technological landscape. In their paper titled "The New Competitive Landscape," Bettis and Hitt (1995, p. 16) conclude that modern organizations "exist within highly turbulent and often chaotic environments that produce disorder, disequilibrium, and significant uncertainty". Organizations need to rely on their organizational capabilities and resources instead of on long term planning (Barney, Wright, and Ketchen, 2001; Teece and Pisano, 1994), in order to cope effectively with these unstable environments. Changes in the market force organizations to alter software products mid-development and to take a liberal stance regarding product planning (Boehm, 2006). This requires organizations to be flexible, while delivering quality software to the market as fast as possible (Conboy, 2009; Lee and Xia, 2010).

This ability "to rapidly or inherently create change, proactively or reactively embrace change, and learn from change while contributing to perceived customer value [...]" (Conboy, 2009, p. 340) is required not only with regard to the functionality they produce, through constant reprioritization (Gupta, Chauhan, Dutta, et al., 2013; Racheva, Daneva, and Buglione,

2008), but also in relation to the organizational structure and its processes, as competitors race them for market leadership (Teece and Pisano, 1994).

The characteristics of change within organizations have been a topic of scientific debate for decades (Miles and Snow, 1978), and organizational change has been studied from the perspective of strategy, structure, processes, technology, and the environment (Kanellis, Lycett, and Paul, 1999). In one study, Smith (2002) analyzed the success rate for a large collection of organizational changes of different types, and found an average success rate of 33% over all types of organizational changes, ranging between 58% for strategic deployment and 19% for cultural change.

Amongst the lowest scoring categories, we also find 'software development and installation' and 're-engineering and process design' (Smith, 2002), suggesting that a part of any organizational change is getting people to change how they work and interact. While each of the facets of organizational change is shown to be demanding—and warrant their own field of study, such as Business Process Management (BPM) and Change Management (CM)—there are strong indications that process re-engineering or Software Process Improvement (SPI) pose additional challenges. This conclusion is worrying, as several studies show a potential positive effect of increased process maturity on product quality and customer satisfaction (Harter, 2000; Kuilboer and Ashrafi, 2000). SPI seems to be an example of the chicken and the egg; practitioners often want to know what the benefits for them will be, before supporting any changes. However, evidence of potential success is often not accepted, unless it is shown that it relates directly to them (Baddoo, 2003). In this dissertation, we aim to narrow this gap by linking changes directly to the situation at hand, and supporting the implementation of those changes within that context.

1.2 THE LACK OF SITUATIONAL KNOWLEDGE

The challenge of tying knowledge to a specific context is present within every aspect of the software development domain. For example, ever since the advent of Software Engineering (SE), researchers and practitioners alike have come up with a wide variety of metaphors to depict the endeavour of producing a successful, functional software system. In his famous textbook on programming practices, *Code Complete: A Practical Handbook of*

Software Construction, McConnell (1994) provides a list of metaphors that range from programming as an act of science (Gries, 1981) to programming as an art (Knuth, 1997), and from gardening (Venners, 2003) to the fairly tongue-in-cheek image of 'drowning with dinosaurs in a tar pit' (Brooks, 1995). The images raised by these metaphors are widely different, and sometimes even opposite to each other, making it clear that the creation of software systems is intrinsically different from other industrial processes that we are familiar with, despite the fact that terms such as Software Engineering, software production, and software architecture are commonplace. When we compare SE to the building of houses, we quickly need to realize that it is not (yet) possible to plan and execute these processes at the same level of detail and with the same level of confidence. The architecture of a building and the architecture of a software system are concepts that can be compared only at a superficial level (Bass, Clements, and Kazman, 2012). Whatever metaphor we pick, it will not be perfect; software and the processes surrounding its creation behave differently than the artifacts and processes we know from older engineering disciplines.

The software industry has shown a shift during the past decade, from project-driven software development to market-driven product development (Hietala, Kontio, and Jokinen, 2004), along with a considerable portion of service revenue (Cusumano, 2008). With this shift, the range of influences, issues and benefits has changed tremendously, giving rise to the domain of Software Product Management (SPM), defined as "the process of managing requirements, defining releases, and defining products in a context where many internal and external stakeholders are involved" (van de Weerd, Brinkkemper, Nieuwenhuis, et al., 2006a, p. 319). Unfortunately, hardly any schooling is available in this field (Ebert, 2007), and product managers have often evolved into that function after first performing a development or project management function (Ebert, 2007). Because of this, a lot of product managers miss essential skills or pieces of knowledge, required for performing their function effectively. The transfer of knowledge between academy and industry is far from optimal, reducing much of the fundamental research currently produced to a state where it is not helpful anymore, thus throwing away a lot of potential (as we will show in Chapter 6).

The field of SPM is experiencing a steady flow of new techniques and methods that are required to deal with this new situation, developed both

by academia as well as practitioners. The problems that are dealt with can vary from distributed prioritization of requirements (Regnell, Höst, Natt och Dag, et al., 2001) to strategic release planning (Svahnberg, Gorschek, Feldt, et al., 2010) and from agile SPM (Dzamashvili-Fogelström, Numminen, and Barney, 2010) to the assessment of SPM processes (Bekkers, Spruit, van de Weerd, et al., 2010). The SPM community is working hard to figure out how to produce software effectively and efficiently, and the way in which they often do this is by developing ever more new solutions to the same problems, with varying success in varying conditions. However, a structured mapping study has shown that the research in the field is fragmented (Maglyas, Nikula, and Smolander, 2011), giving us many specialized solutions and little knowledge about how to match potential solutions to a specific context. This leaves a lot of room for research into situationality of SPM methods and success factors.

1.3 THE NEED FOR INCREMENTAL PROCESS IMPROVEMENT SUPPORT

The lack of situational knowledge can inhibit effective and efficient process improvement, as practitioners are often unable to match appropriate methods to their specific context. Maturity models such as Capability Maturity Model Integration (CMMI) (CMMI Product Team, 2010) and Bootstrap (Kuvaja, Simila, Krzanik, et al., 1994), and improvement methods such as Software Process Improvement and Capability dEtermination (SPICE) (El Emam, 1997), Six Sigma (Pyzdek and Keller, 2003) and IDEAL (McFeeley, 1996) provide high-level guidance in this respect. They aid the organization in increasing their capabilities, but they come at a cost. Many organizations feel the cost and complexity of such frameworks to be too high (Staples, Niazi, Jeffery, et al., 2007), even though a large number of successful implementations has been recorded.

In response to this argument, we can in turn see a move to the development situation-specific variants of these models and frameworks. CMMI has, for instance, been evolved into domain-specific models for services (Team, 2010) and outsourcing (Hofmann, Yedlin, Mishler, et al., 2007). SPICE variants exist for the automotive industry (Automotive Special Interest Group, 2010) and the space industry (Cass and Volcker, 2000). These

adaptations increase the relevance of such frameworks, but it also shows their inherent inflexibility to adapt to constant change and limited resources. It is therefore not surprising that voices have begun to emerge that advocate to combine traditional process improvement frameworks with an agile mindset (Glazer, Dalton, Anderson, et al., 2008; Omran, 2008; Turner and Jain, 2002).

The key behind such proposals is the idea that "there is no detailed information systems [development] methodology which is the best in all situations" (Kumar and Welke, 1992). We need to find a balance between the relative predictability of well-tested approaches such as CMMI, and the flexibility of situation-specific solutions. As described above, the scientific corpus provides such solutions for many domains. The problem then is often how to decide which solutions are applicable to a specific situation, or how they should be adapted to that situation, which is the main driver behind the field of Method Engineering (ME) (Brinkkemper, 1996; Henderson-Sellers, 2003; Rolland, Prakash, and Benjamen, 1999). ME is one of the various research domains that attempts to tackle the complexities of SPI, and it does so from a strong process-and deliverable-centric viewpoint. A common approach among many proposals within the field is to take a base method and adapt it to a specific situation through several strategies, including assembly of multiple method fragments, extension of a method, and a paradigm-based approach that takes a paradigm model or meta-model as its starting point (Ralyté, Deneckère, and Rolland, 2003).

These strategies aid the construction of relevant methods, and thus provide part of the solution to a more dynamic approach to SPI. However, they do not allow us to model the temporal dimension of SPI. For that, we need concepts that allow us to model changes and reasoning the reasoning behind change-related choices. One such concept is the *method rationale*, which provides a means to represent "decisions related to method use and decisions related to method construction" (Rossi and Tolvanen, 2000, p. 3).

Method rationale is essential to constructing and using any meaningful method. However, it is not sufficient to model change. A technique that lies closer to the actual modeling of method fragments is the *method increment*. Method increments are "a collection of method fragments that have been introduced in the method during the method adaptations between t_i and t_{i-1}" (van de Weerd, Brinkkemper, and Versendaal, 2007, p. 474). They are modeled using an adaptation of the Process-Deliverable Diagram (PDD)

(van de Weerd and Brinkkemper, 2008), and focus on the addition, removal and modification of activities and deliverables.

Both concepts aim to develop a better understanding of the specifics of adaptations to an existing method. It is the combination of knowing what to change, understanding why certain changes are meaningful, and understanding how these changes can be realized that has the potential to produce effective, and equally important, implementable process changes. However, as Rossi and Tolvanen (2000, p. 3) state, "although experience is known to be crucial to ISD it is not easy to build up and maintain". It is this challenge that is one of the core objectives of this dissertation.

1.4 CONCEPTUAL FRAMEWORK

The research presented in this dissertation is firmly based on the foundations of ME. Throughout the past two decades, a variety of concepts and techniques has been proposed in this domain. Some of these proposals have become a real or de-facto standard—such as ISO/IEC 24744 (ISO/IEC, 2007)—while others take a more ad-hoc approach to the problem (Brinkkemper, 1996; Karlsson and Wistrand, 2006; Ralyté, 2004; Wistrand and Karlsson, 2004). In many cases, the proposals show significant similarities, that make them to some degree compatible (Henderson-Sellers, Gonzalez-Perez, and Ralyté, 2007). However, in order to prevent confusion about the concepts used within this research, and to frame the results within the appropriate context, we present a conceptual framework that depicts the main relationships between the relevant concepts. The conceptual framework presented in Figure 1.1 is an illustrative domain model with this particular research and its foundations as its scope.

cf. Knowledge & Experience (Figure 1.1)

At the core of the framework lies the method base, an aggregation of meta-and meta-meta-level constructs related to:

- Method Engineering, including elements such as method assembly rules and method discovery templates (in Chapter 4, we present a proposal for a method increment discovery template);
- method knowledge for a particular domain, containing Organizational Capabilities (OCs), Situational Factors (SFs), Method

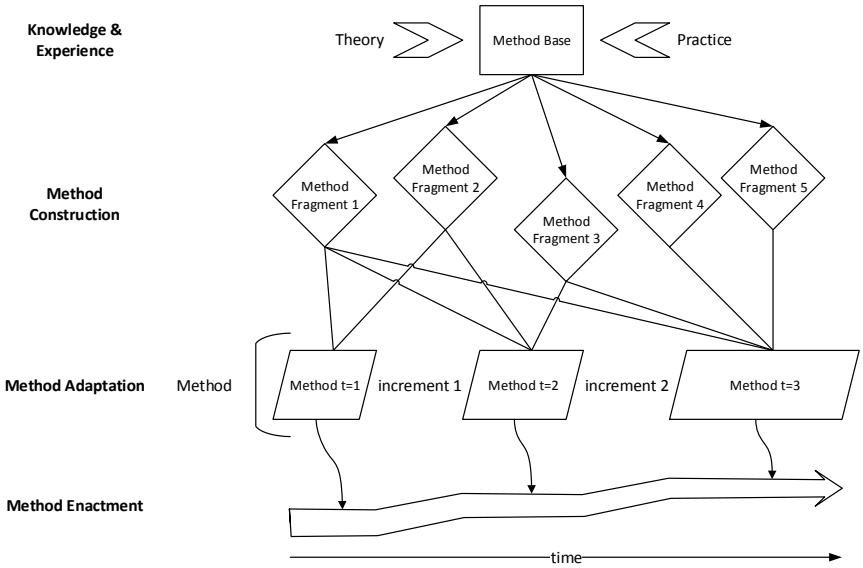


Figure 1.1.: Conceptual model of the major research artefacts

Fragments (MFs), and Situational Factors Effects (SFEs) (the agile SPM method described in Chapter 2 is an example of a MF); and

- method experience, in the form of case descriptions and evaluations (the case studies described in Chapter 3 are examples of such method experience).

Input for the method base is delivered through multiple channels, that can be broadly classified as theory and practice. Theory entails for example generalized solutions to specific problems (Method Fragments described in scientific literature), maturity models, and the aforementioned assembly rules. Practice includes experience reports for enacted MFs, maturity benchmarks, and cost estimations for process changes. To some extent, these two categories overlap (e. g. in the case of MF descriptions).

cf. Method Contruction (Figure 1.1)

MFs are the main building blocks within method engineering. They represent reusable, coherent pieces of software development methods. Several views on the notion of MFs have been proposed, including method components (Wistrand and Karlsson, 2004), method chunks (Ralyté, 2004) and MFs (Brinkkemper, 1996) (note that we employ the term MF the class of

building blocks for ME as well as a specific building block). This research is based on the latter approach, which implies a process-oriented perspective with MFs described in terms of activities (process fragments) and their deliverables (product fragments).

cf. Method Adaptation (Figure 1.1)

Related to the MF concept is that of the method increment (van de Weerd, Brinkkemper, and Versendaal, 2007). Method increments are defined as collections of changes—including insertions, removals, and modifications (van de Weerd, Brinkkemper, and Versendaal, 2007)—of MFs at the smallest level, i. e. at the level of single activities, deliverables, and relationships, between two points in time. These changes can be derived from MFs by decomposing them into their smallest subunits. As such, method increments form a core concept in describing (and prescribing) method evolution.

Using method increments, we are able to gradually change methods. The gradual adaptation of methods is the core of the research presented in this dissertation. This is depicted in the conceptual diagram (Figure 1.1) through a series of method increments that result in discrete versions of the method, also called method snapshots. Snapshots are indirectly composed of multiple MFs, assembled and configured in such a way that they form a consistent method. Each snapshot represents a usable version of the method that can be enacted within an organizational process.

cf. Method Enactment (Figure 1.1)

Method enactment is the final step within the scope of this research. At this stage, changes to a method are translated to a running process, resulting in the end result of a process improvement effort. It includes, amongst others, the configuration of tools and the generation of deliverable template. Experience gained from this step can be fed back into the method base.

1.5 RESEARCH OBJECTIVES AND QUESTIONS

The research described in this dissertation aims to investigate the characteristics of process improvement efforts by looking at their process (how to change?) rather than their content (what to change?). The main objective of this research is to support process improvement efforts based on the idea of incremental improvement. This support is to be realized through the application of automated techniques that combine method knowledge, historical data, and contextual factors into enactable improvement suggestions.

Design and develop a knowledge-centric software system to support incremental process improvement in software production.

Main Research Objective

The main research objective naturally results in three sub-objectives, where we first explore the challenges of Software Process Improvement in an industrial environment, after which we develop techniques to deal with some of these challenges, and finally propose a knowledge-centric environment that integrates these techniques into a tool. Each of these objectives is then actualized through a set of research questions.

Study the practices of software process improvement in the domain of software production.

Research Objective A

The first research objective is to investigate the process by which software development and software product management methods are introduced into software organizations, by studying several instances of such efforts from various perspectives. The design of an adequate solution that satisfies the main research objective requires a thorough understanding of process improvement efforts. We are particularly interested in the steps that are taken during an improvement project, the changes that are made during each step, and the results of these steps.

How can the Scrum development method be adapted to the context of SPM processes?

Research Question A.1

We first analyze the adaptation of an existing method, in this case Scrum (Schwaber, 2004), to an alternative situation. Scrum is originally a software development process model, with a strong focus on the practices of the SE teams. Literature provides little insight in the integration of other aspects of the software development process, such as software product management, into the approach. For this reason, we study the modifications required to achieve this result. This results in the description and evaluation of a specialized method description.

According to which paths can Scrum be implemented?

Research Question A.2

The Scrum development method is an interesting unit of analysis for studying the implementation of process improvements. Scrum consists of

several discrete process and deliverable components. These components are to some extent interrelated, but many variations of the method exist, suggesting that not all components are strictly required and that different configurations are possible. We hypothesize that organizations can choose from different paths when confronted with the implementation of Scrum, ranging from a big bang approach (the entire method at once) to a gradual approach using multiple multiple method increments (carefully selecting applicable components and the order in which the are employed).

Resarch Objective B

Develop foundational techniques for tool-supported software process improvement.

This research starts from several existing theories derived from earlier work. These theories include the notion that (i) methods can be dissected into MFs (Brinkkemper, 1996); (ii) the process of method evolution can be dissected into method increments (van de Weerd, Brinkkemper, and Versendaal, 2007); and (iii) method increments can be used to guide method construction (Brinkkemper, van de Weerd, Saeki, et al., 2008). However, these assumptions have not been thoroughly tested and explored, as little empirical research is available. Therefore, the second research objective is to further investigate the notion of method increments, i. e. collections of coherent, atomic method changes (van de Weerd, Brinkkemper, and Versendaal, 2007), in the context of tool-supported process improvement, by developing foundational techniques that focus on the historical aspect (method increments as a description of previous process improvement efforts) and the planning aspect (method increments as building blocks for gradual method adaptation).

Research Question B.1

How can multiple increments that are part of a larger evolutionary process improvement be documented effectively?

In order to be able to meaningully reason based on previous experience with the introduction of specific method fragments, we need to collect historical data that gives us insight into specific combinations of contextual factors, implemented method fragments, the implementation process (i. e. the steps), and the outcomes of these steps. Throughout the case studies, a pattern for describing this data has emerged, which we solidify through the design of a template.

How can we determine the most appropriate path for implementing process changes, based on the constraints of the stakeholders? — *Research Question B.2*

Within the ME domain, several strategies for obtaining new MFs have been proposed, including method assembly (Brinkkemper, Saeki, and Harmsen, 1999) and method configuration (Gupta and Dwivedi, 2012). These approaches focus mainly on combining and adapting existing methods for use in a specified context. While such approaches provide an end-point, i. e. a goal method, it remains unclear how an organization can arrive at that end-point. For that purpose, we need a more fine-grained approach that allows the derivation of multiple increments that, when implemented consequetively, allow an organization to alter its processes gradually.

Our goal is to employ an automated planning tool to derive all possible implementation scenarios based on the hard constraints within a target method. These hard constraints include dependencies between deliverables and the available resources within the organizations. This set of scenarios is then sorted and filtered based on a set of soft requirements, such as dependencies between organizational capabilities, to obtained a list of optimal and near-optimal implementation scenarios.

Design and develop a system that supports process improvement efforts based on the notion of incremental process improvement. — *Research Objective C*

The final research objective is to apply the knowledge resulting from the previous two objectives by designing and developing a system that supports process improvement efforts based on the notion of incremental improvement. The resulting system is a synthesis of the research described in this dissertation, existing methods, domain knowledge, and contextual factors, and can be used to obtain concrete process advice.

What are the objectives for a knowledge centric software system supporting incremental process improvement? — *Research Question C.1*

The design process requires a thorough understanding of the environment in which the system would be used. We should “infer the objectives of a solution from the problem definition and knowledge of what is possible and feasible” (Hevner and Chatterjee, 2010). Important at this stage is to

create an understanding of how our solution will support the problem of incremental process improvement. Such understanding serves two purposes. On the one hand, it provides justification for the design of such a system, by uncovering issues and shortcomings within the current state of art. On the other hand, it provides us with a set of design guidelines and restrictions that need to be taken into account during the design process.

Research Question C.2 *What is an adequate functional architecture for a knowledge centric software system supporting incremental process improvement?*

After characterizing the objectices, we are able to further elaborate the functional aspect of our design. According to Nunamaker Jr., Chen, and Purdin (1990), the initial phase during system development consists of the construction of a conceptual framework. In addition to this, one should investigate the system functionalities and requirements, understand the system building processes and procedures, and study relevant disciplines for new approaches and ideas.

At this stage, we need to integrate our understanding of the process of incremental method engineering with our knowledge of the current problem space, and enhance our design accordingly. This integration encompasses a specification of the major functional components within our design. We also demonstrate how these components are to be applied in a real life setting, and what the deliverables throughout the process look like. With this functional architecture, we aim to show the applicability of our design to the problem of incremental process improvement.

Research Question C.3 *What is a feasible technical architecture for a knowledge centric software system supporting incremental process improvement?*

To increase the usefulness of our design, we also need to elaborate our functional architecture into a technical design. This elaboration is performed according to Kruchten's 4+1 architectural view model. This model includes both static as well as dynamic components of our architecture.

1.6 RESEARCH APPROACH

The objectives described earlier have a direct impact on the research strategies that are appropriate to achieve those objectives. The research process starts out with several empirical studies of process improvement at Dutch software producing organizations, after which we build on this empirical data to first design techniques that guide organizations during process improvement efforts, and later implement and evaluate these solutions. Although there is overlap in terms of the research strategies applied during the three phases, the nature of the phases changes as we move closer to the final objective, from an exploratory approach to a design approach.

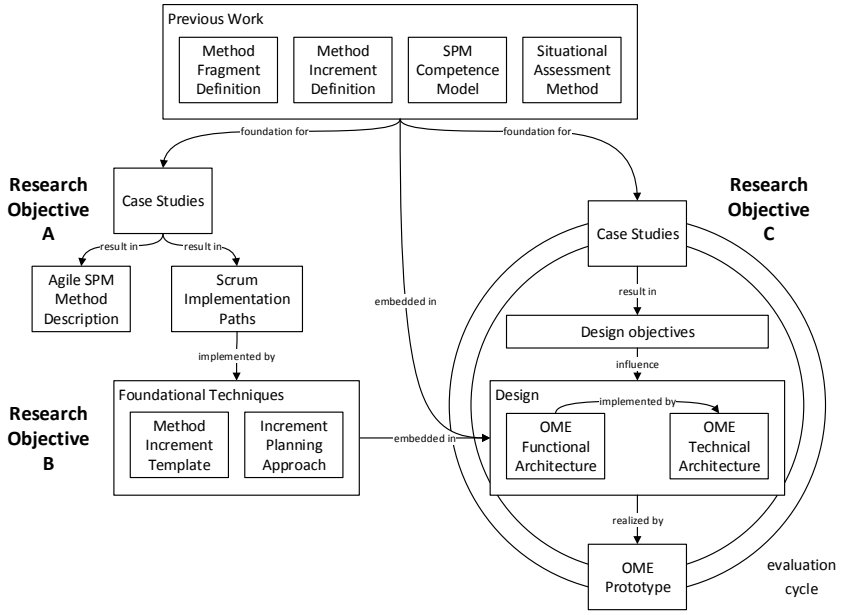


Figure 1.2.: Overview of our research approach

Study the practices of software process improvement in the domain of software production. *Research Objective A*

We take a predominantly exploratory stance during the first phase, using case studies as the main research technique. Case studies have become increasingly accepted as an approach to obtain both an exploratory insight

as well as an explanatory one (Runeson and Höst, 2008). The results from such case studies are highly detailed and provide deep insights into the internal mechanics of the unit of study. They inherently incorporate multiple strategies to data gathering, including document analysis, interviews, and desk research. It is for these reasons that the case study is the main research approach within the first phase of this research.

CHAPTER	RESEARCH METHODS
Chapter 2	Single case study
Chapter 3	Multiple case studies

Multiple approaches are available to obtain and analyze the data at this point, including a survey, a systematic mapping study, or even an exploratory experiment. It is pre-eminent at this point to consider the context in which the research takes place. Although we move towards more general results throughout the research, we start from the specific, in this case the SPM and agile domain. Both of these domains, but mainly the first one, are not yet well understood and a large variety of techniques has been proposed throughout the past decade. In addition, it is unclear how changes to these processes relate to theory on CM and BPM.

Furthermore, researchers in industry have little to no control over the process changes that they are studying. Changing these processes implies a large impact on organizational knowledge and on the professional well-being of employees, introducing a lot of risk and requiring a substantial amount of resources. This nearly rules out the possibility of performing a realistic, useful experiment, or of taking an action research approach.

Within the limitations of access to the unit of study, i. e. process changes, surveys are an adequate tool of gathering data. In addition, they have the benefit of allowing many data-points to be obtained. This would potentially lead to a higher reliability and generalizability of the research. However, at this stage, generalizability is not the primary goal. Instead, we want to create more insight into the specifics of process improvement in the domain of SPM and agile development.

Develop foundational techniques for tool-supported software process improvement.

Research Objective B

The initial exploratory phase provides us with a baseline for the second phase, in which the design aspect of this research becomes more apparent. The goal of this phase is to obtain deeper knowledge of the structure of small changes within a process improvement effort. We take a design stance to approach this problem, instead of analytical alternatives, such as explanatory case studies or experiments. This design stance allows us to make assumptions about the small process changes, or method increments as we will call them, and test these assumptions in a pragmatic fashion. These assumptions are based on earlier research, including the research performed during the first phase.

CHAPTER	RESEARCH METHODS
Chapter 4	Single case study / Design research
Chapter 5	Design research

Design science has not been without its controversy. It is originally deemed appropriate only in an exploratory context. More recently, scientists have refined the method, and the role of design science for validation and testing purposes has increased (Hevner, March, Park, et al., 2004; March and Smith, 1995). Peffers, Tuunanen, Rothenberger, et al. (2007, p. 47) argues that "without a strong component that produces explicitly applicable research solutions, IS research faces the potential of losing influence over research streams for which such applicability is an important value". We agree strongly with this statement, and argue that a combination of theoretical and practical research contributions is required to make an impact on the practices that we are studying. With the development of the techniques presented in Part III, we aim to provide this theory-grounded, practical contribution.

Design and develop a system that supports process improvement efforts based on the notion of incremental process improvement.

Research Objective C

The third phase consists of a combination of empirical science, in the form of exploratory research, and design science. The empirical aspect con-

sists of the case studies that were performed in order to refine the proposal of the Online Method Engine (OME). The design aspect is reflected in the proposal of a functional and technical architecture of the OME.

CHAPTER	RESEARCH METHODS
Chapter 6	Multiple case studies / Design research
Chapter 7	Design research
Chapter 8	Design research

To guide the design science activity, it is important to adhere to several design science guidelines as defined by Hevner, March, Park, et al. (2004). To make sure that all guidelines are adhered to, these should be grounded within an adequate research model. Several models for IS research have been created during the past decade. One of the most used models is the information systems research framework by Hevner, March, Park, et al. (2004). This conceptual framework captures the key concepts required for understanding, executing, and evaluating IS research combining behavioral-science and design-science paradigms.

Although this model allows one to clearly position his or her research, it lacks clear guidance for the steps that need to be performed in a valid design-science research project. A model more suited for this purpose is the systems development research model by Nunamaker Jr., Chen, and Purdin (1990), aimed at providing guidance during the entire development lifecycle of an innovative system. Unfortunately, this model aims only at the development aspect of a research project, and therefore lacks several items. The design science research methodology by Peffers, Tuunanen, Rothenberger, et al. (2007) fills these gaps. Although this model also contains steps related to design, development, and evaluation, it is not as detailed as the systems development research model. However, it includes several very important steps: problem identification and motivation, definition of objectives, and communication.

In order to obtain a research method that encompasses all stages, we have combined the two models presented above into one, displayed in Figure 1.3. By producing all deliverables as presented on the right, this research project adhers to the major design science research guidelines and produces valid and usable results.

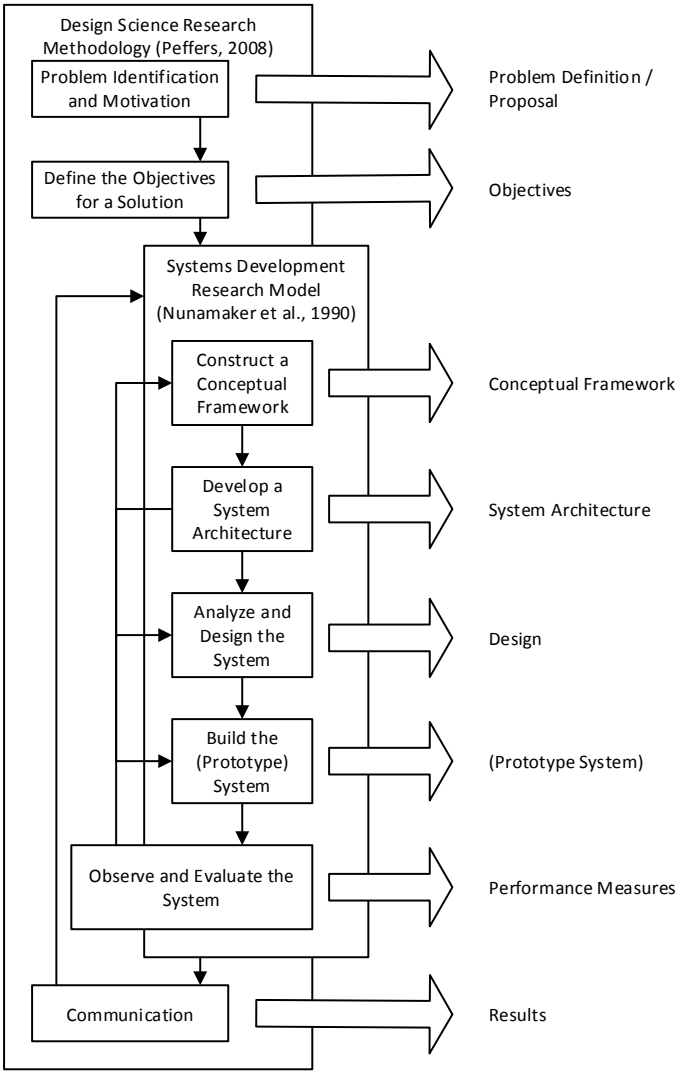


Figure 1.3.: Design science research approach

1.7 STRUCTURE OF THIS DISSERTATION

The structure of this dissertation reflects the three objectives described above. Each part focuses on one objective, and contains two or more articles relevant to that objective.

In Part II, we describe two studies that are used to build a thorough understanding of process improvement efforts. Both of these studies are performed in the domain of agile software development and SPM.

Chapter 2 describes a case study, resulting in a method description for agile SPM. The method integrates agile concepts with SPM related activities by providing a structured approach to requirements abstraction, and several lessons for aligning agile software development and SPM processes. In addition, the study shows insight into the path that was followed to reach the final process. This research has been published in the journal of Information and Software Technology (IST) (Vlaanderen, Brinkkemper, Jansen, et al., 2011).

We expand upon this insight in Chapter 3, in which we analyze and compare the introduction of Scrum at four case companies. The results show several distinct implementation paths, ranging from gradual to disruptive, with distinct effects on effectiveness. This research has been presented and published at the International Conference on Product-Focused Software Development and Process Improvement (PROFES) (Vlaanderen, van Stijn, and Brinkkemper, 2012).

In Part III of the dissertation, we focus on the second objective by describing a more theoretical study of the notion of method increment.

Chapter 4 expands on the topic of knowledge gathering related to method increments by providing a detailed description and evaluation of a method increment template. This template can be used to capture changes in the organizational processes, with a focus on the path to those changes, and the outcomes. This research has been presented and published at the European Conference on System & Software Process Improvement and Innovation (EuroSPI) (van Stijn, Vlaanderen, Brinkkemper, et al., 2012).

In Chapter 5, we propose and evaluate a formalization of the PDD, specifically aimed at describing method changes, along with a planning framework and process for generating plans that comply with different types of constraints. This research is accepted for publication at the International Conference on Advanced Information Systems Engineering (CAiSE) (Vlaanderen, Dalpiaz, and Brinkkemper, 2014).

In Part IV, focusing on the third objective, we describe the design, realization, and evaluation of a Knowledge Management (KM) system that supports incremental method engineering. This system, OME, encompasses the dissemination of method knowledge, along with process assessment, improvement, and enactment. The design is elaborated in three chapters.

In Chapter 6, we describe the design choices that we made based on seven case studies. The case studies resulted in a set of lessons that formed that main input for a revision of the original proposal. This research has been published in the International Journal of Information System Modeling and Design (IJISMD) (Vlaanderen, van de Weerd, and Brinkkemper, 2012).

In Chapter 7, we provide a functional architecture of the OME. We describe each of the four main functional areas—knowledge dissemination, method assessment, method improvement, and method enactment—in more detail. The proposal is supported by a running example. This research has been published in the International Journal of Business Information Systems (IJBIS) (Vlaanderen, van de Weerd, and Brinkkemper, 2013).

Chapter 8 has a more technical perspective, and describes the technical toolset that realizes the OME using the 4+1 architectural model. Part of this toolset, related to the enactment of method increments in tooling, is further elaborated. Prototypes of both the toolset and the enactment mechanism are evaluated through several interviews. This research has been published in the International Journal of Information System Modeling and Design (IJISMD) (Vlaanderen, Dalpiaz, van Tuijl, et al., 2014).

Chapter 9 contains the conclusions that we draw based on the results of the research described in this dissertation. In the appendix, we have included a paper that demonstrates the current prototype of the OME. This work has been accepted for publication at the Forum of the International Conference on Advanced Information Systems Engineering (CAiSE Forum) (Vlaanderen, Spruit, Dalpiaz, et al., 2014).

Part II

IMPLEMENTING AGILE PROCESSES

2

THE AGILE REQUIREMENTS REFINERY: APPLYING SCRUM PRINCIPLES TO SOFTWARE PRODUCT MANAGEMENT

Although agile software development methods such as Scrum and Dynamic Systems Development Method (DSDM) are gaining popularity, the consequences of applying agile principles to Software Product Management (SPM) have received little attention until now. In this paper, this gap is filled by the introduction of a method for the application of Scrum principles to SPM. A case study research approach is employed to describe and evaluate this method. This has resulted in the 'agile requirements refinery', an extension to the Scrum process that enables product managers to cope with complex requirements in an agile development environment. A case study is presented to illustrate how agile methods can be applied to SPM. The experiences of the case study company are provided as a set of lessons learned that will help others to apply agile principles to their SPM process.

This work was originally published as:

Vlaanderen, K., Brinkkemper, S., Jansen, S., et al. (2011). "The Agile Requirements Refinery: Applying SCRUM Principles to Software Product Management." In: *Information and Software Technology (IST)* 53.1, pp. 58–70

2.1 INTRODUCTION

One of the major innovations in software development methodology of the last few years has been the introduction of *agile* principles. Since the creation of the Agile Manifesto in 2001, including the years leading to its creation, several agile software development methods have come into practice (Abrahamsson, Warsta, Siponen, et al., 2003). Examples of such methods are Dynamic Systems Development Method (DSDM) (Stapleton, 1999), eXtreme Programming (XP) (Beck, 1999), Feature Driven Development (FDD) (Palmer and Felsing, 2002) and Scrum (Schwaber, 1995). The strong points of such methods are that by employing them, the development process becomes more responsive to a changing environment, working software is chosen over extensive documentation, individuals and interactions are considered more important than tools and processes, and customer collaboration is valued more than contract negotiation (Beck, Beedle, van Bennekum, et al., 2001).

In the last few years, these agile methods have proven to be successful in a large number of cases. Companies that have put the agile method *Scrum* (Schwaber, 1995) into practice range from small companies as described by Dingsøyr, Hanssen, Dybå, et al. (2006) to large multinationals (Fitzgerald, Hartnett, and Conboy, 2006). Research has shown that the use of Scrum within a company can lead to significant benefits (Mann and Mauer, 2005), and that its use is not limited to local projects (Danait, 2005).

Due to the its success, demand for the extension of agile principles to other domains has risen. One such domain is *Software Product Management (SPM)*. The original area of product management can be defined as "the discipline and role, which governs a product (or solution or service) from its inception to the market/customer delivery in order to generate biggest possible value to the business" (Ebert, 2007, p. 1). SPM is then "the process of managing requirements, defining releases, and defining products in a context where many internal and external stakeholders are involved" (van de Weerd, Brinkkemper, Nieuwenhuis, et al., 2006b, p. 319). The topic of SPM touches upon several other areas. In the related fields of lifecycle management and release planning, several approaches have been proposed, including market-driven requirements engineering (Carlshamre and Regnell, 2000) and requirements interdependencies (Carlshamre, Sandahl, Lindvall, et al., 2001). A systematic review of release planning approaches has been given by Svahnberg, Gorschek, Feldt, et al. (2010). Another related field, requirements prioritization, has seen several publications in recent years, includ-

ing work on requirements prioritizing for product software (Berander, 2007) and distributed prioritization (Regnell, Höst, Natt och Dag, et al., 2001).

Due to the complexity of software products, with a large variety of stakeholders, long lists of requirements and a rapidly changing environment, SPM is a complex task. However, relatively little scientific work has been performed in this area. An attempt to close this gap has been provided by van de Weerd, Brinkkemper, Nieuwenhuis, et al. (2006b) in the form of a reference framework for SPM. Their work aims at providing a structure for the body of knowledge regarding SPM by identifying and defining the key process areas as well as the internal and external stakeholders, and their relations. Some recent work related to the specific areas of the framework includes the use of feature modeling for handling variability throughout the product line lifecycle (Schwanninger, Groher, Elsner, et al., 2009), and key success factors for pricing software products (Kittlaus and Clough, 2009). Another important addition to the field has been the Quality Performance (QUPER) model, developed for handling non-functional requirements (Berntsson Svensson, Olsson, and Regnell, 2008; Regnell, Berntsson Svensson, and Olsson, 2008).

Currently, little work exists regarding *agile* SPM. A case study describing the use of agile requirements engineering is described by Pichler, Rumetshofer, and Wahler (2006). However, the paper does not provide details regarding the agile requirements engineering process. An attempt to link long-term product planning and agile development is provided by Vähäniitty and Rautiainen (2008). Greer and Ruhe elaborate on agile release planning by providing an iterative optimization method (Greer and Ruhe, 2004). Collaboration between product managers and development teams in challenging environments, such as where no complete requirements are available, is investigated by Fricker and Glinz (2010). In a comparative case study by Dzamashvili-Fogelström, Numminen, and Barney (2010), a misalignment was identified between the agile principles and the needs of pre-project activities in market-driven development. They state that the differences between agile methods and the needs of market-driven software development may threaten product development by disabling effective product management.

In order to fill the remaining gap, we will describe in which way SPM can be performed in a Scrum development context. The research described in this paper proposes an agile SPM method based on Scrum, which improves the ability to handle large amounts of complex requirements in an agile environment. Furthermore, a case study was performed at a product software company located in the Netherlands that has worked with the agile SPM method for nearly two years. By

showing their experiences, a set of useful lessons learned is provided that aids in the implementation of Scrum-inspired SPM alongside agile software development at other companies.

Section 2.2 continues with a description of the proposed Scrum-inspired agile SPM process. In Section 2.3, an outline of the case study approach is given, including the research triggers, questions and methods, and a description of the validity threats. This is followed by an introduction of the product software company at which the proposed method has been used. In Section 2.4, the results of the case study are shown in the form of an analysis of the tasks within the process. The validity threats regarding this case study are shown in Section 2.3.3. To conclude, Section 2.5 contains a description of the lessons learned regarding agile SPM and Section 2.6 provides conclusions and suggestions for future research.

2.2 AGILE SPM

This section describes a method for applying agile SPM in product software organizations that work according to agile software development practices. The proposed method is based on the default Scrum process (Schwaber, 1995), developed initially for the purpose of software development. Section 2.2.1 gives a short summary of the Scrum development method, followed by the adaptations that have been applied to make the method applicable to SPM in Section 2.2.3.

2.2.1 *Scrum Development Method*

The Scrum development method was proposed in 1995 by Schwaber (1995), at a time when it became clear to most professionals that the development of software was not something that could be planned, estimated and completed successfully using the common 'heavy' methods. The Scrum method is based on the work of Pittman (1996) and Booch (1995), and adheres to the principles of agile software development.

Central to Scrum is the idea that many of the processes during development cannot be predicted. It therefore addresses software development in a flexible way. The only two parts that are fully defined during a software development project are the first and last phase (planning and closure). In between, the final product is developed by several teams in a series of flexible black boxes called 'sprints'. No new requirements can be introduced during these sprints. This ensures that the

final product is being developed with a high probability of success, even within a constantly changing environment. This environment, which includes factors such as competition, time and financial pressure, maintains its influence on development until the closure phase.

The *backlog* is an important instrument in the Scrum process. The following backlogs play a part in Scrum development:

- *Product Backlog* - The Product Backlog (PB) is central to the Scrum method. The PB contains a prioritized list of all items relevant to a specific product. This list can consist of bugs, customer requested enhancements, competitive product functionality, competitive edge functionality and technology upgrades (Schwaber, 1995). Once a requirement has been fully specified, with the approval of a developer, the requirement can be copied from the PB onto the Development Sprint Nacklog (DSB).

- *Development Sprint Nacklog* - Each team that participates in the software development process maintains its own DSB. All requirements that are assigned to the development team at the beginning of a sprint are put on their DSB. Every requirement is decomposed into several tasks, which are then assigned to specific team-members. The development sprint backlog is fed by the product backlog with items that have been fully specified.

The DSB enables continuous monitoring of the progress of developers and development teams, while the PB enables monthly renegotiation about the priorities for each requirement.

2.2.2 *Agile SPM*

The development of software by large teams of developers requires a steady flow of elicited product requirements. Without this steady flow of requirements, software vendors run the risk of delaying new software releases and bad code due to badly specified requirements, all resulting in the waste of large amounts of resources. To avoid these problems, a functioning team of product managers is required, that can, cooperatively with the development team, supply approved and well-defined requirements. The agile SPM method applies Scrum to maintain a steady flow of new requirements for the DSB. Furthermore, agile SPM enables a software vendor to flexibly define requirements according to a pre-defined procedure. The pre-defined procedure forces a software vendor to explicitly manage the

lifecycle of a requirement, leading to better-defined requirements. Simultaneously, the process remains agile, i. e. some requirements can be defined and implemented quickly, while others move through their lifecycle at a regular pace.

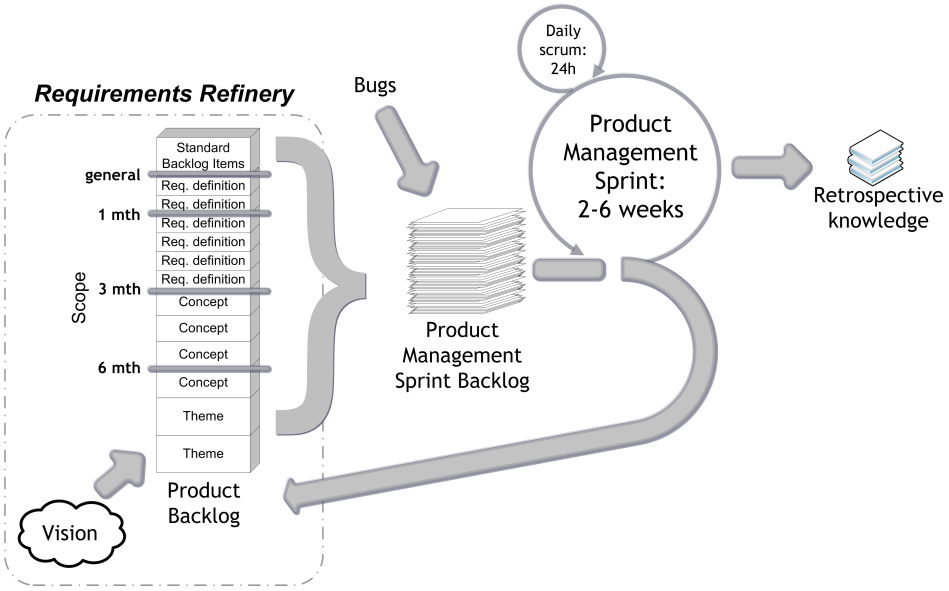


Figure 2.1.: Agile SPM knowledge flow

Figure 2.1 shows the flow of knowledge within the agile SPM process. The figure is based on the default Scrum development process described in the previous section, and is supplemented with SPM -specific adaptations. In the figure the Product Management Sprint Backlog is introduced. The Product Management Sprint Backlog (PMSB) is an agile SPM concept. It provides product managers with a way of working similar to that of developers in the Scrum process, using PMSB items to establish division of work, and work planning.

- *Product Management Sprint Backlog (PMSB)* - The PMSB contains all items that need to be completed within the sprint by each product manager. The PMSB is fed by with items from the product backlog, the full list of concepts, themes, and requirements for a product. The PB feeds the PMSB with items that need further specification before they can enter the DSB.

Scrum and the agile SPM process are similar in the aspects that they both work in sprints, and that both developers and product managers perform tasks

according to the shared PB and a team backlog. The main difference is that at the end of a sprint, developers produce a working version of the software, whereas product managers produce clearly specified requirements. Once these well-defined requirements are approved and prioritized by the product owners, the requirements are put onto the DSB. Table 2.1 lists the differences between the agile SPM process and Scrum.

	PMSB	DSB
Takes work from..	Product Backlog (PB)	Product Backlog (PB)
Demands..	vision (unspecified requirements), bugs	specified requirements
Supplies..	specified requirements	functional software
Deals with..	visions, concepts, themes, requirements	bugs, product enhancements, functionality, technology, upgrades, etc.
Works in..	sprints and daily scrums	sprints and daily scrums
Worked on by..	product managers	developers
Puts back onto PB..	requirements definitions	finished PB items

Table 2.1.: Differences between Scrum development and agile SPM

The input for the agile SPM process is in most cases an idea or a wish for new functionality, but also new technologies, bugs and upgrades. This idea enters the process in the form of a *vision*, shown by the cloud at the bottom left of the figure. During a number of sprints, this vision is then refined several times, going through the agile requirements refinery, which will be discussed in the next section. Based on the amount of time available and the focus determined by the board, the SPM teams select a set of PB items, such as concepts and visions, and place them on their PMSB. During the length of a product management sprint, specification tasks are performed on these PB items. The main result of this process is a list of further specified themes, concepts, and requirements that can be placed back onto the PB. The requirements that have been fully specified and approved by a software developer are candidates for the next development sprint. As a result of sprint review meetings held at the end of each sprint, new (retrospective) knowledge is gained that can help to improve the process.

The agile SPM process also includes bugs from earlier versions. These form an alternative way of generating PB items and do not follow the usual path through the requirements refinery. Instead, they are placed directly on the PB. If the bug can be fixed easily, it goes straight to the DSB. If the bug cannot be fixed easily, however, it will go onto the PMSB, for review and further detailing by the product management team.

Each working day, also known as a *Scrum*, starts with a Scrum meeting, during which the previous day is discussed. As this session is primarily meant to improve the productivity and the effectiveness of the SPM team, a small set of possible improvements is discussed. This helps avoiding experienced problems in the future. The end-result of an agile SPM sprint consists of the requirements definitions, which can in turn be used by the development teams. The sprint length is equal to the length of development sprints, in order to synchronize the heartbeat of the product management and the development process.

There are three stakeholders in the agile SPM process. First and foremost, the *product managers*' work process is the one determined by the agile SPM process. Secondly, the *product board*, consisting of the CEO, the support director, the business consultancy director, the development director, and several representatives from sales departments, determines requirements priority and product vision in a monthly meeting. The *development teams* increasingly monitor and approve requirements as they come closer to entering the DSB.

2.2.3 *The Requirements Refinery*

The structuring of the workflow into sprints and scrums enables agile SPM dealing with customer wishes. Similar to the Scrum development method, no new items can be added to the PMSB, as it has been finalized at the beginning of the sprint. This means that the SPM team(s) can focus on the work at hand without disruptions. On the other hand, it also requires considerable thought about the structuring of specific tasks, since they need to be completed within the timeframe of one sprint. SPM tasks, however, are not easily restructured into fine grained tasks of up to one month. For this reason, the default Scrum-approach to task management has been substituted by the more fine-grained approach that is described in this paper.

This approach, the *Agile Requirements Refinery*, provides a solution for managing complex requirements. The approach is suited to the characteristics of SPM

tasks, and it resembles an industrial refinery in a way that during each sprint or iteration work is being performed on the requirement definitions that appear on the PB, to refine them from coarse-grained to fine-grained. Each refinement, from one stage to the next, can generally be performed within one month. When this is not possible, the item is placed back on the PB to be picked up again in one of the future sprints. Structuring the SPM tasks in such a way results in backlog items with a smaller granularity, suited for the length of a sprint. By refining complex requirements according to the abstraction levels of the requirements refinery, structure is added to the backlog that will help in completing the tasks in an effective manner.

Since Scrum itself does not provide guidelines for effectively managing large amounts of requirements of different granularity, a set of stages is introduced. Within the agile requirements refinery, a product functionality vision will generally move through these stages, during which it is refined with details and specifications. The stages are:

- *Vision* — A vision is the starting point for the lifecycle of most requirements. It is an idea, brought up by the company board, a customer or any other stakeholder, and is defined in generic terms. Once the idea reaches a product manager, he or she then converts it into a (set of) theme(s). An example of a vision is the wish to target small enterprises as potential customers for an ERP software package with a light version.

- *Theme* — A theme is the formal elaboration of a vision, describing it in more detail. The product manager defines the envisioned purpose of the new functionality, the business value of the theme, and the involved stakeholders. A theme should briefly describe the business problem from which it originates and the main issues that fall within the theme scope. This can where possible be extended with a set of provisional requirements. In total, a theme description should not exceed one page of text, in order to maintain clarity. The previously described vision can for instance be translated to the theme ‘small enterprises’, describing its importance and what would be required to accomplish it. In reality, a vision is often so complex that it can be refined into multiple themes. To ensure the technical feasibility of a theme, it is reviewed by the development teams.

- *Concept* — Themes are broken down into smaller pieces called concepts. A concept is a high-level focal point within the theme, consisting of a set

of solution stories that can later be used to deduct detailed requirements. The elaboration of each concept results in a document describing product drivers, product constraints and the concept scope. The description should briefly explain the necessity of the concept, while remaining clear and detailed enough to be useful for the definition of detailed requirements. The 'small enterprise' theme could for instance be converted to a set of concepts such as 'productX Lite', describing the high-level requirements of a product suited to the needs of small enterprises. Each concept definition should be checked with the software architects. Also, the developers help estimate whether the concept is sufficiently defined to further split up the concept into requirements.

- *Requirement definition* — The detailed definition of requirements is performed in three steps, of which only the first one is performed by the SPM team(s). SPM translates the concepts into a list of requirement definitions without going into a lot of detail. Requirement definitions consist up to this point of a description, a rationale and a fit criterion. The latter describes a constraint that must be met in order for this requirement to be successfully implemented. To ensure feasibility and compatibility with other requirements, each requirement definition should be checked with architects, functional designers or lead developers.

  After the initial high-level requirement definitions have been determined based on the previously defined concepts, the software development teams then elaborate these into requirements containing a detailed description of some desired functionality, described in sufficient detail to work with. To accomplish this, each requirement definition is first processed during a development sprint by a development team, to ensure that they are feasible, consistent and understandable in a general manner. Then a second pass is made, where the development team ensures requirement clarity, so that each requirement is understood by all team members. This results in a list with all relevant requirements and their detailed descriptions, including any necessary diagrams, technical specifications or otherwise necessary information that is required for the implementation of the requirement.

With smaller topics, the definition of a vision and a theme might not be necessary, in which case the problem can be placed within an existing theme or concept. They are then elaborated without constructing a vision, theme and/or concept, or

they are elaborated with the vision, theme and concept constructed afterwards. In other words, the requirements refinery is not restricted to a top-down approach, but can also be used bottom-up. This is similar to the approach by Gorschek and Wohlin (2006), who identified four abstraction levels on which a requirement can be placed, along with both a bottom-up and a top-down path along these levels. Also very similar, but with a chief aim of maintaining relationships between high-level and low-level goals during development, Vähäniitty and Rautiainen (2008) identified the stages vision, business goal, iteration goal, backlog item and task.

2.2.4 *Process Description*

Figure 2.2 visualizes the agile SPM process, based on the process-deliverable diagramming technique by van de Weerd and Brinkkemper (2008). In the figure, the deliverable side has been omitted in order to focus on the process aspect. Its notation is based on a UML activity diagram. Standard activities and sub-activities are depicted by white boxes, and open activities are shown by gray boxes. Arrows are used to show the control flow from one activity to the next. The top part of the activity diagram, indicated by a light gray box, will recur several times within each SPM sprint, once for each requirement.

At the start of each sprint, each SPM team has to *prepare its PMSB (1)*. The teams make a selection of PB items, of which they think that they can be completed within the upcoming sprint. This activity is similar to the sprint preparation as performed by the development teams.

The next step is to proceed with either *refining the items that are on the PMSB (2)*, or *introducing new ideas (3)* obtained through customer support, meetings with business consultants, customer sessions, industry analysts and involvement at different types of forums in which market parties are active. During a sprint, each item is refined from its current stage to the next level of detail, i. e. from vision to theme or from concept to requirement definition.

When a vision enters the process, it is described globally, after which one or more *themes are derived (4)* from it. Each theme is described according to the specification in Section 2.2.3. When the description is finished, the required investment needed to implement the team is estimated. Themes are then reviewed by the development teams, after which they are placed on the PB.

Concept specification (5) starts with breaking down the theme into a set of concepts. Every concept contains a set of solution stories which are used for

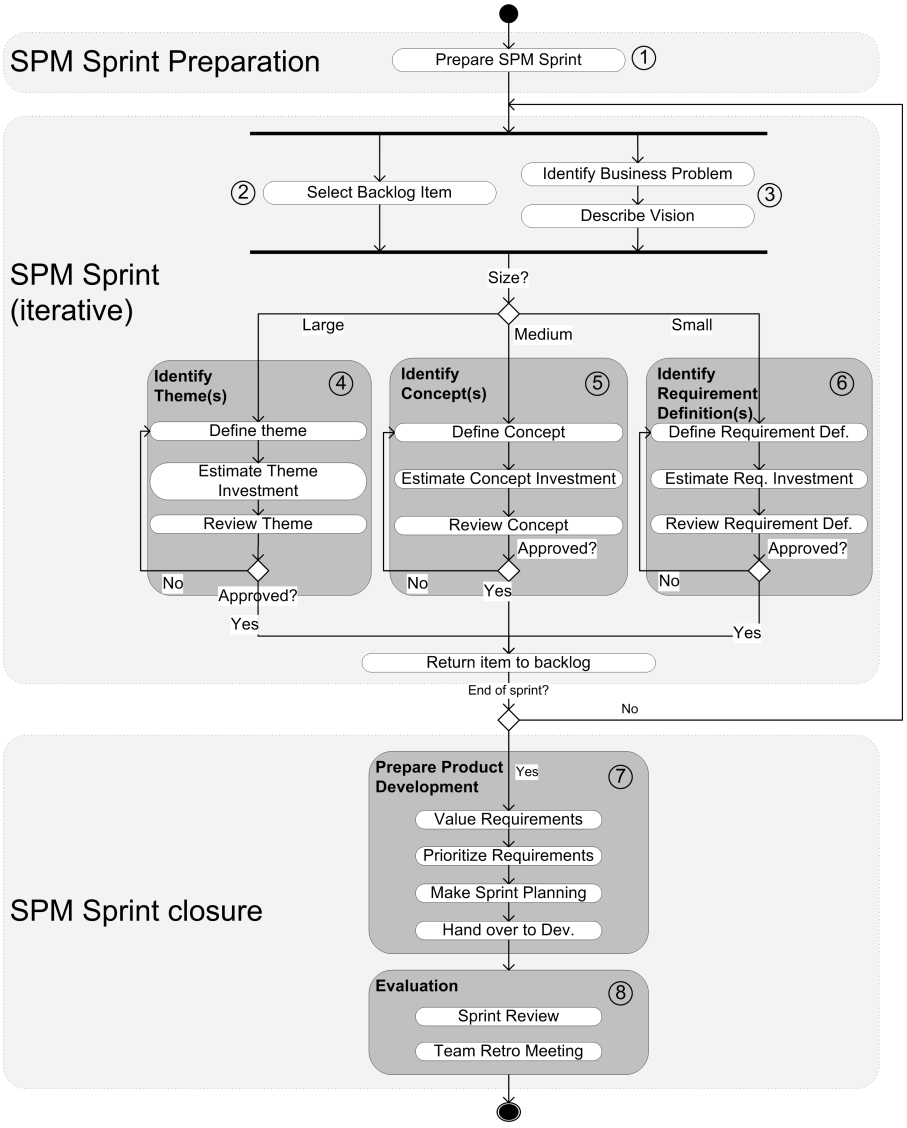


Figure 2.2.: Scrum SPM Process

defining detailed requirements. The concepts are defined by product managers and software architects. Again, an estimation is made regarding the required investment for implementation. After concepts are defined, they are reviewed by software architects and domain experts.

If a concept is approved, the concept is *broken down into requirement definitions (6)*. A requirements engineer and a Scrum development team are responsible for the definition of requirements. Requirements can be broken down into smaller pieces to fit into a sprint. They are also assigned a priority by the product board and the sales department, after which they are put on the PB. The highest rated requirements are to be developed first. The requirement definitions are reviewed by lead developers, architects, functional analysts and domain experts. In some cases, requirement definitions are rejected due to being unclear or not being described in sufficient detail. In such cases, the requirement definition needs to be further specified.

When requirement definitions are approved, *the costs and business value are calculated (7)*. Each requirement is valued and prioritized, after which the ordered list is placed on the PB, where it is used for deciding when features will be developed. After prioritizing the requirements, time is allocated to each requirement or concept to allow the determination of a sprint planning. When requirements are clear and have enough detail they are assigned to the development teams. The requirements are then placed in the DSB of the specific development team.

At the end of each completed SPM sprint, an *evaluation (8)* takes place, during which each team looks back at the last sprint, discussing about the aspect that went good or wrong. The results are written down, and from the resulting list, two or three items are chosen to be put on the sprint backlog of the next SPM sprint. This enables the teams to gradually improve the process, learning not only from their own mistakes, but also from those of the other teams.

2.2.5 *SPM Sprint*

The agile aspect of the proposed SPM approach lies mainly in the fact that, like software development, the SPM task is performed according to sprints with a fixed length of two to six weeks (varying per company). When the SPM sprint is performed simultaneously with the development sprint, the deliverables from one team are not always available in time for the other team's new sprint.

Therefore, sprints should not be performed synchronously to the software development sprint. Instead, they should be shifted back half of the development sprint duration. This ensures that the DSB is always up-to-date and ready for use once the software development sprint starts, reducing the time between the inception of a requirement and its realization in the product. Also, information regarding

implementation progress and the accuracy of requirements sizes and descriptions can flow back from the development teams to the SPM teams.

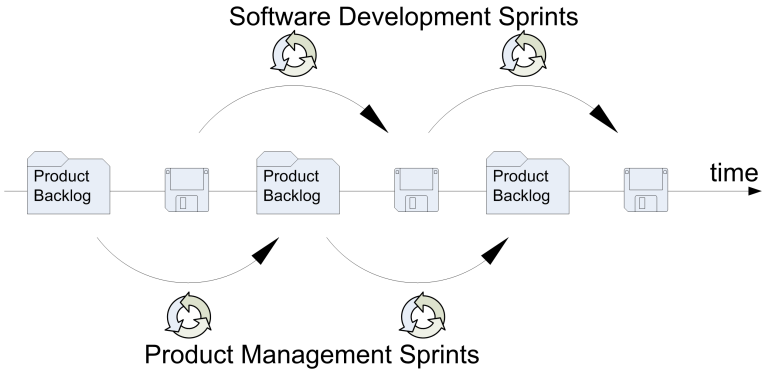


Figure 2.3.: Alternating sprints

Figure 2.3 illustrates this concept of alternating sprints. The horizontal timeline shows the synergy between SPM and software development, by switching from a focus on SPM to a focus on software development and back. The SPM team(s) deliver(s) an updated PB while the development teams are developing the next product release candidate (depicted by a floppy). Based on this release candidate, SPM will then redefine the PB, resulting in continuous double-loop feedback.

Similar to Scrum software development, the PMSB is filled with items from the PB at the beginning of each sprint. The status of completed, canceled or ongoing tasks is continuously kept up-to-date on the PMSB. Each product manager is responsible for keeping the backlog up-to-date as the sprint progresses. As an example of how this can be managed, an example excerpt of a PMSB is shown in Figure 2.4. The excerpt shows some of the tasks related to requirements management, together with their status, assignee and remaining hours. Based on the data in the backlog, a burn-down chart is created continuously to allow monitoring of the progress of the sprint.

2.3 CASE STUDY RESEARCH APPROACH

The main research question of this research is: *In which way can SPM be performed in a Scrum development context?*. This research question can be split into two subquestions. In the first part of this paper, we have answered the question: *how can agile concepts be applied to SPM?*. This research question is answered by

PMT: Sprint 2009-07				Day	1	2	3	4	5	6
Sprint Task Description	Topic	Status	Resp	Orig Est Hrs	14-jul	15-jul	16-jul	17-jul	20-jul	21-jul
Requirements Elaboration			-							
Document Management	Document Management	Not Started	-	0		0	0	0	0	0
Detailed Requirements	Document Management	Cancelled	PM#5	0		0	0	0	0	0
Meeting manager for Outlook	Meeting manager for					0	0	0	0	0
15. Detailed Requirements	Meeting manager for	Not Started	PM#5	6	6	6	6	6	6	6
Contract management	Contract management	Not Started	-	0		0	0	0	0	0
2.A Announcement letters	Contract management	Completed	PM#3	4	4	4	4	4	4	4
2.A Announcement letters	Contract management	Not Started	PM#4	4	4	4	4	4	4	4
2.B Invoicing & Payments requirements	Contract management	In Progress	PM#3	16	16	16	16	16	16	16
2.B Invoicing & Payments requirements knowle dge	Contract management	Not Started	PM#4	2	2	2	2	2	2	2
Maintenance management	Maintenance management	Not Started	-			0	0	0	0	0
4. Detailed requirements condition assessment	Maintenance management	In Progress	PM#3	24	24	24	24	24	20	20
4. Detailed requirements condition assessment	Maintenance management	Not Started	PM#2	16	16	16	16	16	16	16

Figure 2.4.: Example excerpt from a product management sprint backlog

developing a method for agile SPM. In the second part of the paper, we will focus on *what are the implications of introducing agile SPM in an agile development setting*. In order to answer this question, the method is tested in a case study (Yin, 2003) of a production environment in a product software company in the Netherlands. Data has been collected to answer the research question by means of:

- *Interviews* — The main research questions have been answered in part during the unstructured interviews with product stakeholders. We interviewed the chief technology officer, two product managers and one requirements engineer. These interviews were recorded, and information regarding the SPM process and issues related to it were extracted later on.
- *Document study* — The case company provided us with guideline documents such as the altered Volere requirements specification template (Robertson and Robertson, 1998) that is in use at the case company, the product backlog and the sprint backlogs for the SPM team. These documents were added to our case study database. Some of the filled-in Volere templates were used to gain understanding about the relation between the PMSBs. The PMSBs itself were used for a qualitative analysis to obtain further insight in the practical consequences of agile SPM and to extract some examples.

The changes in the SPM approach at the case company were mainly investigated in retrospect. The interviews that formed an essential source for this research

were performed at the start of 2009, at a point in time in which the case company already had several months of experience with agile SPM. Also, product management sprint backlogs up to that point were gathered. After this moment, the evolution of the approach was monitored by analyzing the product management sprint backlogs of the months after that. This has resulted in an overview of standard backlog items in Section 2.4.1 and an overview of relevant roles and tasks in Section 2.4.2. The requirements refinery is then illustrated with two examples in Section 2.4.3 and Section 2.4.4.

Based on the information from the backlogs and the interviews with the product stakeholders, we have derived the set of lessons learned that is presented in the final section. This list has been reviewed by the CTO at Planon to make sure that the most important items have been addressed accurately.

2.3.1 *Case Study Company: Planon*

The main contribution of this work lies in the description of a unique case among Dutch product software companies, and potentially among product software companies in general. The company at which the case study has been performed, Planon International (from now on referred to as Planon), has, as one of the first known companies, implemented an agile SPM process based on the agile principles in general (and the Scrum development method specifically).

Planon is an international software vendor that produces Facility Management and Real Estate management software for organizations (Integrated Workplace Management Systems). Founded in 1984, it currently has a customer base of over 1300, which is supported by more than 325 employees. The company's products are marketed through six subsidiaries, based in the Netherlands, Belgium, Germany, UK, India and the US, and a worldwide network of partners. The company made approximately 1.9 million profit with a revenue of 30 million in 2008. Planon develops client-server software (two-and three-tier architectures) with which it attempts to support the processes of facility management.

2.3.2 *Case Study Narrative*

Until 2004, product development at Planon was based on the Prince2 method, after which a switch was made to Scrum, thus following the ideas of the Manifesto for Agile Software Development. This means that working software is delivered

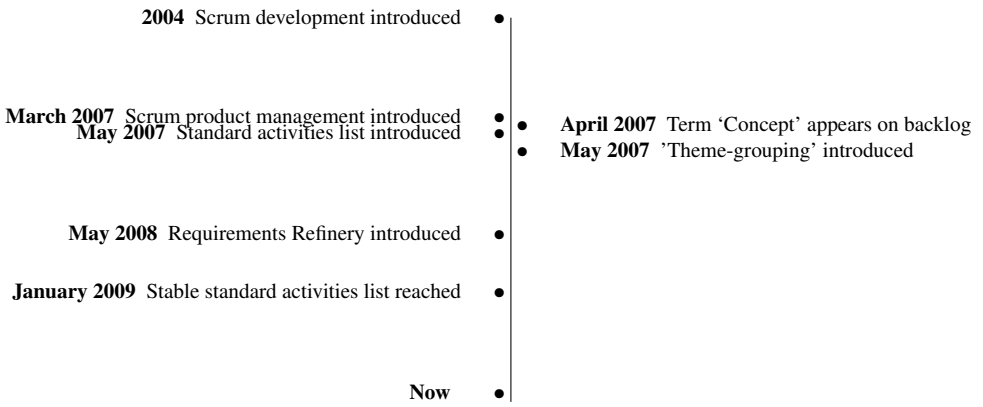


Figure 2.5.: Timeline of SPM process evolution

frequently, changing requirements are welcomed (even late in development) and teams reflect regularly on how to become more effective. Other agile ideas are that the course of a project unfolds in time, decisions are being made in a decentralized team-based way and there is a focus on early feedback.

Until 2007, as can be seen in Figure 2.5, agile product development was accompanied by non-agile product management. Although several stages of elaboration were employed, no fixed cycles were used. This has as an effect that product managers did not manage to provide development with sufficiently detailed requirements before the start of each sprint. To improve this, the question had to be answered whether it was possible to base the management of the product backlog on Scrum principles. This would implicate a continuous adaptation of the product backlog to changing circumstances and a changing environment.

From there on, a new categorization emerged on the product backlog. Initially, the terms 'concept' and 'theme' began to be used to group items according to the specific topic that they belonged to. Further elaboration of product features was displayed under a products-list within the PMSB. After several months, this approach consolidated into a stable approach, in this paper defined as the requirements refinery.

To better understand the implications of the SPM adaptation of Scrum, an analysis of the PMSBs was needed to gain a more detailed view of the results and implications of Planon's adaptations of the Scrum-process in order to ac-

commodate SPM. The data-set consisted of twenty-eight PMSBs, describing an equal amount of months. The PMSBs have been gathered from March 2007, when Scrum was introduced into the SPM process, until July 2009.

Sprint no.	*Period*	*# of Tasks*	*# of Hours at Start*	*# of Hours at End*	*Avg Hours per Task*	*σ Hours per Task*	*Avg Tasks per Person*	*σ Tasks per Person*	*Avg Hours per Person*	*σ Hours per Person*
Original SPM approach										
1	2007-03	45	449	154	10.0	n/a	9.0	n/a	89.8	n/a
2	2007-04	57	513	177	9.0	n/a	11.4	n/a	102.6	n/a
3	2007-05	56	527	147	9.4	n/a	11.2	n/a	105.4	n/a
4	2007-06	57	539	102	9.5	n/a	11.4	n/a	107.8	n/a
5	2007-07	112	409	179	3.7	n/a	22.4	n/a	81.8	n/a
6	2007-08	95	574	301	6.0	n/a	19.0	n/a	114.8	n/a
7	2007-09	81	550	215	6.8	n/a	16.2	n/a	110.0	n/a
8	2007-10	58	750	274	12.9	n/a	11.6	n/a	150.0	n/a
9	2007-11	53	544	188	10.3	n/a	10.6	n/a	108.8	n/a
10	2007-12	43	324	142	7.5	n/a	7.2	n/a	54.0	n/a
11	2008-01	60	648	193	10.8	n/a	10.0	n/a	108.0	n/a
12	2008-02	67	497	226	7.4	n/a	11.2	n/a	82.8	n/a
13	2008-03	65	604	250	9.3	n/a	10.8	n/a	100.7	n/a
14	2008-04	84	669	320	8.0	n/a	14.0	n/a	111.5	n/a
	(Weighted) Avg.	66.6	542.6	204.9	8.1		12.4		101.3	
	Std. Dev.	18.9	104.4	61.1	2.2		3.9		20.7	
Introduction of the Requirements Refinery										
15	2008-05&06	95	627	372	6.6	3.1	13.6	6.3	89.6	42.8
16	2008-06&07	83	614	202	7.4	3.0	11.9	4.7	87.7	25.2
17	2008-07&08	84	570	240	6.8	3.3	12.0	2.3	81.4	47.3
18	2008-08&09	96	446	209	4.6	2.9	16.0	4.3	74.3	25.9
19	2008-10&11	89	575	193	6.5	3.6	14.8	6.2	95.8	26.5
20	2008-11&12	81	577	116	7.1	2.6	13.5	4.3	96.2	19.9
21	2008-12&01	79	455	140	5.8	1.8	13.2	3.7	75.8	21.2
22	2009-01&02	85	582	173	6.8	2.2	14.2	5.2	97.0	27.8
23	2009-02&03	80	574	169	7.2	4.8	13.3	3.6	95.7	22.0
24	2009-03&04	90	735	192	8.2	3.2	15.0	2.3	122.5	20.2
25	2009-04&05	100	565	126	5.7	2.0	16.7	3.2	94.2	12.3
26	2009-05&06	95	517	216	5.4	4.2	15.8	4.5	86.2	16.5
27	2009-06&07	85	646	213	7.6	3.0	14.2	2.5	107.7	18.5
28	2009-07&08	69	421	167	6.1	1.82	11.5	3.3	70.2	30.7
	(Weighted) Avg.	86.5	564.6	194.9	6.5		13.9		90.9	
	Std. Dev.	8.0	80.9	60.0	0.9		1.5		13.2	

Table 2.2.: Analysis of the product management sprint backlogs

The analysis focused mainly on general statistics about the task structure, including task duration and workload per person, as well as on pattern discovery. Table 2.2 displays statistics about the sprints included in our study. From left to right, the table first shows the number of tasks that were placed on the PMSB in that month, the total amount of planned hours for those tasks and the amount of unfinished work at the end of the sprint. Subsequently, the table shows the average amount of hours per task, the average workload per person expressed in amount of tasks and the average workload per person expressed in hours. The bottom two rows show statistics about the average score and standard deviation for all items.

Furthermore, we have checked the backlogs for any anomalies. Any anomalies we found were either removed from the dataset, or further analyzed based on the information received from the CTO at Planon. We then grouped the backlog items according to their characteristics. These groups have been checked with the chief technology officer, to make sure that they were correct.

Over the two years of experience, the PMSBs provide information regarding the number of tasks and their characteristics. The PMSBs provide insight into two years' evolution of the number of tasks and their characteristics. First, several recurring, standard backlog items can be identified. Second, the evolution and introduction of the requirements refinery can be followed from the first introduction of themes, concepts and requirement definitions. The abstraction levels of the refinery (i. e. themes, concepts and requirement definitions) make complex requirements more manageable in an agile environment. To illustrate, two themes will be tracked through the entire SPM process, described in Section 2.4.3 and Section 2.4.4.

The switch to the requirements refinery in month 15 had clear effects on the backlog. Most notable is the immediate structure and clarity that is created by this change. By dividing the tasks related to the elaboration of requirements into lists named 'theme definition', 'concept definition' and 'requirement elaboration', a clearer overview of the workload is obtained. For every task it becomes instantly clear in what phase of elaboration the requirement currently is.

Another consequence of the approach can be seen in two trends in the evolution of task size and amount. On the one hand, the amount of tasks on the PMSBs increased approximately 25%, whereas the average size of the tasks decreased with approximately 25%. Evidence on the PMSBs suggests a relation with the introduction of themes and concepts on the PMSB, as larger tasks such as 'describe requirements' are now split into smaller tasks, specific to the current stage.

2.3.3 *Validity Threats*

In order to ensure the quality of our work, we have tried to adhere to four validity criteria for empirical research. The validity threats are construct, internal, external, and reliability threats (Jansen and Brinkkemper, 2007; Yin, 2003). Construct validity refers to the proper definition of the concepts used within the study. For this study, well established concepts from the area of agile methodologies and SPM were used to construct our theory on agile SPM. These theories were established in a discussion session at the beginning of the project. The terminology employed by the company was similar to that in standard Scrum literature, and concepts from the area of SPM corresponded to those posed by Xu and Brinkkemper (2007) and van de Weerd, Brinkkemper, Nieuwenhuis, et al. (2006b). Since well-known concepts were used to describe novel phenomena, construct validity is guarded. Furthermore, peer review was used to check whether the constructs were used correctly. The internal validity, which concerns relations between concepts, was threatened by incorrect facts and incorrect results from the different sources of information. Interviews were held with several people in order to cross-check documentation found and to confirm facts stated in other interviews.

With respect to external validity, concerning the ability to generalize the results, a threat is that this case is not representative for other software producers working with Scrum (Lee and Baskerville, 2003). Planon is a standard product software supplier, which deals with a lot of new requirements. The practices described in this paper can be a successful way to manage teams of product managers for similar sized software vendors working with Scrum. Finally, to defend reliability, similar results would be gathered if the case study was redone if the circumstances are at least similar (same interviewees, same documents, etc), due to the use of a case study protocol, structured interviews, and a peer-reviewed research process (Jansen and Brinkkemper, 2007).

2.4 SPM SPRINT BACKLOG ANALYSIS

2.4.1 *Standard Backlog Items*

From the PMSB, several standard recurring backlog items can be identified. The standard items, as opposed to incidental tasks, form a basic structure of recurring

tasks, mostly with the same amount of hours allocated each sprint. These tasks can be used to create a form of rhythm within the team(s).

At the case company, the list of standard backlog items has evolved during the reported period from a disorganized list into a stable list of tasks, shown in Table 2.3. On the left-hand side, all standard backlog items related to the SPM sprint are shown. On the right-hand side, all standard backlog items related to the development sprint are shown. All tasks are performed by the SPM team(s).

SPM RELATED	DEVELOPMENT RELATED
Prepare and Attend Product Board	Backlog Preparations
Sprint Review	Sprint Planning with Dev. Teams
Team Retro Meeting	Sprint Review with Dev. Teams
Team Allocation Overview	How-to-demo Stories
Problem and Change Management	

Table 2.3.: Standard items on the product management sprint backlogs

As described earlier, the PMSBs were at first mainly structured in a product-focused manner. As a result, recurring backlog items were spread across the PMSB, resulting in a disorganized list which had to be recreated from scratch every month. As of month five, a small list of recurring backlog items related to the product board is distinguished. However, this is comprised of only ninety hours. This list grows to a set of five different tasks (of which some occur multiple times, once for each product manager), with a total amount of 268 planned hours. This list stays relatively stable until month fifteen, in which the new PMSB structure is introduced. At that moment, the list of standard backlog items is reduced to two tasks with a total amount of 72 hours. Remarkable is the steady growth of this list in the next six months, after which the list of standard backlog items consists of five different tasks, similar to the tasks of the earlier months, totaling only 80 hours, which amounts to an average of 14.3% of the total amount of required hours each month. The final list of standard backlog items is shown on the left-hand side of Table 2.3.

The low amount of planned hours can be explained by taking into account the introduction of Scrum principles. At the same time as the introduction of the new PMSB structure, a new group of tasks has been introduced on the list, containing all the tasks related to the management of the upcoming development

sprint. Although the exact contents of the group differ every sprint, a large share of the tasks is recurring and thus added to the right-hand side of Table 2.3.

The following items are recurring on the product backlog:

- *Prepare and Attend Product Board* - The product board consists of several lead positions in the company, such as the CEO and the sales director, who have a major stake in the product itself. Once a month the product management team presents what has been developed and what the future will hold to the product board. The product board contributes in two ways. First, the product management team informs major product stakeholders of the progress of visions and plans that have an impact on the product. Secondly, the product management team is forced to report on their progress, which requires them to evaluate progress speed and SPM process quality.

- *Sprint Review* - The sprint review consists of a full review of the SPM sprint. Furthermore, the sprint review leads to an update of the internal and partner information portals of the product. These portals are used to report on the progress of the work and on the upcoming features for partners and sales teams.

- *Team Retro Meeting* - Once a month during the team retro meeting the internal functioning of the SPM team is discussed. The retro meeting does not specifically address practical problems, but tries to achieve better quality and use feedback to improve the agile SPM process.

- *Team Allocation Overview* - Throughout the agile SPM process, themes are assigned to teams, consisting of a product manager and a development team. Generally, teams will remain active within that theme. However, when a certain set of requirements that originates from a certain theme can also be implemented by a team that has resources available, requirements sets might be transferred from one team to another during the team allocation overview.

- *Problem and change management* - The task of problem and change management deals with customer problems and large changes that require the interference of a product manager. Furthermore, product managers go through the list of reported problems from customers and respond within one month. The response to a reported problem generally consists of a decline, i. e. the

problem will not be solved, or an accept, i. e. the problem will be inserted in the planning.

- *Backlog Preparations* - A basic structure needs to be provided before each sprint, with the appropriate names and task types. This only requires very little work.
- *Sprint Planning with Development Teams* - The sprint planning with development teams consists of an eight hour meeting. During these meetings product managers and developers negotiate, accept, and approve PB items for the DSB. This is a typical part of the Scrum process.
- *Sprint Review with Development Teams* - During the sprint review the development teams present the functionality they have implemented to the other development teams. Developers also defend why the functionality is necessary.
- *How-to-Demo stories* - Product managers create How-to-Demo stories for the developers who are working within their theme. These how-to-demo stories are specified to indicate to the developers how they should demo the functionality they have implemented during the sprint review. The main reason for the creation of these stories is that developers frequently have a different view of the interesting parts of the functionality they have implemented.

These activities provide an overview of the different standard tasks that are executed monthly by the SPM team.

2.4.2 *Roles and Tasks*

Although the identified recurring backlog items already form useful knowledge within a practical context, the link between the product management tasks and the actual execution of an agile SPM process is still missing. Therefore, we decided to further analyze the distribution of tasks over the product managers.

To do this, we analyzed the non-standard tasks per sprint, amounting to an average of 85.7% of the total allocated time per month. In doing this, we identified a set of roles within an agile Product Management team, each focusing on a specific set of tasks. Although no fully specialized roles were found, i. e. persons that handled

only one or two kinds of tasks, each role has a characteristic combination of tasks assigned to it.

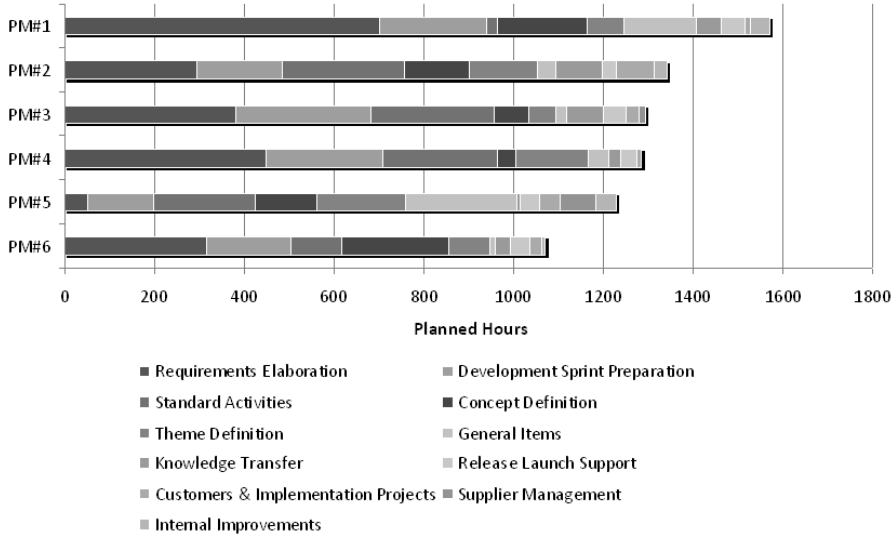


Figure 2.6.: Distribution of (non-standard) tasks over the product management team

Figure 2.6 displays the amount of time spent by each member of the product management team on the types of backlog items. The horizontal axis shows the planned hours, not the hours that were actually spent on the tasks. On the vertical axis, all members are displayed.

In Figure 2.7 we have displayed another visualization of the task distribution, complementary to that of Figure 2.6. First of all, the bar representing tasks related to 'requirements elaboration' is for a large share attributed to one person, PM#1. When compared to his bar in the previous figure, we can see that the majority of his time is actually spent on 'requirements elaboration', implying a position strongly focused on this area.

A second observation that we can make is that a large share of the high-level theme-definition tasks is performed by PM#5. Also, he spends little time on low-level activities related to 'requirements elaboration'. This indicates a position with more power and more influence on high-level decision making. Combined with the large share of 'general items' (referring to meta-activities such as process management, etc.) performed by PM#5, this indicates a management-oriented position.

Thirdly, when we look at the persons PM#6, PM#3, PM#4 and PM#2, we see a fairly regular pattern in the distribution of time spent on the activities 'theme definition', 'concept definition', 'requirements elaboration, 'standard activities' and 'development sprint preparation'. This seems to imply that the position of these persons is fairly similar.

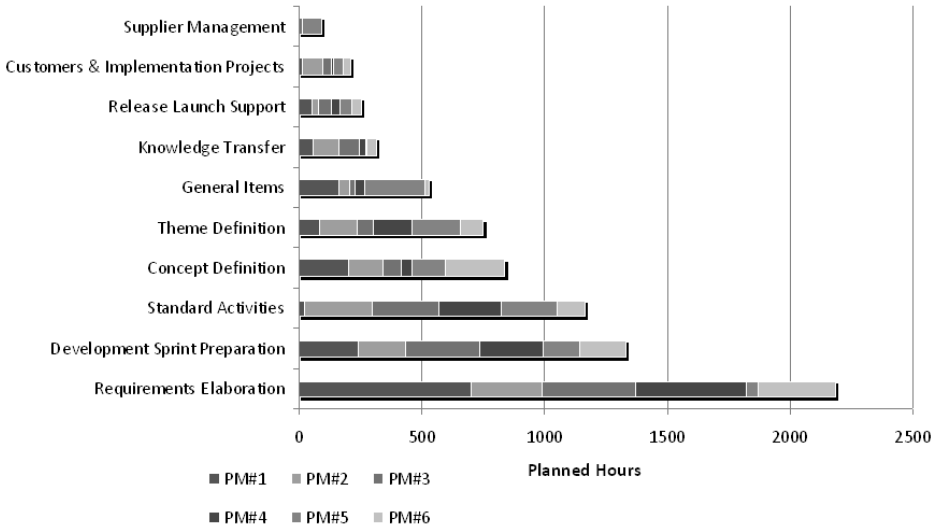


Figure 2.7.: Distribution of persons over the product management tasks

Based on the observations made above, we can identify three different roles. The first role is the *Senior Product Manager*, personified by PM#5. This role is mainly management-oriented, reflected by the low amount of time spent on requirements elaboration. Instead, a large share of the senior product manager's time is spent on high-level tasks, i. e. on the concept-or theme-level. The remaining time is for a fair amount spent on 'general tasks', 'supplier management' and other management related activities. The 'senior product manager' generally has the biggest influence on issues related to high-level decisions.

The second role that we identified was the general *Product Manager* role. In this particular team, this role was personified by four people with similar task-profiles, namely PM#2, PM#3, PM#4and PM#6. Each of these persons spent a similar, large amount of time on both requirements elaboration and development sprint elaboration. The 'product manager' role is responsible for the lion's share of low-level activities related to requirements and concepts. Furthermore, due to

the close relation between 'product managers' and the development teams, most of the activities related to the development sprint can be attributed to the 'product manager'.

The final identified role is that of the *Requirements Engineer*, which mainly focuses on low-level work. Almost one half of its time is spent on requirements elaboration. The other half of the time is chiefly divided between 'development sprint preparation', 'concept-level activities' and 'general activities'. The requirements engineer is not an active participant during the 'standard activities' such as sprint review and planning meetings. In this particular team, the role of 'requirements engineer' was personified by PM#1.

2.4.3 *Illustration: Maintenance Planning*

To illustrate the specific workings of themes, concepts and requirements within the PMSB, the maintenance planning theme is followed throughout its evolvement. The theme was introduced in 2008, when the case company chose to achieve a redefinition of its existing maintenance management solutions. The theme describes functionality related to the maintenance of facilities, and was initially introduced on the PB in month fourteen of the analysis. The entire SPM lifecycle of the theme lasts seven months.

Although the theme elaboration has not been explicitly documented in the PMSBs, due to the fact that a theme-section was not yet available, the theme is elaborated into several concepts. These concepts are 'planned maintenance (PM)', 'planned preventative maintenance (PPM)' and 'maintenance management (MM)'. Besides these, several other concepts exist that fall partially within this theme, such as 'work orders' and 'asset'. Each of these concepts is described in the 'vision, scope & requirements' document. Within the theme, a focal transition is visible from 'planned preventative maintenance' to 'planned maintenance'. Furthermore, 'maintenance management' is introduced in a later stage. For this example, the focus lies on the concepts of 'planned maintenance' and 'maintenance management'.

The 'planned maintenance' concept was introduced in month fourteen, right before the introduction of the new PMSB structure. The PB shows that the concept of 'planned maintenance' was elaborated into 152 requirements, subdivided over the groups 'no value', 'contract', 'generic' and 'maintenance planning'. Although a theme-section was not yet available in the PMSB of that month, it is clear from

the task-descriptions that they are related to theme-level requirements. During the next five months, the tasks related to the theme should shift from theme-level towards requirements-level. However, the PMSB shows that 'planned maintenance' tasks are placed under the requirements section right away. These tasks are concerned with the detailed elaboration of requirements, and would thus be expected later in the process.

This is slightly different for the initial tasks related to 'maintenance management' (i. e. the other concept within the 'maintenance planning' theme). The tasks related to the concept can be found on the PMSBs for the first time in month fifteen of our analysis, at the same time as the introduction of the new PMSB structure, and for the last time in month twenty. These tasks are, similar to 'planned maintenance' tasks, initially placed on the theme-level. As the concept matures, task-focus moves towards the concept-level and finally towards requirements elaboration, analogous to the 'theme/concept/requirements' lifecycle.

2.4.4 *Illustration: Planon Lite*

As shown in the previous section, the introduction of themes, concepts and requirements on the PB does not necessarily mean that all ideas brought up within the company follow the same, complete track through all phases. Although it is recommended to do so with large, complicated themes, the previous section has shown that it is possible and perhaps more efficient to take a 'shortcut'.

At the same time, introducing a more fine-grained notation also does not mean that every theme or concept will make it through all the steps of the requirements refinery. In fact, the added detail allows for an increased visibility of theme life cycles, which can result in the deletion of certain themes or concepts from the backlog. As an example of this, we describe the lifecycle of a new product idea, coined within the company in the fifth month of the analysis. This concept, called Planon Light, aimed at providing smaller companies with facility management services. The concept started out as an idea with a set of tasks related to the elaboration of the vision. After this vision was created, it was discussed and revised. It then had to be reviewed by the CIO. However, as the priority of this task was not high, it remained on the PMSB for several months. Only in month eleven is the task completed, after which the Planon Light concept disappears from the backlog, indicating a rejection of the concept.

This example shows two important points. Firstly, it shows that not all features start out at the theme-level. As indicated before, only complex features are considered themes, whereas smaller features can be directly translated to concepts or requirements. Secondly, the fact that tasks keep recurring on the backlog indicates that basic Scrum principles can be successfully translated to the product environment process, adding more clarity and structure.

2.5 LESSONS LEARNED

During its attempts to implement an agile SPM method, our case company has gained valuable experiences in this area. These experiences, which have mostly been mentioned in the previous sections, are listed in this section as a set of lessons that should be taken into account when implementing agile SPM alongside an agile software development method.

- *Alternate sprint cycles for SPM and Development* — One of the main lessons learned has been the importance of the alternating sprints. As discussed in Section 2.2.5, the software development and the SPM sprint are both performed continuously, but with a difference in starting date of approximately half of the sprint duration. This implies that each SPM sprint ends halfway the software development sprint, ensuring that the PB is ready to be used when the development teams start their new sprint.

- *Complex requirements are in need of structured detailing* — The essence lies in the division of requirements into themes, concepts and requirements. The structured agile requirements refinery approach has made it possible to effectively manage large sets of requirements of different granularity. Both high level and low level requirements are placed on the PB and handled in time by the appropriate person.

- *Daily Scrum meetings are essential* — The daily stand-ups, or Scrum meetings, that are essential within the Scrum development method, are also valued highly within the agile SPM method. The fifteen-minute meeting at the start of each day is experienced as a positive, helpful aspect of the process. By providing constructive critique, potential problems can be avoided and existing problems can be solved.

- *Backlog administration requires discipline* — We have observed that strict documentation of all tasks on the PMSB is still difficult to achieve. Although the PMSB can play a useful role in controlling the SPM process and keeping track of the progress of a sprint, the motivation to keep the current set of tasks and the amount of time spent on a specific task up-to-date is still lacking. However, it should be noted that one of the agile principles is a favoring of individuals and interactions over processes and tools. This means that, as long as the work gets done, project administration becomes less important.

- *Early collaboration promotes reuse and integration* — Since product managers in a Scrum team cooperatively work on a PMSB and discuss requirements before they have been implemented, re-use and integration opportunities can be spotted at an early stage. We suspect that higher quality software products are built using this approach than other approaches with less communication during the requirements specification process.

The final three lessons are similar to key aspects of the original Scrum development approach. Our research has shown that they also apply for agile SPM. The first two lessons apply specifically to agile SPM , and we consider them essential to a successful implementation of agile SPM.

2.6 CONCLUSIONS AND OUTLOOK

This paper demonstrates an attempt to apply agile principles to the SPM process, based on the proven structures of a well-adopted agile development method in an established software company. By providing the lessons that have been learned during this process, it is our hope that other companies can benefit from the experience of the case study company and that other researchers can apply and measure the effects of the requirements refinery.

In the description of the Scrum development method we have shown that an agile development process implies an environment that is dynamic and to which it is constantly adapting, be it in a controlled, effective way. It is not hard to imagine that such a dynamic development environment requires an SPM process that is adequately adapted to this. The effect of this is an increased demand for agile SPM processes, of which one has been described in this work. The main contribution of this work has been the description of an innovative SPM process based on agile

principles. The textual description along with process-deliverable diagrams both for the software development as well as the SPM processes allows effective reuse of the described method in other companies that find themselves in a comparable situation.

The experiences of the case study company have shown that, to ensure effective agile SPM , several factors should be taken into account, such as task size, backlog structure and willingness to keep the backlog up-to-date. By providing the specific lessons that Planon has learned during its experience with agile SPM and Scrum, we allow companies that wish to implement agile SPM to circumvent potential problems related to these items. However, as stated before, not a lot of research has been performed in the area of agile SPM. Future research should be aimed at further elaboration and formalization of the requirements of agile SPM processes. Part of this consists of further analysis of the tasks that are relevant to an agile SPM process. Besides this, more insight should be gained regarding the suitability of development methods for the application of agile SPM. More information should be gathered regarding current implementations of agile SPM processes, and their integration with agile development.

ACKNOWLEDGEMENTS

We would like to thank Planon for sharing its experiences, and for providing us with all required documents. Also, we would like to thank Jaap Kabbedijk for his input during the writing of this paper and Tjan-Hien Cheng for his contribution to the work.

3

GROWING INTO AGILITY: PROCESS IMPLEMENTATION PATHS FOR SCRUM

Many organizations struggle with the implementation of agile methods. Such methods pose considerable challenges related to organizational demand and process configuration. In this paper, we analyze the introduction of Scrum in the development organization in order to determine distinct approaches to its implementation. We compare the Scrum introduction paths of four case companies. This results in a discussion of implementation paths ranging from gradual to disruptive introduction of Scrum. The description of these paths provides insight into process improvements. We demonstrate how a structured description of process improvements can improve understanding of process improvement paths.

This work was originally published as:

Vlaanderen, K., van Stijn, P., and Brinkkemper, S. (2012). "Growing into Agility: Process Implementation Paths for Scrum." In: *Proceedings of the International Conference on Product-Focused Software Development and Process Improvement (PROFES)*. Ed. by Dieste, O., Jedlitschka, A., and Juristo, N. Madrid, Spain: Springer, pp. 116–130

3.1 INTRODUCTION

Since the publication of the Agile Manifesto in 2001 (Beck, Beedle, van Bennekum, et al., 2001), agile software development methods have become an ever-increasing part of the software development industry. The Agile Manifesto, but also the period before its publication, gave rise to several agile software development methods (Abrahamsson, Warsta, Siponen, et al., 2003). Examples of such methods are Dynamic Systems Development Method (DSDM) (Stapleton, 1999), Extreme Programming (Beck, 1999) and Scrum (Schwaber, 2004). The principles of such methods are that by employing them, the development process becomes more responsive to a changing environment, working software is chosen over extensive documentation, individuals and interactions are considered more important than tools and processes, and customer collaboration is valued more than contract negotiation (Beck, Beedle, van Bennekum, et al., 2001). In the last few years, these agile methods have proven to be successful in a large number of cases (Salo and Abrahamsson, 2008).

Scrum is one of the agile methods that is gaining popularity (Dybå and Dingsøyr, 2008). The Scrum development method was proposed by Schwaber (2004), at a time when it became clear to most professionals that the development of software was not something that could be planned, estimated and completed successfully using the common 'heavy' methods. Therefore, the Scrum method adheres to the principles of agile software development. Companies that have put Scrum to practice range from small companies as described by Dingsøyr et al. (Dingsøyr, Hanssen, Dybå, et al., 2006) to large multinationals (Fitzgerald, Hartnett, and Conboy, 2006). Research has shown that the use of Scrum within a company can lead to significant benefits (Fitzgerald, Hartnett, and Conboy, 2006).

In its purest form, Scrum provides a rather simple approach to agile, iterative software development. Although various additions have been made during the years, the process boils down to a planning phase, a closure phase, and several iterations, or sprints, during which the software is developed by one or more development teams. No new requirements can be introduced during these sprints. This ensures that the final product is being developed with a high probability of success, even within a constantly changing environment. This environment, which includes factors such as competition, time and financial pressure, maintains its influence on development until the closure phase. Sprints are structured according to a set of recurring activities. These basic, relatively fixed activities are the

sprint planning meeting, the daily Scrum meeting, the sprint review meeting, and the retrospective meeting. The essential deliverables within Scrum are the sprint backlog, the product backlog, and the increment of potentially shippable product functionality (Schwaber, 2004).

Although Scrum can provide significant benefits, implementing it in an organization is not a trivial task. Many authors have described the risks and critical success factors of Scrum implementations in various (types of) organizations (Scotland and Boutin, 2008; Smits and Pshigoda, 2007). Not only for bespoke software development but also for product software development organizations, as recognized by several authors, including one of the founders of the Agile Alliance (Cohn and Ford, 2003; Nerur, Mahapatra, and Mangalaraj, 2005). While agile methods can provide significant advantages to a software producing company, there are many challenges that can inhibit a successful move from traditional software development approaches to an agile environment, such as developer resistance, challenges in decision making, and the need for increased customer involvement (Boehm and Turner, 2005; Moe and Aurum, 2008). From a questionnaire on the subject conducted by Livermore (2008), it becomes clear that knowledge sharing is one of the most important factors in the success of implementation projects, followed by management support.

Process improvements can intrinsically be performed according to two main types. On the one hand, process improvements can be implemented in a revolutionary manner. In such a case, large changes are introduced to a process at once. On the other hand, process improvements can be cut into smaller chunks, causing a more step-wise evolution of the process. This distinction has been discussed quite frequently in literature (Krzanik and Jouni, 2002; Pettersson, Ivarsson, Gorschek, et al., 2004; Sawyer, Sommerville, and Viller, 1997; van de Weerd, Brinkkemper, and Versendaal, 2007) for various domains. The approaches are visualized in Figure 3.1.

Scrum comprises a fairly simple, iterative process model, which is to a large extent adaptable by the organization. The fact that it does not entail many constraints (Boehm and Turner, 2004) makes Scrum suitable to many types of organizations. However, this also implies that their specific approaches, and consequently the path to reach agility, can vary considerably. During recent year, a decent body of knowledge has been built regarding agile software development in the context of software process improvement (Borjesson and Mathiassen, 2004; Salo, 2006; Sidky, 2007). In this paper, we relate different styles of process improvement to the introduction of Scrum. We present a multiple case study, using the data from

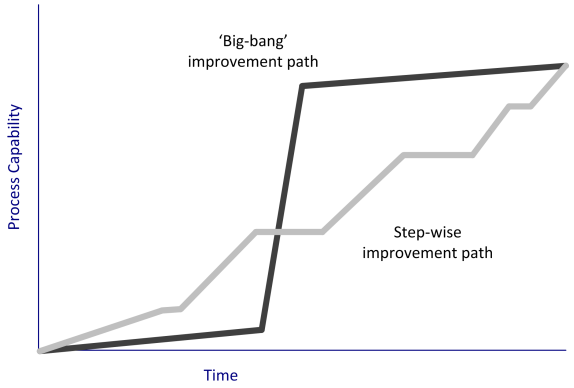


Figure 3.1.: Generic process improvement paths

the case studies to discuss the introduction of process improvements in general, and the introduction of Scrum specifically. As such, our analysis discusses the question *'Acccording to which paths can Scrum be implemented?'* In the context of this question, we define a path as a series of changes to a process.

A description of the research approach is provided in Section 3.2. In Section 3.3, we provide a description of the four case studies in which we outline the approach that was taken to implement Scrum in the organizations. Based on the case studies, we analyze the implications for the implementation paths in Section 3.4. We briefly reflect on the research project in Section 3.5. Conclusions and further research are described in Section 3.6.

3.2 CASE STUDY RESEARCH APPROACH

The research presented in this paper is part of a series of case studies that is performed with the aim of improving our knowledge of incremental method evolution. Out of the seven case study companies that are part of the overall research, four have implemented Scrum during the past few years. The implementation of Scrum is interesting from the perspective of software process improvement, as it is clearly defined in literature and it is a rather confined process. The case study companies selected for this research displayed different approaches to the implementation of Scrum, which makes the Scrum implementation approach an interesting unit of analysis. Each case company is briefly introduced throughout Section 3.3. For the sake of confidentiality, company names have been pseudonymised.

Data gathering was partly performed on-site, and partly through internet research, telephone calls, and e-mails. For each case, between three and five stakeholders were selected and interviewed. Roles that were suitable for interviews included (depending on the specific situation) the product manager, manager product management, general manager (in small companies), development manager, lead developer, senior consultant, and support managers. In addition, data was gathered through analysis of relevant documents. These documents included, if available and not limited to, requirements databases, internal communication regarding process changes, process descriptions, templates, change-plans, presentations regarding the Scrum process and other documents that describe changes in the process. All data collected either through interviews or through documents were included in a case study database.

3.3 CASE STUDIES

Each section below gives an account of the evolution of Scrum per organization. Each case study is described according to three stages; *preparation* describes the process up to the start of the Scrum implementation, *implementation* describes the actual process changes related to the generic Scrum activities, and *customization* describes additional changes to the process.

3.3.1 *ChatComp*

ChatComp is a privately-held company, which develops two product lines; instant messaging, and real-time messaging for smartphones, serving approximately 100 million users (within a growing market). ChatComp develops its products for multiple platforms, including the Android, Windows Mobile and BlackBerry platforms. As a consequence, it has to deal with constantly changing development platforms. Its products are released in cycles of 4 weeks. The organization deals with a large number of incoming wishes that vary significantly in nature, resulting in approximately 500 distinct product requirements per year. Within three years, the company has grown from 20 employees to over 100 employees. This was accompanied by several extensive changes to the development and product management processes.

Preparation - The introduction of Scrum in ChatComp started around August 2008, based on the initiative of the development manager. The aim of introducing

Scrum was to improve the company's ability to deal with an increasing amount of requirements and a quickly growing development team. In order to determine how Scrum was supposed to be implemented, an external consultant was hired. Once the idea was concrete enough, the company received an in-house training regarding the workings of Scrum.

Implementation - After the training, the entire development department started working according to the process of Scrum. During the first six months, the management team was strongly involved in the Scrum process. Once the teams became more adept at Scrum and the company started to grow, this involvement steadily decreased. The initial implementation was rather standard. This means that the sprint planning, daily standup, demo meeting and retrospective meeting all were in place, and the product managers have taken the role of product owners. The organization did make changes to the parameters of the process (such as sprint duration and team size) several times, causing a series of minor process changes.

Customization - Initially, the sprints lasted two or three weeks, depending on the team, causing a discrepancy between the sprint durations of some of the teams. At the same time, interdependencies started to become more of an issue as the amount of teams grew. This caused problems when, for instance, front-end teams needed functionality in the back-end that was not yet developed. These problems, which consisted mainly of delays, were solved by fixating sprint duration at two weeks for all teams.

The sprint planning meeting initially took approximately one day, because all the user stories had to be discussed, estimated and planned at once. This was considered too long to be effective. Therefore, story estimation is now done regularly throughout the sprints. This requires shorter bursts of attention, which eases the process. On the day of the sprint planning, the product owners present the user stories to their own team(s), followed by the selection and planning of user stories by the development teams. The process of selection and planning now takes approximately one hour, instead of one day.

Interesting in this light has been the introduction of the kick-off week. The kick-off week was an attempt to streamline the Scrum development process by dividing the year into four quarters of 13 weeks, each starting with a kick-off week during which all the user stories for the remainder of the quarter were identified, estimated and planned. Although the idea of a detailed roadmap per quarter seemed useful, the approach turned out to be problematic in practice. Developers were kept an entire week from developing their products, spending one entire week one estimating and planning is an exhausting activity, and the

workload turned out to be too high for the product owners. Quickly after the first quarter, ChatComp decided to cancel the kick-off week.

Due to the quick growth of the organization, internal communication of the product plan became increasingly important. For this reason, the product owners have changed the original schedule of the sprint planning meeting. The meeting now starts out with a presentation on the roadmap, the development focus, and the target market segment, in order to make sure that all internal stakeholders have the same objective and are aware of the context in which they are working. After this presentation, selection and planning continues as usual.

When Scrum was just implemented in ChatComp, IM was the only product line. Initially, there were three teams with a specific functional focus. After approximately eight months, one platform specific team was extended with a developer for another platform, making it a cross-platform team. The team quickly realized that this was not an ideal situation, as it inhibited collaboration. Once enough developers were available, the team was split into two teams, each focusing on their own platform. This process has recurred several times during the past few years. Each time, the organization moved back to platform specific teams to increase effectiveness of the teams. Currently, seven teams are active, each with a specific focus.

Other changes, less related to the interface between Scrum and product management, include the introduction of a Scrum of Scrums in order to facilitate inter-team communication, and a similar ScrumMaster counsil in which all ScrumMasters briefly meet to discuss any problems they encountered, things they learned or questions they have. The focus of this meeting is process-oriented instead of development-oriented.

3.3.2 *FacilityComp*

FacilityComp is an international software vendor that produces a total of five facility management and real estate management software products for medium to large organizations (Integrated Workplace Management Systems). Founded in 1984, it currently has a customer base of over 1300, supported by more than 325 employees. The company's products are marketed through multiple, international subsidiaries, and a worldwide network of partners. FacilityComp releases its products bi-yearly.

Preparation - Until 2004, product development at FacilityComp was based on the Prince2 method. New functionality for each product was fully described and planned upfront, after which it was developped in a waterfall approach. As the organization became larger and the products more complex, it became increasingly difficult to manage the development activities. Release cycles could take up to one or one and a half year and release dates were difficult to predict. Additionaly, many changes were requested during a project. The Prince2 method did not offer sufficient support for this, resulting in a lot of calculations that caused a large share of the product managements time to be put into these tasks instead of in product value. Based on these issues, the chief technology officer decided to change the development process to Scrum. He was also the person responsible for the actual introduction of Scrum. After he studied Scrum, he performed several internal sessions to explain the process. In addition, several persons took an external course to familiarize them with the method.

Implementation - Once a sufficient knowledge level was reached, the development team switched to Scrum completely. In principle, all basic Scrum activities were introduced at once. Initially, Scrum was only utilized for the new products, and the organization kept managing its cash-cow product using the old waterfall development method. One after the other, all products where then switched to the new Scrum process. Along the way, minor adjustments were often made. The teams were constantly searching for the right balance between sprint duration, size estimation, requirement size, and time spent on backlog administration. Changes to the process were initially often managed by the the CTO. However, they were increasingly delegated to the appropriate ScrumMasters.

Customization - Until 2007, agile product development was accompanied by non-agile product management. Although several stages of elaboration were employed, no fixed cycles were used. This caused that product managers did not manage to provide development with sufficiently detailed requirements before the start of each sprint. To improve this, the product management team adjusted their process to the Scrum principles as well. This implied a continuous adaptation of the product backlog to a changing environment. Since Scrum itself does not provide guidelines for effectively managing large amounts of requirements of different granularity, a set of stages has been introduced, called the *Agile Requirements Refinery* (see Chapter 2). Within the agile requirements refinery, an idea will generally move through the stages *vision*, *theme*, *concept*, and *requirement definition*. During these stages, requirements are refined with details and specifications. This has resulted in a new categorization on the product backlog.

3.3.3 *SocialComp*

SocialComp is a large social networking site in the Netherlands, focusing mainly on Dutch visitors and members. It was founded in 2004, and since then it has built a user base of over eleven million accounts and serves over 5,8 billion page views per month. Even though its active user base is declining in the past few years, it potential market, consisting of individual users, is growing. SocialComp releases updates to its platform once per week. The organization has a very low level of organizational policy and does not need to deal with a high number of standards or a high level of legislation. However, it does receive many product requests directly from its users.

Preparation - SocialComp is a very young company, despite the fact that it was founded in 2004. Many of the employees are below the age of 35, and this is reflected in a very creative and informal setting. This setting has always resulted in a development environment without strict processes and, consequently, without a strong structure. This attitude is seen throughout the organization. However, the considerable growth during the past few years has forced the organization to apply some structure to its internal processes. The move to Scrum should therefore be seen in the context of a larger movement within the organization.

During the period between 2009 and late 2010, the organization moved away from its 'startup structure' and added a clearer division of responsibility. During this time, Scrum is mentioned several times, and requirements are from that time on written in the form of user stores. However, new functionality was still developed in the form of waterfall projects, where all functionality was designed upfront. Requirements prioritization and selection was still done based on 'fingerSoftware Process Improvement (SPI)tzen-gefühl'. Real improvements were made in the end of 2010, when a clear product vision and strategy were defined. The previous 15 focus areas were reduced to 6, with one responsible product manager per area. The choice to implement Scrum was made by the CTO at that time, based on earlier, positive experiences with it. He felt that it offered advantages to SocialComp as well in the form of improved backlog management and scope change management. Scrum's team-based approach seemed like it would respect the independence of the programmers while providing a scalable structure.

Implementation - The introduction of Scrum was performed gradually. The first step towards agile development was the creation of fixed teams that were located in one room. Each team was assigned a team lead and a ScrumMaster. Through several internal presentations, all stakeholders were taught the workings

of Scrum. During the first period, the teams got used to the process by working with a product backlog in fixed teams.

After some time, daily standups were introduced. These standups were performed in the original sense as described by Schwaber (2004). Each day, teams would come together to discuss work performed the day before, work planned for the current day, and any problems that were experienced. The introduction of daily standups was gradually followed by other essential elements of Scrum. Rather quickly, a definition of done was introduced. This definition has changed somewhat throughout the last two years, and basically states that functionality is done when it is developed, verified and prepared for automated testing. Soon after the introduction of the definition of done, the first demo meetings were organized. Such meetings were initially only held when important features were finished. Later, demo's became a regular part of the Scrum process. Retrospective meetings were introduced gradually. Initially, only a few developers would discuss a sprint after it had finished. After some time, other teams started to hold retrospective meetings as well.

Customization - The gradual introduction of Scrum seems to correlate with the acceptance of Scrum within the organization. Initially, many developers were rather skeptical due to the changed responsibilities, more formal processes and reduction in freedom. During the past two years, approximately 20% has come to actively support Scrum, and 60% does not have a strong opinion about it. The final 20%, mainly developers that have been at the company for a long time, actively resists a full Scrum implementation. Due to the informal setting within SocialComp, this resistance is not seen as a huge problem. If a developer fails to comply to the process but maintains a good rate of productivity, this is accepted as well.

3.3.4 *TimeComp*

TimeComp is a small independent Dutch software company, founded in 1992. The organization provides qualitative software applications and accompanying services to fulfil the need of achieving a higher efficiency from the utilization of human resources. TimeComp currently has two software products in its portfolio: one for time resource management, and one for printing and copying facilities for organizations. Its market consists of small to medium enterprises, and its size is fairly stable. It serves approximately 400 organizations within Europe. It releases

its products twice per year, based on a steady inflow of customer requests. As TimeComp is a fairly small organization, with approximately 25 persons, it has a fairly loose internal structure with low internal policy.

Preparation - Until 2010, TimeComp worked in a non-agile and non-iterative manner. The product planning and development processes were not officially structured. Development was performed ad hoc and little overview existed on the features that were currently requested, worked on, or implemented in the latest release. The development process was not formally structured. Functionality was built as it was made up, and new releases were delivered when enough improvements had been implemented.

Due to a sudden demand increase and rise in customer wishes, the old development approach of the company was not suitable any more. The management team and employees started to realize that a more reliable requirements management process and a more predictable release heartbeat were needed in order to stay competitive. This awareness was strengthened by the recognition of increased communication problems in the development department. There was little teamwork among the developers, and a great deal of expertise resided with specific employees.

The first suggestion to use Scrum in the company was opted by the former head of product development (now product manager). He explicitly pointed out these issues to the internal stakeholders, and introduced Scrum as a possible solution. The introduction of Scrum was accompanied by a change of management, change of organizational structure and an improved requirements management approach. It took more than a year before Scrum was actually picked up by some of the developers.

Implementation - Scrum was formally introduced in 2010. This introduction was performed within one single meeting with the management team and development employees together. The meeting was held in a Scrum-like fashion, i. e. everybody was standing and the presentation progress was captured with a burndown chart. During this meeting, the Scrum method was explained briefly to the audience again. Before this meeting, the new product manager together with the managing director, created a first product backlog from a large list of customer wishes. During the meeting, user stories that would be implemented in the next sprint were chosen, story points were assigned and the tasks were divided. These actions initially took a lot of time and discussion. This especially applied to the estimation of story points; the developers were not used to predicting what time they would need to spend on a feature implementation.

All meetings prescribed by the Scrum methodology were included and planned for this new agile development method, and the first three-week sprint was started. The scrum team consisted only of the five developers, the role of Scrum Master was occupied by the development manager and the role of the Product Owner was occupied by the product manager. The Scrum process elements were followed as strictly as prescribed. To facilitate the Scrum process and to store user stories, a Scrum specific tool was employed.

Customization - During the initial period, the developers showed a lot of resistance towards Scrum. One effort to undermine the Scrum process was to cancel the daily Scrum meeting. However, once the product manager discovered this, it was quickly reinstated. Another example is the introduction of a refactoring sprint, which was mainly used as 'slack time' to perform miscellaneous tasks and bug fixes, instead of actual refactoring. This sprint type was also quickly cancelled. Instead, developers now have a 'delay-day' before the start of a new sprint, during which outstanding issues can be resolved that would otherwise disrupt the next sprint.

An important addition to the process has been the introduction of grooming sessions. At first, these grooming sessions strongly resembled the usual sprint meetings. The meetings were still very long, and many developers still did not understand the user stories after the explanations. After a while, these sessions were timeboxed to one hour and much more visualizations (e. g. adapted screenshots) were used to clarify user stories. Very complex user stories (which could not be cleared with one short meeting) were first attended in a work group.

3.4 SCRUM IMPLEMENTATIONS PATHS

The case companies show a high degree of variation in their specific implementation of Scrum, as well as the path that they have followed in order to reach that situation. The cases have shown that a Scrum implementation path is highly dependent on the context of the organization, including its internal culture and the implementation strategy, i. e. top-down or bottom-up. Although the case study research presented in this paper is not suitable for quantitative analysis regarding the relation between situational factors and implementation paths, we can make several important observations.

First of all, we observe that only one case company implemented Scrum as an explicit part of a larger process improvement strategy. In most cases, the drivers

Table 3.1.: Characteristics of the Scrum implementation

OVERARCHING PROCESS IMPROVEMENT FRAMEWORK	
ChatComp	None
FacilityComp	None
SocialComp	None, but part of a larger professionalization effort
TimeComp	None
MAIN DRIVERS FOR THE IMPLEMENTATION OF SCRUM	
ChatComp	Increasing amount of requirements; Quickly growing development team
FacilityComp	Increasingly complex development setting
SocialComp	Unprofessional and unstructured product development process
TimeComp	Low development productivity; Unclear product definition process
INITIATOR FOR IMPLEMENTING SCRUM	
ChatComp	Chief Technology Officer
FacilityComp	Chief Technology Officer
SocialComp	Development Manager
TimeComp	Development Manager / Product Manager
EXTERNAL ADVICE	
ChatComp	Initial Scrum training
FacilityComp	Initial Scrum training
SocialComp	External advice after implementation
TimeComp	None

were similar to the advocated and well-known advantages of agile software development methods, such as quicker time-to-market, less process overhead, and scalability. However, opposite to what is often stated in literature, the implementation of Scrum was in all cases initiated by the management team instead of the developers. In one of these cases, at TimeComp, Scrum was actually forced onto the developers.

All managers responsible for the introduction of Scrum have indicated that the initiation period took a significant amount of time. A large share of this time was spent on familiarization with the method, either through available literature or with the help of an external advisor. This process was never steered using existing process improvement frameworks or knowledge databases, leaving room for improvement in the areas of process selection and knowledge acquisition.

Table 3.2 provides an overview of the paths that the case companies followed during the implementation of Scrum. The first column shows all basic elements of Scrum as described by Schwaber (2004), in addition to the additional activities found throughout the cases. For each activity, the table indicates the implementation order. The arrows indicate where multiple elements were introduced at the same moment. Such collections of changes are named increments, which are defined as the collection of changes to a method between two points in time (van de Weerd, Brinkkemper, and Versendaal, 2007). The bottom row summarizes the style of the process improvement effort.

We can see a clear distinction in the approaches that were taken to implement Scrum. In three of the case, the implementation path mostly resembles that of disruptive or revolutionary process improvement. In these cases, an initial training period was followed by a sudden switch to Scrum. At this time, a large, high-impact change to the development process is made, changing the way of working radically. Such a process improvement path has a high impact on the organization, and can cause considerable resistance among the employees, such as in the case of TimeComp.

In the case of SocialComp, we see a different approach. During the introduction of Scrum at this organization, a range of small process changes is made with some time in between, applying the Scrum constructs in an iterative manner. SocialComp has applied a more step-wise approach to move towards the desired process state. New activities and deliverables were introduced in small bundles, and refinements were constantly performed.

In all cases, the process is constantly being adjusted based on internal feedback. This constant change is an inherent aspect of Scrum, which advocates continuous improvement. Interesting in this regard is that such changes are not always an improvement. In the case of TimeComp, the removal of the daily Scrum meetings from the process was quickly reversed, once it became clear that this would jeopardize the quality and stability of the process.

Once a certain part of Scrum has been implemented, all organizations demonstrate a trial-and-error approach to reach the desired process state. These trials can consist of additional Scrum activities (shown in the bottom section of Table 3.2), or configurations of the already implemented process (e. g. changes in the team size or sprint duration). With disruptive improvement paths, the introduction of Scrum and the fine-tuning are two separate processes. With gradual implementation paths, a continuous improvement can be observed.

Table 3.2.: Increment sequence of Scrum elements

	SCRUM ELEMENTS	CHATCOMP	FACILITYCOMP	SOCIALCOMP	TIMECOMP
Generic Scrum Elements	*Product Backlog* *Fixed Teams* *Sprint Backlog* *Product Increments* *Sprints* *Sprint Planning Meeting* *Daily Scrum Meeting* *Sprint Review Meeting* *Definition of Done* *Retrospective Meeting*	Increment1	Increment1	Increment1 Increment2 Increment3 Increment4 Increment5	Increment1
Additional Scrum Elements	*Agile SPM*	-	Increment2	-	-
	Requirements Refinery	-	Increment3	-	-
	Scrum of Scrums	Increment2	Increment4	-	-
	ScrumMaster Counsel	Increment3	-	-	-
	Kick-Off Week	Increment4	-	-	-
Improvement style		Disruptive & Incremental	Disruptive & Incremental	Incremental	Disruptive

The four case indicate that the style of the process improvement path is highly dependent on the context of the organization, including amongst others its drivers, size, culture, and products. One approach to create a better insight into the drivers behind a specific process improvement effort and the possible issues related to certain process improvements, and to provide a direct link between specific goals, detailed process attributes, pre-conditions and post-conditions, is by describing process improvements in a structured manner. Ultimately, it facilitates knowledge sharing by standardizing the gathering of relevant process improvement information.

In order to provide the insight described above, we have partially described the implementation of Scrum at SocialComp using a structure called the *method increment case description* in Table 3.3. This structure is further elaborated in Chapter 4, with the aim of providing a concise description of improvement paths that allow organizations to reflect on their implementation and to guide similar improvement efforts.

Sharing detailed experience related to process improvement in a structured manner such as demonstrated above can aid in the prevention of repeating mistakes in similar contexts. Method increment case descriptions can be used to either describe improvement paths at organizations that have already implemented a certain process, or to prescribe a process improvement path for an organization, taking into account its specific context and goals.

3.5 DISCUSSION

In order to ensure the quality of our work, we have followed the guidelines that have been defined for performing multi-case study research (Jansen and Brinkkemper, 2007; Yin, 2003). Interviews were held with several people in order to cross-check documentation found and to confirm facts stated in other interviews. Concerning the ability to generalize the results, we cannot make quantitative statements about the general population based on the findings in this paper. Generalizability is partially ensured by the analysis of four separate cases. However, we expect that the qualitative results are applicable in other, similar situations.

A threat to the validity of the research is that not all case studies were performed by the same persons. However, the first author of this paper was present at each interview. In addition, a case study protocol has been written prior to the start of the case studies. This protocol has been used during each case. The findings of the case studies have been stored in a case study database (Jansen and Brinkkemper, 2007).

3.6 CONCLUSIONS AND FUTURE RESEARCH

The multiple case study research presented in this paper shows distinct approaches for the implementation of Scrum. Three of the organization initially followed a

Table 3.3.: Structured process improvement description for SocialComp

NAME		IMPLEMENT SCRUM
Goal in context	#	Improve the effectiveness of the development team
Scope		Development process
Primary and Secondary Stakeholders	#	Head of product
	#	Developers
	#	Product manager
Trigger		The development organization resembles that of a 'startup' organization. Due to growth, there is a need to apply some structure to the internal processes.
Pre-Conditions	#	Unstructured software development process
Post-Conditions	#	Full initial implementation of Scrum
Increment Path	1.	Assign Scrum roles
		- Driver: Get people involved
		- Stakeholders: Head Of Product, product managers
	2.	Implement product backlog
		- Driver: Clarify the work to be done, improve uniformity of requirements, provide a way of work planning, improve communication
		- Stakeholders: Head of Product, product managers
	Further details omitted due to space limitations	
Unordered Increments	#	Increments not part of the sequence or executed during multiple steps.
		None
Failed Paths	#	Steps that were undone afterwards.
		None

disruptive path which, in one case, led to significant resistance from developers. In two of these cases, the basic Scrum process was extended with additional activities in an incremental manner, such as a Scrum of Scrums and a Kick-Off Week. The fourth organization followed a gradual implementation path for the entire introduction of Scrum.

On the scale of revolutionary versus evolutionary process improvement, the position of the cases varies considerably. In neither of the cases, we can speak of a completely 'big-bang' approach to the implementation of Scrum. In all cases, the process is implemented or altered in an iterative manner to some extent. In the case of TimeComp, the implementation can be classified as fairly disruptive. However,

given the state of the development team at the time of implementation, this could not be avoided. Once the team had accepted Scrum as a new development process, it started improving it iteratively.

Situational factors have highly influenced the parameters of Scrum in all organizations. Based on organizational properties such as company policy and business unit size, the organizations have implemented different sprint lengths and Scrum activities, and have employed different implementation styles. However, the manner in which these choices were made was in most cases rather unstructured. The link between context and decision is rather unclear, and trial-and-error approaches often resulted in unwanted results and lost resources. Based on this observation, we think that further research into the relationship between situational factors and the 'parametrization' of methods would be very valuable.

In order to provide more insight into the change process, we have demonstrated how a structured approach can be used. Such structured process improvement descriptions aid in sharing knowledge related to the individual steps within a process improvement effort, using attributes such as stakeholders, drivers, and improvement steps. However, further research into evolutionary process improvement and more insight into the structure of improvement steps is required. This is in line with the statement that "incremental policies are anxiously required by the industry" (Krzanik and Jouni, 2002, p. 1). Essential in this light is a better linking between situational factors, available knowledge, and process needs. A knowledge infrastructure for incremental process improvement is currently being developed (see Chapter 6–Chapter 8).

Part III

METHOD INCREMENTS

4

DOCUMENTING EVOLUTIONARY PROCESS IMPROVEMENTS WITH METHOD INCREMENT CASE DESCRIPTIONS

Evolutionary process improvement is a common approach to manage the complexity and risk of large software process improvement efforts. Performing Software Process Improvement (SPI) through a sequence of small steps allows organizations to reflect and steer the effort often and avoid failed improvements. However, few methods currently exist to structure improvement paths in a clear and concise manner. In this paper, we present a template for such a structuring method, based on Use Case Descriptions and Method Engineering (ME) techniques. A concise description of improvement paths allow organizations to reflect on their implementation and to guide similar improvement efforts. A case study of two large improvements within a small Dutch software company is used for evaluation.

This work was originally published as:

van Stijn, P., Vlaanderen, K., Brinkkemper, S., et al. (2012). "Documenting Evolutionary Process Improvements with Method Increment Case Descriptions." In: *Proceedings of the European Conference on System & Software Process Improvement and Innovation (EuroSPI)*. Ed. by Winkler, D., O'Connor, R. V., and Messnarz, R. Vienna, Austria: Springer, pp. 193–204

4.1 INTRODUCTION

The quality of Software Product Management (SPM) processes strongly influences the productivity of the organization and the quality of its products (Ebert, 2007). Process improvements tend to be complex and are often implemented evolutionary (van de Weerd, Brinkkemper, and Versendaal, 2010), consistent with the definition of Software Process Improvement (SPI), which is formulated as "a continuous and *evolutionary* approach to improve a software organization's capability to develop quality software in response to customer requirements" (Iversen, Mathiassen, and Nielsen, 2004, p. 396). This approach emphasizes *stepwise improvement* of the software processes, systematic assessment of an organization's current operations, and application of normative models for organizing a software operation (Iversen, Mathiassen, and Nielsen, 2004).

Several methods exist to model processes within companies. A well know method in the Method Engineering (ME) domain is MAP (Rolland, Prakash, and Benjamen, 1999). In the Business Process Management (BPM) domain, Business Process Modeling Language (BPML) is a common approach (Thiagarajan, Srivastava, Pujari, et al., 2002). Another approach from the ME domain is the Process-Deliverable Diagram (PDD), proposed by van de Weerd and Brinkkemper (2008). A variation of this method has been used to model the individual increments (van de Weerd, Brinkkemper, and Versendaal, 2010). However, no convenient methods have been found in literature to describe a sequence of increments in a clear and concise way. Such a method would be useful for reporting and reviewing process changes. Furthermore, these overviews can prove valuable for companies that are considering a similar process improvement. By building a knowledge base of process improvement paths, those companies can use the reference improvement paths as guideline to implement their own process improvements. Therefore, the research question this article aims to answer is *How can multiple increments that are part of a larger evolutionary process improvement be documented conveniently?* In order to answer this question, we apply the use case description technique (Cockburn, 1997), as it fits with the high-level attributes that are important for describing process improvements.

This paper is organized as follows: first, an overview of the most relevant literature on software process assessment and improvement is provided. Then, the research method is described in more detail. After this, Section 4.3 describes our approach for describing process improvement steps, after which the approach

is evaluated with data obtained through a case study which is described in Section 4.4. The paper concludes with a conclusion and discussion of further research.

4.2 RESEARCH CONTEXT

4.2.1 *Related Literature*

SPI has been researched for over twenty years, after the publication of the frequently cited work of Humphrey (1989). Process improvements generally improve the quality of the software product directly, although the impact per quality factor can differ depending on which methodology is used (Ashrafi, 2003). Many specific methods, usually in the form of maturity models, have been developed for these software process improvements. The most well-known of these are Bootstrap (Kuvaja, Simila, Krzanik, et al., 1994), the Capability Maturity Model (CMM) (Paulk, Weber, Curtis, et al., 1995), Software Process Improvement and Capability dEtermination (SPICE) (El Emam, 1997) and the Quality Improvement Paradigm (QIP) (McGarry, Pajerski, Page, et al., 1994). A widespread idea in SPI approaches is the use of an evolutionary approach. Changes are not implemented through one single (big-bang) transformation but by a sequence of changes (steps) over a period of time (Aaen, Arent, Mathiassen, et al., 2001). An example of a method based on such an evolutionary approach is business process re-engineering (Davenport, 1993).

Measuring the current state of any (software) process is the first and a very important step for any process improvement (Paulk, Weber, Curtis, et al., 1995; Weinberg, 1997). Many different measurement methods exist, such as the Goal/Question/Metric (GQM) measurement method, statistical process control, practical software measurement and the Balanced Score Card (BSC) (Komi-Sirvio, 2004). In an earlier study, van de Weerd, Brinkkemper, and Versendaal (2010) have investigated how large improvements can be separated into small increments. When doing so, process assessment can be performed between the different steps. Although they proposed a way to visualize a single method increment, no solution is provided to visualize multiple increments (steps) within one improvement. Furthermore, the drivers of the increment are not adopted in the visualization. No other publications have been found covering these aspects.

4.2.2 *Research Approach*

The aim of this research is to fill the gap identified above by providing an approach for documenting sequences of steps that form a larger software process improvement effort. This approach should provide all necessary information on individual steps needed for their reproduction, while maintaining a high level overview to quickly assess the whole improvement. Furthermore, it should allow for an easy comparison of two comparable process improvements within different companies.

In terms of the design science research framework by March and Smith (1995), we *build* and *evaluate* a *model* that allows for the description of process improvements. As a basis for the design of the new documentation approach, we use the use case description template by Cockburn (1997). The result is a template in which process improvements and their steps can be described.

The proposal is evaluated in the sense of functional testing as defined by Hevner, March, Park, et al. (2004), i. e. to discover failures and shortcomings. For this purpose, data from an earlier performed retrospective case study have been filled in the template. The case study was performed as part of a multi-case study to unravel large process improvements into smaller steps. During a set of interviews, in combination with a document analysis, data was gathered regarding several process improvements performed within a small Dutch software company during the past two years. These process improvements were then unraveled in sequences of small steps (increments), which resulted in useful information to test the template on.

During the case described in this paper, process improvements in the domain of SPM were investigated. SPM is defined as the process of managing requirements, defining releases, and defining products in a context where many internal and external stakeholders are involved (van de Weerd, Brinkkemper, Nieuwenhuis, et al., 2006b). The involved key process areas, together with the (internal and external) stakeholders and their relations are summarized in the SPM competence model (van de Weerd, Brinkkemper, Souer, et al., 2006). As the quality of the SPM processes enhances the quality of the software product (Ebert, 2007), improving these (software) processes is often required.

For measuring the maturity of the organization's SPM process, we employed the Situational Assessment Method (SAM) (Bekkers, Spruit, van de Weerd, et al., 2010). The SAM combines multiple descriptions of an organization and its product management process by performing an analysis of the organization's Situational Factors (SFs) (Bekkers, van de Weerd, Brinkkemper, et al., 2008) and

its product management capabilities. The results of the approach are described as solution oriented and realistic, allowing for incremental growth and requiring little effort to obtain. This assessment has resulted in a filled in SPM maturity matrix (Bekkers, van de Weerd, Spruit, et al., 2010). The SPM key processes are represented by the rows and are divided into four groups (the business functions). The columns 0 to 10 represent the maturity levels (where zero is low and ten is high). The letters A to F represent the capabilities. Each focus area has its own unique capabilities; the amount of capabilities within a focus area varies from two (A- B) to six (A-F). The maturity matrix suggests the best implementation order for the capabilities (from left to right). The placement of the capabilities is based on a series of interviews with experts from both the scientific world and the field of practice, and questionnaires among product managers (Bekkers, Spruit, van de Weerd, et al., 2010). The SPM maturity matrix (Bekkers, van de Weerd, Spruit, et al., 2010) is very useful for our purpose as it allows us to quantify the process changes in the method increment case description.

4.3 METHOD INCREMENT CASE

4.3.1 *Method Increment Case Description*

A use case specifies the behavior of a system or part of a system, and is a description of a set of sequences of actions, including variants, that a system performs to yield an observable result of value to an actor (Booch, Rumbaugh, and Jacobson, 1999). Use cases capture the intended behavior of the developed system, with no specific emphasis on how this behavior is implemented. In many senses, this is very comparable to a process improvement when seen as a sequence of actions that is performed (Iversen, Mathiassen, and Nielsen, 2004) and that results in an observable result of value to stakeholders: an improved capability to develop quality software (McFeeley, 1996).

These similarities have been the reason to employ Use Cases as a basis for the semantic framework to document process improvements. Cockburn (1997) has defined a template for the description of use cases, providing a textual overview that supports the visualization. This description consist of the title, goal, scope, pre-conditions, success end conditions, failed end conditions, actors and triggers. Furthermore, a part for the descriptions of the different steps in the case exists, and for the extensions and variations.

Many of the use case description elements can be directly applied to the domain of SPI. This has resulted in the template shown in Table 4.1. To emphasize the relation of this approach with the Unified Modeling Language (UML)'s use case descriptions, it is named *method increment case description*. Most elements from the original use case description template were directly applicable to the method increment case description. However, some adjustments were made.

Table 4.1.: Template of a method increment case description

NAME		GOAL AS A SHORT ACTIVE VERB PHRASE
Goal in context	#	Extended description of the goal
Scope		Identifier of the process under consideration
Primary and Secondary Stakeholders	#	Stakeholders [1..n]
Trigger		Brief description of the original problem
Pre-Conditions	#	Conditions [1..n] applicable before the process improvement
Post-Conditions	#	Conditions [1..n] applicable after the process improvement
Increment Path	#	Path steps [1..n] - *Driver, Stakeholders, Affected Capabilities*
Unordered Increments	#	Increments not part of the sequence or executed during multiple steps. - *Driver, Stakeholders, Affected Capabilities*
Failed Paths	#	Steps that were undone afterwards. - *Driver, Stakeholders* - *Failure Reason*: Why was this failed and what were the consequences?
Reference to PDD		PDD Increment relevant to this method increment case description.

The *Scope* element is used to denote the sub-domain to which the process improvement applies. This can be described in terms of an existing process framework, such as *product planning* in the case of SPM , or as a brief description. In use case descriptions, the *Level* is used to describe whether it is related to a summary, a primary task, or a sub-function. As we don't have a similar way of categorizing our process improvements, we have left it out of the template.

The *Trigger* in the context of use case descriptions is the action upon the system that starts the use case. Similar to this, we interpret the trigger as a description of the problem or dissatisfaction that initiates the process improvement effort. The

elements *Preconditions* can serve as a more detailed elaboration of the driver. If possible, these preconditions should be described in a (semi-)formal manner, such as with the capabilities of the SPM Maturity Matrix (Bekkers, van de Weerd, Spruit, et al., 2010) in the case of the SPM domain. When no formal method is available, a short textual description can be provided.

To allow for a description of the change of state after the successful implementation of a process improvement, we have renamed the element *Success End Condition* to *Post-Conditions*. The changes in process maturity should also be described in a (semi-)formal manner, e. g. a low maturity in the area of portfolio management or low employee satisfaction amongst developers. When no formal method is available, a short textual description can be provided.

For describing the steps within a software process improvement, we use three elements. Successful improvements paths are described in the element *Increment Path*, renamed from *Description*. Paths that failed, i. e. did not result in the desired end state, can separately be described in the element *Failed Paths*, renamed from *Failed End Condition*. As some steps cannot chronologically be placed inside the main *increment path*, we have added *Unordered Increments*.

All steps of the improvement path elements described above should be recorded in a clear and concise manner in order to allow a good overview. Besides a one-line description of the step, the following three attributes should be described.

DRIVER: The driver(s) or rationale of the increment need to be described briefly to allow for later analysis of the improvement. Furthermore, this allows other companies that consider the same improvement to determine whether or not they also need the specific increment.

STAKEHOLDERS: As with use cases, the stakeholders involved in the specific step should be described. This can aid when determining shifts in roles.

AFFECTED CAPABILITIES: For every step in the improvement path, the affected capabilities are quantified to show the exact consequences of an individual improvement step on the processes.

As with use case descriptions, the element of time is not included in the variant method for SPI descriptions. The rationale behind this is that the time needed for the process improvement depends largely on the available (human) resources, the company culture (e. g. resistance to change) and other situational factors. Furthermore, the exact time and dates of a increment are often difficult to discover after a

large process improvement. Additionally, even if the start and end date of a large process improvement can be discovered, this will still prove to be very hard (if not impossible) for individual increments. By not including the element of time, these issues are acknowledged and it is emphasized that the different steps are more relevant than the exact timings of these steps.

Finally, we recognize that there are often multiple approaches to implement a certain process improvement, often depending on the situational factors of the company. The original use case description template provides for this in the form of *Sub-variations*. We have removed this element from the method increment case definition template as the description's purpose is to capture a single process improvement effort. When bundled, method increment case descriptions can provide insight into alternative paths for similar process improvements.

The final attribute of the original use case description template is *Extensions*. This attribute's purpose is to describe each step that is altered, the condition under which this happens and the actions or sub-use cases that are performed extra. However, we have removed extensions in the method increment case description, due to the reason that extensions can also be seen as individual improvement steps that precede or succeed the step they are supposed to extend. As the goal is to find individual improvement steps to a level of granularity as low as possible, these extensions are considered as individual steps themselves. Thus, these extensions or extended paths fall under the attribute *Increment Path*.

4.3.2 *Linking Method Increment Case Descriptions to PDDs*

A field related strongly to SPI is that of method engineering, which is defined "the engineering discipline to design, construct and adapt methods, techniques and tools, for the development of information systems" (Brinkkemper, 1996, p. 276). These methods are based on a specific way of thinking and consist of "directions and rules, structured in a systematic way in development activities with corresponding development products" (Brinkkemper, 1996, pp. 275–276). Method engineering that takes the situational factors of the concerned company into account is referred to as situational ME (Harmsen, Brinkkemper, and Oei, 1994). During the last several years, several modularization constructs have been proposed for situational ME (Ågerfalk, Brinkkemper, Gonzalez-Perez, et al., 2007; Cossentino, Gaglio, Henderson-Sellers, et al., 2006; Deneckère, Iacovelli, Kornyshova, et al., 2008).

The core concept within our research is the method fragment. Method Fragments (MFs) (Harmsen, Brinkkemper, and Oei, 1994) are defined as "... a description of an Information Systems (IS) engineering method, or any coherent part thereof". MFs consist of a process part and a product part, with a link between these two parts. The general approach to visualize a method is by means of a PDD, which is a meta-modeling technique that is based on a combination of a UML activity diagram and a UML class diagram (van de Weerd and Brinkkemper, 2008).

In an earlier study, van de Weerd, Brinkkemper, and Versendaal (2007) applied the PDD modelling technique to describe process improvements, resulting in the definition of PDD increments. An example of such a PDD increment is shown in Figure 4.1.

We have included a reference to a PDD increment in the method increment case description template, in order to facilitate the extension of the case description with a visual diagram. We believe that this will increase the clarity of the process improvement description, and that it will provide more insight into the impact of the described changes.

4.4 EVALUATION

The proposed template has been evaluated with the results from a case study at a small Dutch software company with 25 employees, providing a software application and accompanying services to fulfill the need of achieving a higher efficiency from the utilization of human resources. The product has around 400 customers, mostly local authorities, insurance companies and banks, with a total of 300.000 users utilizing the product. In the period between 2009 and 2011, the case company implemented several process improvements in the areas of requirements management, release planning, and development processes.

At the start of the process improvement efforts, the organization was developing in an unstructured, waterfall-like fashion. Roles were not clear and customers were not involved in the process. Furthermore, the documentation of the requirements was problematic, due to which the requirements could not be traced back to the original feature request and corresponding customer who placed it. The fact that requirements were often not (well) documented ultimately resulted in requirements getting lost.

In order to measure the maturity of the organization's SPM process at that moment (June 2009), we employed the Situational Assessment Method (Bekkers, Spruit, van de Weerd, et al., 2010), described in Section 4.2.2. The situation is visualized in the form of an SPM Maturity Matrix in Table 4.2 with the light gray boxes indicating the original maturity.

FOCUS AREA		MATURITY LEVELS										
Title	*Code*	0	1	2	3	4	5	6	7	8	9	10
Requirements Management												
Requirements Gathering	RG		A		B	C		D	E	F		
Requirements Identification	RI			A			B		C			D
Requirements Organizing	RO				A		B		C			
Release Planning												
Requirements Prioritization	RP			A		B	C	D			E	
Release Definition	RD			A	B	C				D		E
Release Definition Validation	RDV					A			B		C	
Scope Change Management	SCM				A		B		C		D	
Build Validation	BV				A			B		C		
Launch Preparation	LP		A		B		C	D		E		F

Table 4.2.: Partial SPM maturity matrix for the case company

After two years, the maturity of the requirements management process has increased significantly, due to an improved requirements management workflow, improved tooling and better improved customer involvement. Currently, all requirements are centrally stored and rewritten to product requirements, while a connection with the original market requirement (and information such as related customer) remains. The dark gray boxes in Table 4.2 indicate the added capabilities after the process improvements.

The observed process improvements at the case company can be grouped in terms of requirements management, release planning, customer involvement, tooling, and development improvements. In order to evaluate the proposed template, the results of the case study have been described in five method increment cases; preparing the introduction of Scrum, the introduction of Scrum, adjusting the Scrum implementation, improving requirements management tool support, and improving customer involvement. These cases were all described using a method increment case description. However, due to space limitations, we only present a part of these descriptions here.

Table 4.3.: Method increment case for requirements management tool Support

NAME	IMPROVING THE REQUIREMENTS MANAGEMENT TOOL SUPPORT
Goal in context	# Put requirements at disposal of all relevant stakeholders # Improve identification and organization of requirements
Scope	Requirements Management and Tooling.
Stakeholders	# Management Team # Consultants # Support Specialist
Trigger	Requirements gathering and communication is inadequate
Pre-Conditions	# A simple tool exists to centrally store and organize requirements and keep track of basic information. # Inadequate implementation of RG:B.
Post-Conditions	Better Req. Mngmt. maturity in the SPM maturity matrix. - Improved Req. Gathering, Identification and Organization. - Improved Scope Change Management.
Increment Path	1. Disbanding of the requirements organization tool *Debby* - *Driver*: Tool has become outdated, unsupported with an increased risk of failure, and unused by employees. - *Stakeholders*: Managing Director, Requirements Gatherers - *Affected Capabilities*: Remove RG:B; RO:A; RO:B. 2. Introduction of (hardcopy) leaflets to store the requirements (as user stories) - *Driver*: Requirements were not documented anymore and got lost - *Stakeholders*: Product Manager, Requirements Gatherers. - *Affected Capabilities*: Add RI:A. *Further details omitted due to space restrictions*
Unordered Increments	-
Failed Paths	-
Reference to PDD	Figure 4.1

Table 4.3 shows a method increment case description for the tooling improvements. The goal during this improvement effort was to improve the requirements management process by putting the requirements at the disposal of all relevant stakeholders instead of only the product manager, and to improve the amount of information available for each requirement. In this specific method increment

Table 4.4.: Method increment case description for adjusting the Scrum process

NAME	ADJUSTING THE SCRUM DEVELOPMENT PROCESS	
Goal in context	#	Put requirements at disposal of all relevant stakeholders
	#	Improve identification and organization of requirements
Scope		Development Process (Scrum)
		. . . Details omitted due to space limitations. . .
Unordered Increments	#	Include "attitude towards Scrum" in yearly employee assessments. - *Driver*: Increase employee motivation towards Scrum. - *Stakeholders*: Management Team; Developers.
	#	Initiate a continuous improvement project. - *Driver*: Employees needed time apart from the sprints to improve processes, so as to increase employee motivation towards Scrum. - *Stakeholders*: Management Team; Developers.
Failed Paths	#	Removing the daily stand-up meetings. - *Driver*: Decrease resistance from development (who felt restricted and controlled by the daily stand-up meetings). - *Stakeholders*: Scrum Master. - *Failure Reason*: Daily stand-up meetings were considered essential in the Scrum process by the management board.

case description, we did not see any failed paths. Some tools were introduced and later replaced, but they were an essential step in the increment path. Furthermore, the increments were all executed in a logical flow which is the reason that no unordered increments have been described.

To demonstrate that these elements are in fact relevant, we provide an example of another method increment case description in Table 4.4, which is related to the adjustments made to Scrum after its initial introduction. Under *Unordered increments* we see some continuous activities, while under *Failed paths* we see an example of a change that was deemed unsuccessful.

In addition, we applied the PDD modeling technique described in Section 4.3.2 to the described process improvements. This has resulted in the PDD increment described in Figure 4.1. The PDD corresponds with the changes in Table 4.3.

Describing process improvements using the method increment case description template leads to a better insight into the drivers behind a specific process improvement effort. It provides insight into the possible issues related to certain process improvements and it provides a direct link between specific goals, detailed process

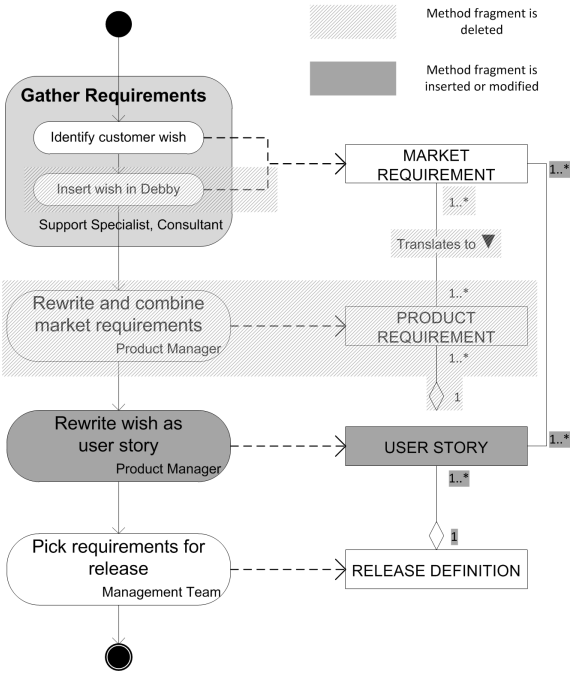


Figure 4.1.: PDD of the method increment between the initial and final situation

attributes (such as the SPM capabilities), pre-conditions and post-conditions. Ultimately, it facilitates knowledge sharing by standardizing the gathering of relevant process improvement information.

The combination of process improvement modeling using PDDs with schematic improvement modeling according to the method increment case definition allows for a thorough and semantically rich approach to the description of process improvements. By coupling a PDD with the method increment case description, it is easier to gain quick insight into the impact of certain changes. It enhances communication and it facilitates correct documentation of the processes.

4.5 CONCLUSIONS & FURTHER RESEARCH

The research question of this article was *"How can multiple increments that are part of a larger evolutionary process improvement be documented conveniently?"*. The question is answered by proposing a template to describe multiple increments

part of a larger process improvement. The template is based on use case descriptions and is referred to as a *method increment case description*. The advantage of the resemblance with use case descriptions is that the technique is easy to learn and very recognizable, as use cases are widely applied in the information sciences. The template provides a high-level overview of the process improvement, detailing the individual steps with more low-level descriptions.

Evaluation of this new documentation method has been performed using the data gathered during a case study within a small Dutch software company. Two cases of large process improvements were examined resulting in five method increment cases. Descriptions of two cases of large process improvement were examined resulting in five method increment cases. One of the cases is described in this article. The case study results indicate that the method increment case description template is adequate to document these improvements in a clear and concise way. The final template, after evaluation, is shown in Table 4.1.

The method increment case description presented in this paper is part of a larger project that aims at developing an online knowledge system for incremental process improvement (see Chapter 6–Chapter 8). Method increments play an essential role within this infrastructure, but we have inadequate means to describe them at this point. An important challenge lies in formalizing the link between the modeling approach employed in Section 4.3.2 with the schematic description of method increments using method increment case descriptions. Furthermore, we foresee two types of use for method increment case descriptions. The first use is in retrospect, where it can be a tool to record experiences from practitioners. This experience forms a key component in situational process improvement. The challenge of this approach will lie in motivating practitioners to spend time on documenting past experiences. They will need a strong incentive to do so, and we think that the creation of a public process improvement Knowledge Management (KM) system can provide this incentive. On the other hand, method increment case descriptions can form the basis for the description of future process improvements. They can be used to structure relevant knowledge required to implement process changes step by step. However, this approach has not yet been investigated.

In addition, the approach has only been used in the SPM domain. This has resulted in the usage of the SPM Maturity Matrix for coupling the consequences of the individual improvement steps to the influenced SPM capabilities. When using the template for other domains, other maturity measurements will need to be used or developed. Full validation of this template has not been achieved by

performing the case study. Especially external validity cannot be ascertained. Full validation should therefore be part of future research.

5

FINDING OPTIMAL PLANS FOR INCREMENTAL METHOD ENGINEERING

Incremental Method Engineering (ME) proposes to evolve the information systems development methods of a software company through a step-wise improvement process. In practice, this approach proved to be effective for reducing the risks of failure while introducing method changes. However, little attention has been paid to the important problem of identifying an adequate plan for implementing the changes in the company's context. To overcome this deficiency, we propose an approach that assists analysts by suggesting—via automated reasoning—optimal and quasi-optimal plans for implementing method changes. After formalizing the Process-Deliverable Diagrams (PDDs) language for describing the method changes to implement, we present a planning framework for generating plans that comply with different types of constraints. We also describe an implementation of the modeling and planning components of our approach.

This work was originally published as:

Vlaanderen, K., Dalpiaz, F., and Brinkkemper, S. (2014). "Finding Optimal Plans for Incremental Method Engineering. Manuscript accepted for publication." In: *International Conference on Advanced Information Systems Engineering*

5.1 INTRODUCTION

Incremental Method Engineering (ME) (Rossi, Ramesh, Lyytinen, et al., 2004; van de Weerd, Brinkkemper, and Versendaal, 2007) is a paradigm that proposes to change information systems development methods through a continuous improvement process. This strategy adheres to the common understanding that spreading changes over a time period is less risky and more efficient than introducing them all at once.

Empirical studies have provided significant evidence in favor of incremental ME (Diaz and Sligo, 1997; Pino, Pedreira, García, et al., 2010; van de Weerd, Brinkkemper, and Versendaal, 2007), including cases on the introduction of Scrum (see Chapter 3), showing that it helps to cope with the major obstacles to change, including resistance to change, fear of ineffectiveness by the involved personnel, and resource constraints (Baddoo, 2003).

However, existing research has largely ignored the relevant problem of identifying a plan that specifies *when* to implement the changes. This requires defining which are the most (and least) urgent changes, how many changes can be implemented given time and budget constraints, and when a plan is better than another. See the following example.

> A software organization wants to improve customer satisfaction by introducing the Kano analysis (Kano, Seraku, Takahashi, et al., 1984). However, introducing and learning the Kano analysis requires a significant effort, due to its complexity (Figure 5.4). The management team is concerned with upfront costs and risks of resistance to change. A product manager suggests introducing it incrementally, but she cannot devise a proper plan. How can Kano analysis be embedded within the current process? Are there any variations possible? □

In this paper, we address the problem of devising a plan for the incremental implementation of a set of method changes in an organization (as elaborated in Section 5.2). We use automated reasoning techniques for generating *optimal* and *quasi-optimal* plans. We propose a formalization of the Process-Deliverable Diagram (PDD) modeling language (van de Weerd and Brinkkemper, 2008), that we use to describe the changes to be implemented.

We combine these elements into a method that, based on a description of the changes to be enacted, assists the analysts by suggesting possible plans, and helps

to refine these plans to improve fitness with the organizational context. These plans have to satisfy mandatory constraints, and should *satisfice* (Simon, 1956) weak (nice-to-have) constraints.

While we are inspired by automated planning techniques, we develop a novel solution that copes with the specificities of method engineering. This includes defining sequencing based on deliverables rather than activities, and using weak constraints to derive a plan leading to an incremental maturity growth in the organization.

After stating our problem in Section 5.2, and discussing our research baseline in Section 5.3, we propose the following contributions:

- We formalize the PDD modeling language, adding clear semantics, so as to make it usable for automated reasoning. This formalization is described in Section 5.4.
- We define a formal framework that defines optimal and quasi-optimal plans with respect to an input PDD, an organizational context, and a set of time and budget constraints. The framework in Section 5.5.1 forms the basis for plan generation.
- We propose a process that guides the analysts in applying the framework to generate plans and to refine them by strengthening and relaxing constraints (Section 5.5.2).
- We develop tool support for our method: PDDmodels can be created via a graphical modeling environment, and automatically converted into input for a logic program in disjunctive Datalog (Leone, Pfeifer, Faber, et al., 2006) that generates (quasi-)optimal plans (see Section 5.6).

Section 5.7 illustrates our approach using the scenario in Example 5.1. Section 5.8 discusses related work, presents conclusions, and outlines future directions.

5.2 PROBLEM STATEMENT

Our research context is method evolution, i.e, the process through which an organization's methods change over time. We focus on incremental method engineering, a well-defined process for managing and incrementally introducing method changes.

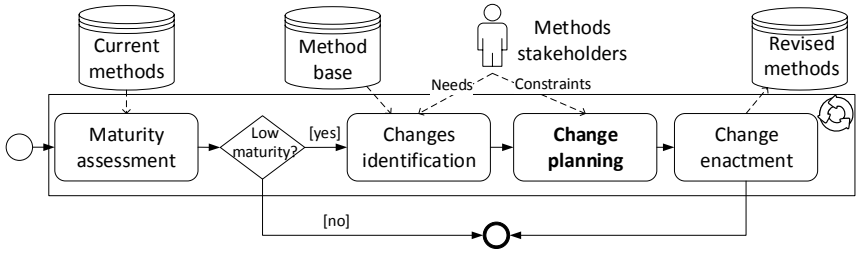


Figure 5.1.: Research context: systematic method evolution illustrated

This approach is illustrated in Figure 5.1. Whenever the process is triggered (either by an occurred event, or every N months/years), the maturity of the current processes is assessed. If any process shows low maturity, the stakeholders' needs are considered to identify *what* to change, and, subsequently, by defining a plan that specifies *how* and *when* to deploy these changes in the organization. Finally, the changes are enacted as per the plan. This process is highly iterative, for organizations are in constant evolution.

In this paper, we focus on the important yet under-explored activity of *change planning*. The problem is that of determining a (quasi-)optimal scheduling for implementing the changes, based on all the constraints (hard and soft ones) of the stakeholders. To address this non-trivial problem, several sub-questions have to be answered:

- Which factors determine the optimality of a plan?
- Which elements describe the hard constraints that cannot be violated?
- Which factors determine the priority of changes?
- How can one ensure that, even when partially deployed, the introduced changes can be effectively used in the organization?

In this paper, we consider the gradual introduction of new methods. In reality, it is more common to deal with changes to existing methods, which also requires to consider the removal and replacement of fragments. We leave dealing with the more complex case of decrements (as opposed to increments) for future research.

5.3 BASELINE

We approach the challenges in Section 5.2 as part of the Online Method Engine (OME) (see Chapter 6–Chapter 8), a Knowledge Management (KM) system for incremental method engineering. The OME consists of a method base that contains Method Fragments (MFs) (generic descriptions of common approaches to software development tasks), rules for combining fragments, and organizational experience related to the fragments. These elements feed the four main functions of the OME: (a) disseminating method knowledge; (b) assessing the maturity of an organization's processes; (c) suggesting improvements based on MFs and experience from similar organizations; and (d) enacting improvement proposals.

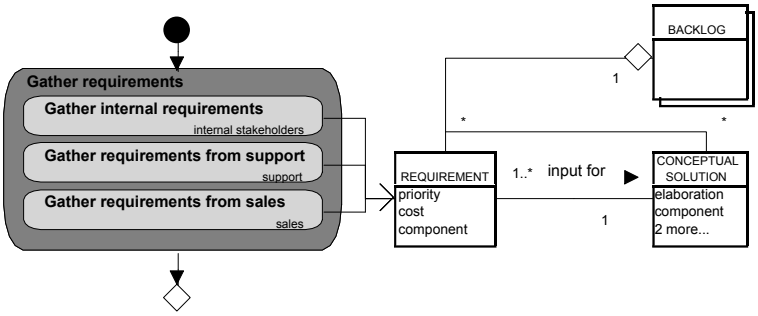


Figure 5.2.: Example of a PDD (excerpt)

PDDs are a fundamental component of the method base: MFs are modeled using a combination of UML activity diagrams—to describe the procedural aspects—and UML class diagrams—to express the data aspects (using classes called 'deliverables'). These models are connected through a 'results in' relationships from the activities to the deliverables. PDDs also distinguish between simple, closed, and open activities and deliverables (van de Weerd and Brinkkemper, 2008). Figure 5.2 illustrates the core components of a PDD through an example. More details can be found in van de Weerd and Brinkkemper (2008).

We also adopt the notion of focus area maturity matrix from the Situational Assessment Method (SAM) (Bekkers, Spruit, van de Weerd, et al., 2010) (see Table 5.1). This matrix is filled in based on questions concerning situational factors as well as organizational capabilities. Each of these capabilities (with level A-F) contributes to the maturity of the organization in a specific focus/process area. For example, capability *A* in the focus area (row) 'Requirements gathering' cor-

responds to a basic registration of the requirements, which contributes to maturity level 1, while capability F in the same area corresponds to the involvement of partners in the product management process, which contributes to maturity level 8.

By using a focus area maturity matrix instead of a fixed level maturity matrix, we are able to suggest more detailed improvement suggestions (Bekkers, Spruit, van de Weerd, et al., 2010). The SAM enables assessing the maturity level of an organization and identifying the desired (target) maturity level.

Table 5.1.: Example capability maturity matrix (excerpt)

FOCUS AREA		MATURITY LEVELS										
Title	*Code*	0	1	2	3	4	5	6	7	8	9	10
Requirements Management												
Requirements Gathering	RG		A		B	C		D	E	F		
Requirements Identification	RI			A			B		C			D
Requirements Organizing	RO				A		B		C			

5.4 PDDS FOR PLANNING METHOD CHANGES

We formalize the relevant parts of the PDD language (van de Weerd and Brinkkemper, 2008) so as to make it usable for automated reasoning. We state the requirements for our refinement in Section 5.4.1, and we present a revised metamodel and its semantics in Section 5.4.2.

5.4.1 *Requirements*

We want to leverage previous work:PDDsare a simple yet expressive means to model a method fragment's activities, deliverables, and their relationships. This choice is made to reuse the method base of fragments (modeled as PDDs) within the OME system.

PDDs were designed as a means to intuitively communicate methods and fragments to users. The price of this choice is that some constructs have ambiguous semantics that cannot be readily used for automated reasoning. Thus, we need to

provide a clear semantics of the PDD metamodel, by fulfilling the three requirements ($\mathbf{R_1}$ to $\mathbf{R_3}$) below.

$\mathbf{R_1}$*: Avoid generic associations.* Consider a directed association between two deliverables: 'customer wish' and 'theme'. The original PDD metamodel does not specify a detailed semantics for associations, which inhibits determining the nature and the strength of the link. Thus, we require our language to avoid generic associations.

$\mathbf{R_2}$*: Include input and output relationships.* PDDs do not express input relationships, i. e. that an activity needs to use a deliverable. This choice eases readability, but does not express when a deliverable is needed. This obstacles planning for change: an activity that requires a deliverable cannot be introduced until that deliverable is produced. Our language needs to unambiguously express input and output relationships.

$\mathbf{R_3}$*: Distinguish deliverable dependency types.* Consider the following dependency between deliverables: the priority of a requirement is based on the input from customers and partners, but one of them suffices. This can be represented as an association between a requirement and an aggregated *input* deliverable in PDDs, but the semantics are not sufficiently detailed to indicate the choice. We thus require our language to support more specialized deliverable relationships.

5.4.2 *PDD Metamodel*

In Figure 5.3, we present a refinement of the part of the PDD metamodel (van de Weerd and Brinkkemper, 2008) that relates with describing method change. The semantics of the language should enable describing *what* changes have to be introduced, and expressing the dependencies between deliverables that pose constraints on *when* the different activities are introduced.

Since we are interested in the implementation order of a set of activities, rather than in their execution order, our metamodel does not restrict the process side of the PDD: *ControlFlowElement* in Figure 5.3 is a generic placeholder for all the control flow constructs of PDDs (sequence, decision point, fork and merge, etc.).

ContributesTo: this relationship indicates that an *Activity* contributes to a certain *Capability*, thus helping the organization to reach that capability's maturity level. An activity can contribute any number of capabilities, and a capability can be contributed by any number of activities. Graphically, contributions are textual annotations 'FocusAreaCode:Capability' on the left of activities. In Figure 5.4,

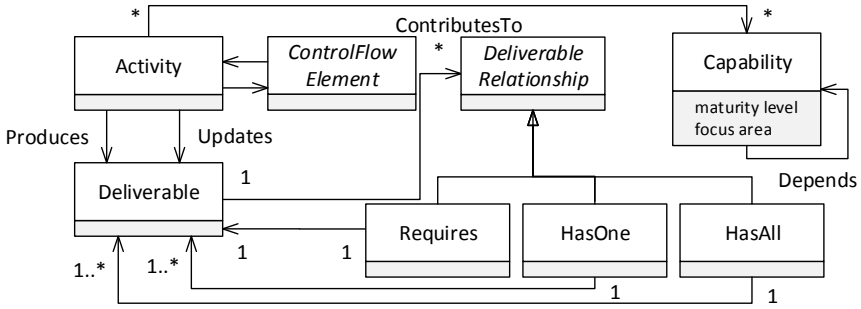


Figure 5.3.: Partial metamodel of the PDD language for describing method changes

e. g. activity 'Analyze Product Environment' contributes to capability D of focus area 'Requirements Gathering'.

We implement several changes to fulfill $\mathbf{R_1}$–$\mathbf{R_3}$. We remove the distinction between simple, complex, and closed deliverables, as these concepts are not useful for our purposes. Also, we replace generic associations ($\mathbf{R_1}$) with the following relationships.

Requires: a transitive and asymmetric binary relationship between deliverables D_1 and D_2, indicating that the D_1 can be produced only when D_2 already exists. In Figure 5.4, deliverable 'Functional Questions' requires 'Requirements'.

Produces/*Updates*: in a PDD, activities can be linked to deliverables only through the 'results in' relationship between activity A and deliverable D. Here, we specialize this relationship into *Produces*, to denote that A creates a previously non-existent deliverable, and *Updates*, to indicate the modification of a previously available deliverable. In Figure 5.4, activity 'List Results of Individual Criteria' produces deliverable 'Table of Results', while activity 'Evaluate Answer Frequencies' updates it.

Together, the *Requires*, *Produces*, and *Updates* relationships satisfy $\mathbf{R_2}$: *Requires* (D_1,D_2) denotes that any activity that produces or updates D_1 needs D_2 as an input; *Produces*(A,D) indicates that A has output D; and *Updates*(A,D) specifies that D is both an input and output for A.

HasOne/*HasAll*: to allow for fine-grained deliverable dependencies ($\mathbf{R_3}$), we specialize aggregation into the *HasOne* and *HasAll* relationships. The former indicates that at least one of the parts of a deliverable shall be available in order to produce the deliverable itself, while the latter requires that all of its parts are available. In Figure 5.4, 'Requirements' requires both (*HasAll*) 'Customer Requirements' and 'Product Requirements'; on the contrary, 'Questionnaires' requires at

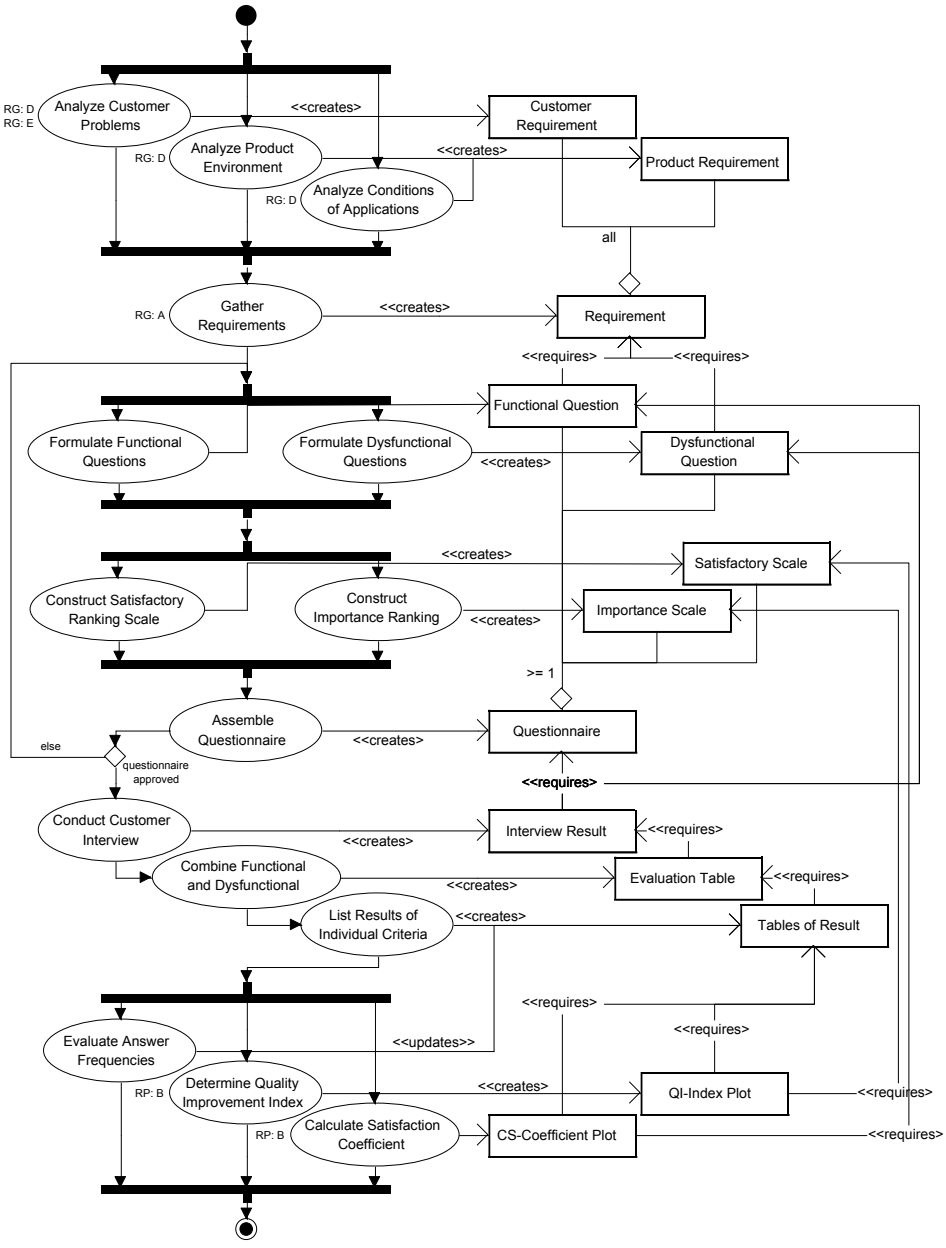


Figure 5.4.: PDD for the KANO analysis

least one among 'Functional Questions', 'Dysfunctional Questions', 'Importance Scales', and 'Satisfactory Scales'.

5.5 PLANNING FOR METHOD EVOLUTION

We present the formal framework that derives plans for implementing a set of changes described in aPDD(Section 5.5.1). We introduce a process for determining the most adequate plan (Section 5.5.2), and illustrate our approach on the Kano analysis scenario (Section 5.7).

5.5.1 *Formal Framework*

We introduce the necessary elements to define the plan generation function *genplans*, which returns all plans for implementing a set of changes, given a set of constraints. We represent time instants via natural numbers: typically, 0 would denote the current time (CT), 1 would denote CT + X milliseconds, 2 would denote CT + 2·X milliseconds, etc.

> *Implementation cost* is defined by a function $cst : \mathcal{ACT} \times \mathcal{ORG} \rightarrow \mathbb{R}^+$ s.t. $cst(\mathsf{A}, \mathsf{Org})$ is the cost of implementing activity A in the organization Org. □

The notion of cost here goes beyond monetary expenses, and it includes additional costs for the organization, including the learning effort, risk of failure, impact on motivation, time, etc. We measure cost in terms of abstract cost units; the definition of a mapping that reduces real cost to units is beyond the purpose of this paper.

> A *scheduling schema* is a list of time slots $(\langle \mathsf{St}_1, \mathsf{End}_1, \mathsf{Bdg}_1 \rangle, \ldots, \langle \mathsf{St}_n, \mathsf{End}_n, \mathsf{Bdg}_n \rangle)$ wherein changes can be implemented. The i-th time slot $\langle \mathsf{St}_i : \mathbb{N}_0, \mathsf{End}_i : \mathbb{N}_0, \mathsf{Bdg}_i : \mathbb{R}^+ \rangle$ starts at time St_i, ends at time End_i, and has budget Bdg_i. For all i, it is required that $\mathsf{St}_i \leqslant \mathsf{End}_i$ and $\mathsf{End}_i < \mathsf{St}_{i+1}$. □

A scheduling schema defines the time slots within which changes are to be implemented. Each slot has a budget, i. e. an upper bound on the implementation cost in that slot's time frame. The specification of a scheduling schema depends

on the characteristics and strategy of the organization, and on the changes to implement. For example, an organization may define a linear schema where all slots have the same duration and budget, while another may start with low effort, and increase it in later slots.

> Given a PDD model Mdl and a scheduling schema $(\langle \mathsf{St}_1, \mathsf{End}_1, \mathsf{Bdg}_1 \rangle, \ldots, \langle \mathsf{St}_n, \mathsf{End}_n, \mathsf{Bdg}_n \rangle)$, a set of *scheduling constraints* Cstr defines temporal restrictions on the allocation of the activities and the production of the deliverables of Mdl with respect to a time $\mathsf{T} : \mathbb{N}_0$:
>
> - $\mathsf{actBefore}(\mathsf{A}, \mathsf{T})$: activity A shall be scheduled in a slot i, s.t. $\mathsf{End}_i < \mathsf{T}$;
> - $\mathsf{delBefore}(\mathsf{D}, \mathsf{T})$: deliverable D shall be produced in a slot i, s.t. $\mathsf{End}_i < \mathsf{T}$;
> - $\mathsf{actAfter}(\mathsf{A}, \mathsf{T})$: like actBefore, but $\mathsf{St}_i > \mathsf{T}$;
> - $\mathsf{delAfter}(\mathsf{D}, \mathsf{T})$: like delBefore, but $\mathsf{St}_i > \mathsf{T}$;
> - $\mathsf{actAt}(\mathsf{A}, \mathsf{T})$: activity A shall be scheduled in a slot i, s.t. $\mathsf{St}_i \leqslant \mathsf{T} \leqslant \mathsf{End}_i$;
> - $\mathsf{delAt}(\mathsf{D}, \mathsf{T})$: deliverable D shall be produced in a slot i, s.t. $\mathsf{St}_i \leqslant \mathsf{T} \leqslant \mathsf{End}_i$. □

Scheduling constraints enable imposing fine-grained temporal restrictions on the allocation of activities and on the production of deliverables. In Section 5.7, we will show how these constraints are a useful tool in our method to identify the most suitable plan.

> Given a PDD model Mdl and a scheduling schema Schema, a *plan* is a set $\mathsf{Pln} = \{\langle \mathsf{A}_1, \mathsf{T}_1 \rangle, \ldots, \langle \mathsf{A}_n, \mathsf{T}_n \rangle\}$ such that:
>
> - $\mathsf{A}_1, \ldots, \mathsf{A}_n$ are all and only activities in Mdl, and
> - for each i, $1 \leqslant i \leqslant n$, there exists exactly one slot $\langle \mathsf{St}_j, \mathsf{End}_j, \mathsf{Bdg}_j \rangle$ in Schema such that $\mathsf{St}_j \leqslant \mathsf{T}_i \leqslant \mathsf{End}_j$. □

A plan is an allocation of all and only the activities of a PDD model into a scheduling schema. The activities shall be allocated within exactly one slot.

We say that a plan is *feasible* if it respects budget constraints, i. e. if for each slot, the sum of the implementation costs of the activities in that slot does not

exceed the budget. We say that a plan is *contiguous* when all slots have at least one allocated activity. In this paper, we are concerned with the generation of feasible and contiguous plans.

The function *genplans* brings together the concepts defined above and generates the feasible and contiguous plans for implementing a set of changes described by a PDD model in an organization, according to a specified scheduling schema, additional temporal constraints, and considering a cost function for implementing the changes.

> *Plan generation* is a function that returns all feasible and contiguous plans for implementing a set of changes in an organization. Formally, $genplans : pdd \times ORG \times CSTF \times SS \times CSTR \times \rightarrow 2^{PLN}$, and $genplans(\mathsf{Mdl}, \mathsf{Org}, cst, \mathsf{Schema}, \mathsf{Cstr}) = \{\mathsf{Pln}_1, \ldots, \mathsf{Pln}_n\}$ is such that:
>
> - Mdl is a description of the changes to implement in PDD;
> - Org is the organization where the changes are implemented;
> - *cst* defines the cost of implementing the activities of Mdl in Org (Def. 5.5.1);
> - Schema is a scheduling schema (Def. 5.5.1);
> - Cstr is a set of constraints on scheduling (Def. 5.5.1);
> - for each i, $1 \leqslant i \leqslant n$, Pln_i is a feasible and contiguous plan (Def. 5.5.1) for Mdl and Schema such that (i) Pln respects all constraints in Cstr; and (ii) Pln complies with the deliverable and capability dependencies in Mdl. □

While *genplans* deals with hard scheduling constraints that cannot be violated, it does not consider the optimality of a plan. In this paper, we conceive plan optimality in terms of *incremental maturity improvement*: the changes should be introduced in accordance with a growing maturity level of the organization. The activities that contribute to capabilities with lower maturity levels shall be introduced first, followed by the activities contributing to higher maturity levels, up to the highest maturity. Def. 5.5.1 introduces the notion of penalty for a plan, i. e. its distance from an optimal plan where activities are introduced with a monotonic increasing level of maturity.

Plan penalty is a function that returns the distance between a given plan and an optimal plan that would introduce all activities in increasing order of maturity. Let the predicate $\mathit{preceeds}(\mathsf{A}, \mathsf{A}', \mathsf{Pln})$ indicate that activity A is scheduled before activity A' in Pln. The maturity of an activity $\mathit{mat}(\mathsf{A})$ is the lowest maturity level among the capabilities that the activity contributes to. Formally, $\mathit{penalty} : \mathcal{PLN} \times \mathit{pdd} \rightarrow \mathbb{N}_0$, s.t. $\mathit{penalty}(\mathsf{Pln}, \mathsf{Mdl}) = \sum_{\mathsf{A},\mathsf{A}':\mathit{preceeds}(\mathsf{A},\mathsf{A}',\mathsf{Pln})} \max(\mathit{mat}(\mathsf{A}) - \mathit{mat}(\mathsf{A}'), 0)$. □

When an activity contributes to multiple capabilities, we consider the capability having the lowest maturity level, for that activity is important for the organization to achieve that level. The penalty is the number of 'steps' that have been skipped in the plan: for instance, consider only activities 'Gather Requirements' and 'Analyze Customer Problems' in Figure 5.4. The former activity has lowest maturity level 1 (it contributes to capability A in area requirements gathering), while the latter has lowest maturity level 6 (the contributed capability with lowest maturity is D in requirements gathering). If the former activity is introduced before the latter, the plan penalty would be 5.

We call a plan *optimal* when its penalty is zero, and *quasi-optimal* when its penalty is greater than zero but lower than Δ. This number Δ is domain-specific, and it depends on the number of activities in the plan, the organization, the cost of activities, etc.

In this paper, we limit ourselves to a special kind of scheduling schema, where for each slots i ($1 \leqslant i \leqslant n$), $\mathsf{St}_i = \mathsf{End}_i$, and the slots are such that $\mathsf{St}_1 = 1, \ldots, \mathsf{St}_n = n$.

We do not consider constraints related to organizational resources (other than the abstract unit Cost) and human factors (such as worker resistance), as such factors are harder to attribute to single activities and deliverables. The method we propose is a support tool for analysts, and does not replace their role as decision makers.

5.5.2 *A Method for Identifying and Refining Plans*

We present an elaboration of the *Change planning* step in Section 5.2 that uses *genplans* (Def. 5.5.1) for identifying plans and for refining them to fit well with the organization at hand. This method, illustrated in Figure 5.5, helps to restrict

or widen the space of alternative plans, depending on the plans that the function *genplans* returns.

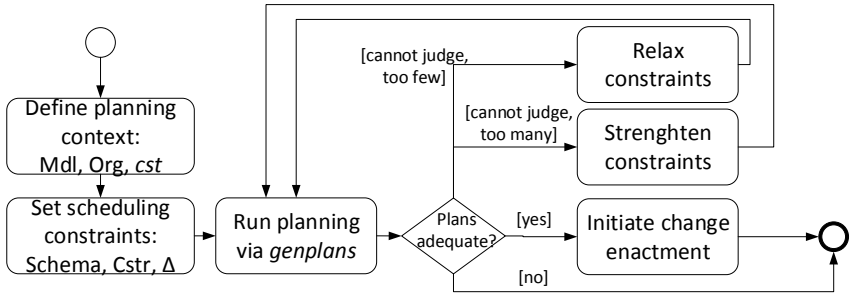


Figure 5.5.: A method for identifying and refining (quasi-)optimal plans

The process begins with two preparatory steps that provide the inputs to the *genplans* function: the planning context is defined (the changes to implement, the organization, and the cost function), and an initial version of the scheduling constraints is created (the scheduling schema and constraints, and the quasi-optimality upper bound Δ). This input feeds the *genplans* function, which returns all optimal and quasi-optimal plans. The analyst then judges the adequacy of the returned plans:

- *Adequate*: a plan is identified and *change enactment* starts (see also Figure 5.1);
- *Uncertain*: a final decision cannot be made at the moment, due to scarcity or excess of results. In order to identify a better plan, the analyst can modify the constraints:
  - *Relaxation*: when too few plans are returned, the scheduling constraints can be relaxed/weakened. Time slots can be merged to spread changes over a longer period with higher budget. Temporal scheduling constraints can be relaxed, or the quasi-optimality upper bound Δ increased. More intrusive options (from the organization's perspective) are to increase the number of slots or the budget.
  - *Strengthening*: if too many plans are returned, the analyst cannot make an informed choice, and the constraints can be tightened: the slots in the schema and their budget can be reduced, and temporal constraints can be added.

- *Inadequate*: no satisfactory plan can be devised in the current planning context, irrespective of the scheduling constraints. This terminates the process, which can be re-executed in a different context.

5.6 REALIZATION

We have developed a graphical modeling tool for PDD within MetaEdit+ (Tolvanen and Rossi, 2003). This tool is built around the Graph, Object, Property, Relationship, Role (GOPRR) metamodel, which we used to describe the PDD metamodel; instances of the latter metamodel are PDDs. The editor enforces syntactical rules, thereby ensuring the well formedness of the model. In addition, we have developed a set of code generators that transform any PDD model into a set of Datalog statements, which are needed to generate plans.

We have realized the mechanisms for identifying optimal and quasi-optimal plans via logic programming (specifically, via the disjunctive Datalog engine DLV (Leone, Pfeifer, Faber, et al., 2006)). The program consists of a set of inference rules (Table 5.2) that returns feasible and contiguous plans for a given input. The input consists of the Datalog statements that are generated from the PDD model, the temporal constraints, and the Δ quasi-optimality upper bound.

The following extensional predicates formalize in Datalog the primitives of our languages (Section 5.4 and Section 5.5): activity(A) states that A is an activity; produces(A,D) and updates(A,D) denote the relationships between activities and deliverables; requires
(D1,D2) indicates that namesake relation between deliverables in Section 5.4. The predicate atLeastOnePart(D) (allParts(D)) says that D requires at least one (all) of its parts, each part being stated through hasPart(D,D1); cost(A,N) states that the deployment of activity A costs N (a natural number); slot(N1,N2) states that slot at time N1 has a budget of N2 cost units; actAt(A,N), delAt(D,N), actBefore(A,N), delBefore(D,N), actAfter(A,N), delAfter(D,N) state constraints telling that activity A or deliverable D shall be scheduled at/before/after slot N; contributes(A,C) states that activity A contributes to capability C; depends(C1,C2) says that capability C1 shall be implemented strictly after capability C2; maturity(C,N) states that capability C has maturity level N.

Table 5.2 presents the inference rules (syntax head :- tail.) to generate plans. By default, the program returns all possible plans that satisfy the constraints. A plan is defined as a set of predicates chosen(A,N), each stating that the deployment of

Table 5.2.: Core disjunctive Datalog inference rules for deriving plans

ID	RULE DEFINITION
1	chosen(A,T) :- activity(A), slot(T,_), not chosenNotT(A,T), not missingReq(A,T).
2	chosenNotT(A,T) :- activity(A), chosen(A,T2), slot(T,_), slot(T2,_), T2!=T.
3	:- activity(A), 0=#count{T: chosen(A,T)}.
4	:- slot(T,_), 0=#count{A : chosen(A,T)}.
5	chosen(A,T) :- actAt(A,T).
6	:- delAt(D,T), T1=T-1, producedAt(D,T1).
7	:- delAt(D,T), not producedAt(D,T).
8	:- actBefore(A,T), chosen(A,T1), T1>=T.
9	:- delBefore(D,T), 0=#count{T1 : producedAt(D,T1), T1<T}.
10	:- actAfter(A,T), chosen(A,T1), T1<=T.
11	:- delAfter(D,T), 0=#count{T1 : producedAt(D,T1), T1>=T, not producedAt(D,T)}.
12	:- slot(T,B), #int(C), C=#sum{Cs,Act : cost(Act,Cs), chosen(Act,T)}, C>B.
13	missingReq(A,T) :- uses(A,D), #int(T), not producedAt(D,T), T2=#max{T1 : slot(T1,_)}, T2<=Z.
14	producedAt(D,T) :- chosen(A,T1), #int(T1), #int(T), T1<=T, produces(A,D), T3=#max{T2 : slot(T2,_)}, T3<=Z.
15	uses(A,D1) :- produces(A,D), requires(D,D1).
16	uses(A,D) :- updates(A,D).
17	requires(D1,D3) :- requires(D1,D2), requires(D2,D3).
18	hasOnePart(D) :- atLeastOnePart(D), produces(A,D), chosen(A,T), hasPart(D,D1), producedAt(D1,T).
19	:- atLeastOnePart(D), not hasOnePart(D).
20	:- allParts(D), produces(A,D), chosen(A,T), hasPart(D,D1), not producedAt(D1,T).
21	:- contributes(A1,C1), contributes(A2,C2), A1!=A2, depends(C1,C2), chosen(A1,T1), chosen(A2,T2), T1<=T2.
22	minmaturity(A,M) :- activity(A), M=#min{L : contributes(A,C), maturity(C,L)}.
23	penalty(Cs) :- #int(Cs), Cs=#sum{C,A1,A2 : chosen(A1,T1), chosen(A2,T2), T1<T2, minmaturity(A1,M1), minmaturity(A2,M2), M1>M2, C=M1-M2}.

activity A will be scheduled in slot N. Rules without a head (those starting with ':-') are integrity constraints: the condition expressed in the tail shall not become true in any model.

Rules 1–3 ensure that all activities are assigned to exactly one slot. Rule 4 guarantees that all slots have at least one assigned activity. Rules 5–7 deal with actAt

and delAt constraints: the activity (the deliverable) shall be scheduled (produced) in the specified slot. Rules 8–9 and rules 10–11 guarantee the fulfillment of the similar constraints in the before/after variant. Rule 12 ensures that the sum of the costs for implementing the activities in a slot does not exceed the slot budget. Rules 13–14 ensure that deliverables are produced before their use. Rule 15 states that an activity A uses a deliverable D1 if X produces D, and D requires D1. Rule 16 says that updating a deliverable implies using it. Rule 17 says that the requires relationship is transitive. Rules 18–20 take care of atLeastOnePart(D) (allParts(D)): at least one part of D (all parts) shall be available when D is produced. Rule 21 handles the depends(C1,C2) relationship between capabilities: all activities that contribute to C1 are scheduled strictly before any activity that contributes to C2. Rules 22–23 compute the penalty of the plan as in Def. 5.5.1.

The program can be run with different parameters to return only optimal and quasi-optimal plans. By adding the rule :~ penalty(Cs). [Cs:1], only the plans with minimum penalty are listed (':~' is a weak constraint that DLV optimizes by returning the models that minimize it). In order to return all quasi-optimal plans, the parameter '-costbound=Δ' can be specified when executing DLV.

5.7 ILLUSTRATION

We apply our planning method to the scenario of Example 5.1 and Figure 5.4. Kano analysis uses a two-dimensional quality model used for the analysis of customer requirements, which is useful to elicit customer needs about a service or product under design. It uses two types of questionnaires and an evaluation table for classifying the requirements into different categories (Kano, Seraku, Takahashi, et al., 1984). We show how the company can use our method to identify an incremental plan to introduce the 15 activities and 13 deliverables of Kano analysis.

Step 1: Define the planning context. We begin with the creation of a PDD model that describes the changes to introduce (Kano analysis, as in Figure 5.4), the organizational context (our example organization), and the cost function that returns the costs for each activity. In this example, we use a simple cost scheme, where we use natural numbers in the range [1,5] to describe the complexity of implementing and learning each activity. For each activity, we add a fact as input to our datalog program:

cost(analyze_customer_problems, 4). cost(analyze_product_environment, 2)., etc.

Step 2: Set scheduling constraints. This activity involves the specification of the number of slots for implementing the plan, and the budget for each of them. Here, we define four slots, each with a budget of nine cost units (adopting the same unit as for activities cost). We do not define any temporal constraint, and we set $\Delta = 25$.

Step 3: Run planning. When we run our planner with the settings above, we obtain 1,442 plans, with penalties between 8 and 22. Making a choice at this point would obviously be difficult; moreover, no optimal plan exists (no plan has penalty 0). Some constraints have to be introduced.

Step 4: Strengthen constraints. The company wants to have a running implementation of Kano analysis in slot 0. This requires to have at least a table of results based on requirements from the customer and the product environment. More advanced tools, such as the QI-index plots and the CS-coefficient plots, can be introduced later. To such extent, the following temporal constraints are added:

```
delAt(tables_of_results, 0).
delAfter(cs_coefficient_plots, 0).
delAfter(qi_index_plots, 0).
actAt(analyze_product_environment, 0).
actAt(evaluate_answer_frequencies, 1).
```

When we re-run the planner, we obtain no results. By forcing the planner to schedule the activity that produces the table of results at slot 0, the require relationships between the deliverables imply that several other activities have to be schedules in slot 0 as well. Their total cost exceeds the budget of slot 0. This forces us to relax some constraints.

Step 5: Relax constraints. To cope with the required effort for implementing the table of results in slot 0, the organization can either combine the effort of multiple slots, thereby lengthening the implementation time, or allocate more resources to slot 0. We assume the analyst opts for the latter: the budget of slot 0 is raised to 20, but a slight reduction in the overall budget is required (-4 units); the remaining units are allocated in two slots with budget 6, and the last slot is removed.

When we re-run our planner, we obtain 5 plans. Table 5.3 shows them and outlines their differences via a gray background color. Two of them have a penalty of 12, the other three have a penalty of 8. The analyst is free to consider a restricted

Table 5.3.: Slot allocation for the activities of Kano analysis: the (quasi-)optimal plans in Step 5

ACTIVITY	A	B	C	D	E
analyze_customer_problems	0	0	0	0	0
analyze_product_environment	0	0	0	0	0
analyze_conditions_of_applications	2	2	2	1	1
gather_requirements	0	0	0	0	0
formulate_functional_questions	0	0	0	0	0
formulate_dysfunctional_questions	0	0	0	0	0
assemble_questionnaire	0	0	0	0	0
conduct_customer_interview	0	0	0	0	0
combine_func._and_dysfunc._answers	0	0	0	0	0
list_results_of_individual_criteria	0	0	0	0	0
evaluate_answer_frequencies	1	1	1	1	1
construct_satisfactory_ranking_scale	1	1	2	2	1
construct_importance_ranking_scale	1	2	1	1	2
determine_quality_improvement_index	2	2	1	2	2
calculate_satisfaction_coefficient	2	1	2	2	2
penalty	8	8	8	12	12

set of plans. The choice is between introducing both plots in slot 2, implementing QI index plots first and then CS coefficient plots, or vice versa.

5.8 DISCUSSION AND FUTURE DIRECTIONS

We have presented a method that assists analysts in the planning phase of method evolution, i. e. to identify plans for implementing a set of changes in an organization. Our method enables representing changes via PDD models, and is supported by our graphical modeling tool. The method includes the automated generation of plans that comply with scheduling constraints, and that maximize incremental growth in maturity, by trying to introduce changes according to an increasing maturity level. We also propose a process that guides analysts in refining plans by strengthening and relaxing constraints.

This work is performed within the context of the OME, and it touches upon several related fields. We discuss our approach in the light of these fields.

Software Process Improvement. Research in the area of Software Process Improvement (SPI) has produced effective frameworks to determine what to change, including Capability Maturity Model Integration (CMMI) (CMMI Product Team, 2002) and Software Process Improvement and Capability dEtermination (SPICE) (Dorling, 1993). We complement these works by proposing a method for planning the implementation order of these changes.

Situational Method Engineering. This discipline deals with describing, constructing and adapting software development methods for a specific situational context, thus promoting reuse of standardized approaches while maintaining flexibility (Becker and Knackstedt, 2007; Harmsen, Brinkkemper, and Oei, 1994; Henderson-Sellers, Gonzalez-Perez, and Ralyté, 2007). In this research, we employ the MF concept (Brinkkemper, 1996) for compatibility with the OME system; however, other notations can be used, as long as they satisfy the requirements of Section 5.4.1. There has been some work related to the notion of method evolution (Ralyté, Rolland, and Ayed, 2005; Rossi, Ramesh, Lyytinen, et al., 2004). Most approaches in method evolution consist of manual activities, although some approaches support (semi-)automatic method construction (Aharoni and Reinhartz-Berger, 2011).

Automated planning. The problem of identifying a plan to reach a given goal is well-known in Artificial Intelligence (AI) (Ghallab, Nau, and Traverso, 2004). Recent planners are able to deal with sophisticated planning constraints on state trajectory, preferences, soft constraints, and plan quality (Gerevini and Long, 2005). Our approach differentiates from existing solutions in that it employs a capability-driven planning policy that takes in to account deliverable-based constraints, as opposed the activity-based constraints that are typical of AI planning. We do not preclude that an extended version of our framework could employ Planning Domain Definition Language (PDDL).

Project management. The implementation of a set of changes in the methods of an organization is usually executed in the form of a project. Project management is a very mature field, which offers effective mechanisms and tools to deal with change by planning, scheduling, and controlling it (Kerzner, 2013). Our approach is inspired by this field, but focuses on a very specific type of scheduling that relates to method change.

We have focused only on introducing new fragments; the next step is to consider the removal and replacement of fragments. We will also explore the preceding

step of *method construction*. Furthermore, we plan to convert our prototype into a comprehensive tool that supports the analysts in the plan refinement process by recommending possible refinements. We will evaluate the efficacy of our approach with case studies in the industry, and based on the feedback from practitioners, we will extend the supported constraints. Finally, we aim to assess the scalability of our reasoning techniques.

Part IV

ONLINE METHOD ENGINE

6

ON THE DESIGN OF A KNOWLEDGE MANAGEMENT SYSTEM FOR INCREMENTAL PROCESS IMPROVEMENT FOR SOFTWARE PRODUCT MANAGEMENT

Incremental software process improvement deals with the challenges of stepwise process improvement in a time where resources are scarce and many organizations are struggling with the challenges of effective management of software products. Effective knowledge sharing and incremental approaches are essential for improving the success rate of process improvement efforts. During recent years, we have worked on the development of a Knowledge Management (KM) system, the Online Method Engine (OME), that enables incremental, situational process improvement in the field of Software Product Management (SPM). This has resulted in an initial system design. In this paper, we describe the findings from seven exploratory case studies on incremental process improvement. The lessons learned during these case studies are used to refine the design of the OME.

This work was originally published as:

Vlaanderen, K., van de Weerd, I., and Brinkkemper, S. (2012). "On the Design of a Knowledge Management System for Incremental Process Improvement for Software Product Management." In: *International Journal of Information System Modeling and Design (IJISMD)* 3.4, pp. 46–66

6.1 INTRODUCTION

Software Process Improvement (SPI) has been the subject of many scientific undertakings during the last decades. SPI is not a simple, one-time activity. Changing an organizational structure and its rules and processes has tremendous impact on that organization. During each process improvement effort, several aspects play a role, such as the organizational culture, technical support, and human capabilities (Basili and Green, 1994; Shih and Huang, 2010). Because requirements for SPI efforts are often incomplete, contradictory, ever-changing, and difficult to recognize, and because there is no perfect solution that fulfills all requirements, we can classify SPI as a wicked problem (Churchman, 1967).

Recent SPI literature shows appropriate attention for the wicked nature of SPI. Software engineers have been working with different process improvement models such as Capability Maturity Model (CMM) (Paulk and Curtis, 1993), Software Process Improvement and Capability dEtermination (SPICE) (Dorling, 1993) and the more bottom-up approach Quality Improvement Paradigm (QIP) (Basili, 1993). Unfortunately, lack of resources, time pressure, and the difficult nature of change often prevent successful process improvement efforts (Baddoo, 2003).

One key success factor that is mentioned in many studies is the need for iterative and incremental improvement. A systematic literature review revealed that process improvement programs need to be guided in "an iterative and incremental approach (...) that allows a continuous adoption of improvement practices" (Pino, García, and Piattini, 2008, p. 50). In addition, Sawyer, Sommerville, and Viller (1997) state that process improvement should not be seen as a one-step process, but as a sequence of several improvement cycles in which good practice can be introduced in the organization. An advantage of incremental improvement as opposed to revolutionary improvement is that it is a fundamental way to reduce risk on complex improvement projects (Krzanik and Jouni, 2002).

From a Knowledge Management (KM) perspective, incremental improvement has specific advantages as well. Experience suggests that "companies can institutionalize incremental improvement (...) with those doing the work identifying and implementing small changes in product and process" (Davenport, 1993). Furthermore, the introduction of KM in the software development domain led to the new area of experience based process improvement (Sharma, Singh, and Goyal, 2010; Sharma, Singh, and Goyal, 2011). In this approach knowledge that is created during software processes can be captured, stored, disseminated, and

reused, so that better quality and productivity can be achieved (Sharma, Singh, and Goyal, 2010).

Research during the past decade has shown a need for process improvement support that takes the situation of the organization into account, enables incremental implementation of improvements, and that pragmatically leverages existing knowledge and experience (Pino, García, and Piattini, 2008; Sulayman, Urquhart, Mendes, et al., 2012). Unfortunately, although experience management is gaining interest and there is increasing support for building knowledge bases (García, Amescua, Sánchez, et al., 2011), there is no proof that current method bases and knowledge infrastructures are effective. Practitioners do not always know exactly what they are looking for, or how to apply a formal method description to the processes of their organization (Niazi, 2011).

During the past years, we have designed an approach to process improvement that addresses the issues of evolutionary improvement, situationality, and knowledge dissemination. This has resulted in the design of the Online Method Engine (OME); a KM tool for incremental SPI. The OME is based upon various other research projects. The design process of the OME can be described in terms of the Systems Development Research Process (SDRP) by Nunamaker Jr., Chen, and Purdin (1990) (Figure 6.1).

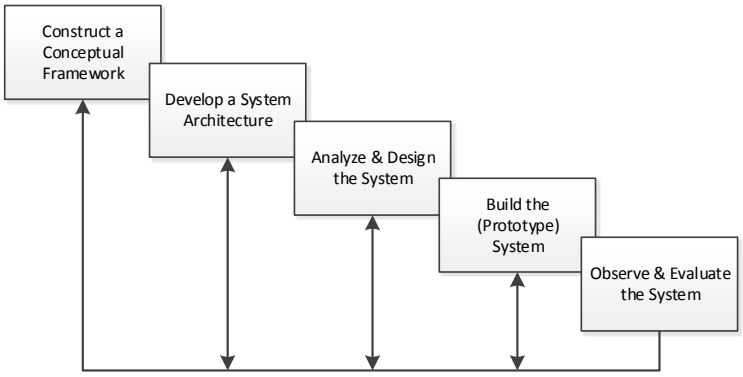


Figure 6.1.: Systems development research process

As consistent with the SDRP, the analysis, design and development of the OME are ongoing activities. This means that the conceptual framework, the system architecture and the design of the OME are constantly changing. However, a significant body of research has been published in recent years that forms the conceptual framework on which the development of the OME is based. The main

purpose of the next section is to outline this conceptual framework and the resulting initial system architecture. Consequently, we have integrated the related literature discussion with a description of the initial OME design. In the following section, we further analyze the problem and solution space based on the findings from seven exploratory case studies, resulting in an enhanced understanding of the solution space. We describe how these lessons affect the OME design in the section 'Design Impact', followed by some conclusions and pointers towards further research.

6.2 TOWARDS INCREMENTAL PROCESS IMPROVEMENT SUPPORT

The design process of the OME started in 2006 with the design of a KM system for process improvement in the domain of Software Product Management (SPM) (the Product Software Knowledge Infrastructure or Product Software Knowledge Infrastructure (PSKI)) described by van de Weerd, Versendaal, and Brinkkemper (2006). This approach combined an incremental process improvement approach with a method base and the instruments to gather the required data. Using, among other things, the case study results presented in this paper, the PSKI model has been refined and transformed into the functional architecture of the OME as depicted in Figure 6.2. The diagram is shown early in the text in order to structure the remainder of the paper. However, it contains both the original design choices as well as the choices based on the case studies described later on.

Each of the four layers contains a set of functional components, shown by rounded boxes. The layers describe the main functionality related to knowledge dissemination, method assessment, method improvement and method enactment. Each layer builds upon the layer below it.

6.2.1 *Knowledge Dissemination*

The research described in this paper is grounded in the field of Situational Method Engineering (SME). Brinkkemper (1996, p. 276) defines the term Method Engineering (ME) as "the engineering discipline to design, construct and adapt methods, techniques and tools for the development of information systems". This is later extended by Harmsen, Brinkkemper, and Oei (1994), who define a situational method as an information systems development method tuned to the situation of the project at hand.

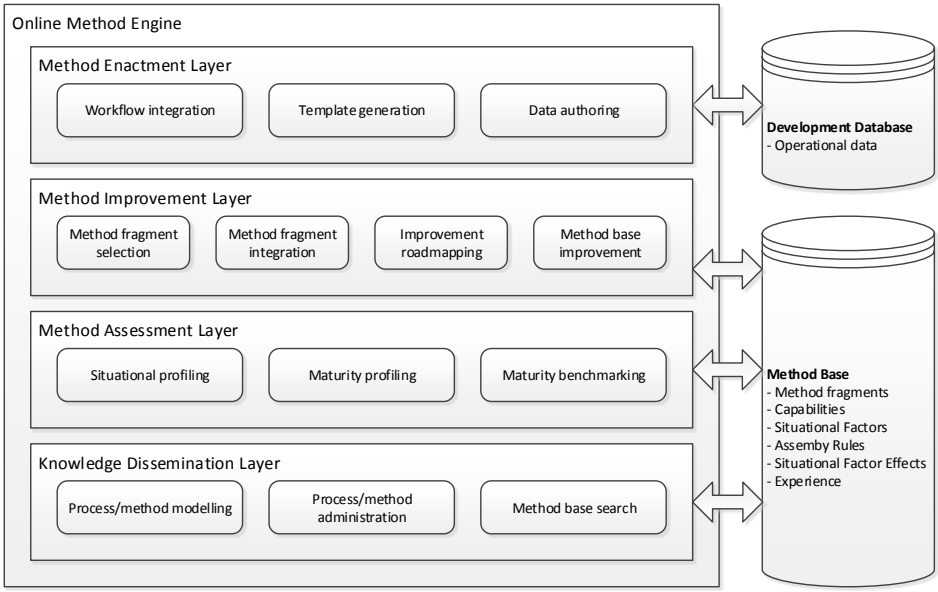


Figure 6.2.: Functional architecture for the OME

Research within the field of SME has focused on structuring available knowledge into small, reusable fragments. The area of SME has been searching for solutions that allow the construction of methods, techniques, and tools based on the specific characteristics and desires of the user. The coverage of the main approaches ranges from the modeling of methods and systems (OMG, 2008; Saeki, 1995) to the situational construction of new ones (Brinkkemper, Saeki, and Harmsen, 1998; Ralyté and Rolland, 2001) and its application to SPI (Bajec, Vavpotič, Furlan, et al., 2007). SME uses the notion of meta-model to describe, alter, and combine methods or parts thereof. During the years, several modularization constructs have been proposed for SME. Although these constructs have many aspects in common, some essential differences exist. The six main constructs are ‘Method Fragments (MFs)’, ‘method chunks’, ‘method components’, ‘method services’, ‘OPF fragments’ and ‘FIPA fragments’. Extensive comparisons of these constructs were performed by Ågerfalk, Brinkkemper, Gonzalez-Perez, et al. (2007), Cossentino, Gaglio, Henderson-Sellers, et al. (2006), Henderson-Sellers and Gonzalez-Perez (2008), and Ralyté, Deneckère, and Rolland (2003).

One of the main concepts in most ME research is that of the method base. Several researchers (Henderson-Sellers, Simons, and Younessi, 1998; Ralyté, Jeusfeld, Backlund, et al., 2008; van de Weerd, Versendaal, and Brinkkemper, 2006) describe the use of a method base in SME. Method bases are used to store MFs in order to make them available for later reuse. Once retrieved from the method base, they can be combined following a set of assembly rules, such as the rules described by Brinkkemper, Saeki, and Harmsen (1999). Some method bases have been implemented and put into practice, such as OPF (Henderson-Sellers, 2002) and the CREWS method base (Ralyté, 1999). However, there is no strong scientific proof that practitioners are able to effectively retrieve MFs from them and use those in their daily work. A prerequisite is that the method user is aware of the exact MF that he or she is searching for. In addition, the method user must know what to do with the retrieved MFs, how to interpret them, and how to implement them in the organization.

The method base concept has been used in the OME design as well, resulting in the knowledge dissemination layer. This layer uses a method base to store information about scientific methods, implemented methods within organizations, experiences with MFs, and other information relevant to process improvement within the domain of SPM. As such, it forms the backbone of the KM system.

6.2.2 *Method Assessment and Improvement*

The application of situational ME allows for a style of process improvement that is not prescriptive in nature. Prescriptive models are often criticized for being too rigid and ‘heavy’. Instead, SME allows for incremental and situational process improvement; incremental in terms of the possibility to improve the process in a series of small steps, and situational in terms of a solution adapted to the situation at hand.

Instead, our research during the last five years has mainly focused on supporting evolutionary process improvement through incremental method evolution (van de Weerd, Brinkkemper, and Versendaal, 2007). During an exploratory case study, van de Weerd, Brinkkemper, and Versendaal (2007) defined the atomic types of adaptations that can be made to an existing method, called method increments. Insight into these method increments enables focused improvement of software development processes based on localized issues and improvement possibilities. In theory, it allows the linkage of identified local problems to specific changes

to the current method. Method increments play a crucial role in our vision on situational and incremental process improvement, and therefore in the design of the OME.

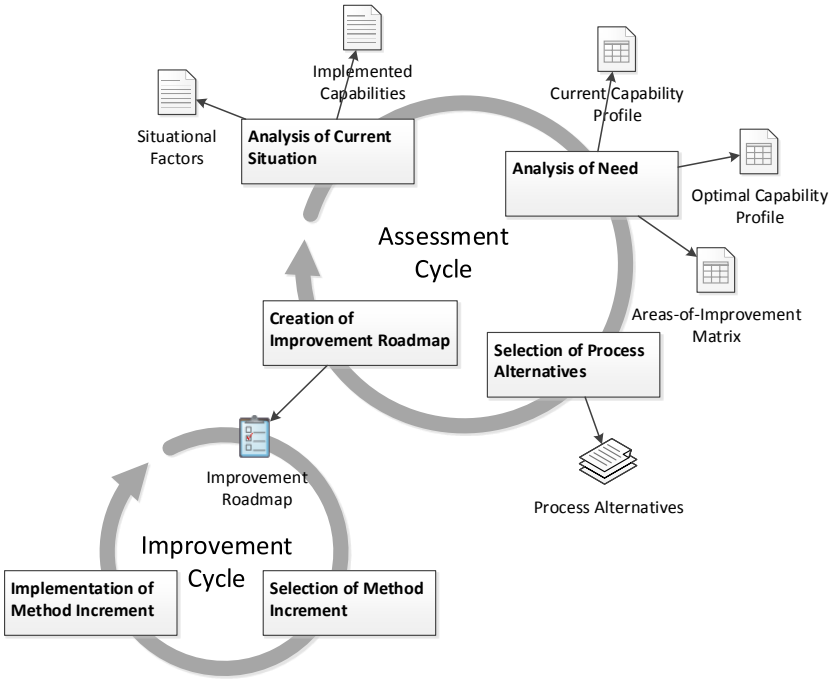


Figure 6.3.: Incremental process improvement

Based on the Situational Assessment Method (SAM) by Bekkers, Spruit, van de Weerd, et al. (2010), and influenced by process improvement approaches such as the Deming cycle of Plan, Do, Check and Act (Deming, 2000), and the QIP (or experience factory) (Basili, 1993), a high-level approach to incremental method evolution has been proposed by Vlaanderen, Brinkkemper, and van de Weerd (2011) (see Figure 6.3). The approach consists of one major process improvement cycle with one subcycle. The starting point for each process improvement project is an analysis of the current process, based on which a maturity profile can be calculated. The situational context (Bekkers, van de Weerd, Brinkkemper, et al., 2008) of the organization is used to determine an optimal maturity profile. By calculating the gap between these two, the required process improvement is determined. This process improvement is further detailed by relating it to suitable

MFs that can be combined into a new method that can be implemented to improve the organization's process.

The method assessment layer of the OME deals with the analysis of methods using situational profiling and maturity profiling. These profiles can be used during a process improvement effort, or for direct benchmarking against other organizations within industry.

The method improvement layer contains all functionality related to the translation of process improvement requirements to suitable method increments. This includes the selection of suitable MFs, the assembly of original and new MFs into a new method description, and the creation of an improvement roadmap.

6.2.3 *Method Enactment*

The top layer describes functionality related to method enactment. This includes workflow integration to synchronize method descriptions with actual process data, template generation to update existing tools such as spreadsheets and requirements databases according to the updated method, and data authoring to manage company specific process data.

Method enactment has received some attention in the ME literature, but it remains a very complex challenge. The enactment of a situational method requires the integration of multiple tools, models and documents. The ISO/IEC 24744 standard defines enactment as the "act of applying a method[ology] for some particular purpose, typically an endeavor" (ISO/IEC, 2007, p. 3). The standard also states that enactment is often performed by technical managers. Some researchers have shown that enactment can be facilitated by tools as well (Gonzalez-Perez and Henderson-Sellers, 2007; Grundy and Venable, 1996).

Within the OME, the advantages of the underpinning meta-model are used to facilitate deliverable templates and to link and adapt multiple tools. In essence, this layer is strongly related to the concept of Method/as/a/Service (MaaS), described by Deneckère, Iacovelli, Kornyshova, et al. (2008), Guzélian and Cauvet (2007), and Rolland (2009). By adopting a Service/Oriented Architecture (SOA) for method engineering, the authors aim to change MFs into method services which are implemented as Web services. Deneckère, Iacovelli, Kornyshova, et al. (2008) describes how the concept of SOA is adopted in a Method/Oriented Architecture (MOA). This MOA facilitates a method services registry in which

available method services are organized. Unfortunately, the MaaS concept is not thoroughly understood yet.

6.2.4 *Technical Architecture*

Although the underlying conceptual model is constantly being enhanced, and the system architecture and design have only been partially completed, development of a scientific prototype (cf. Figure 6.1) has started. The OME is implemented as a hot-pluggable Web Component Bundle (WCB), i. e. a set of components and services based on the concept of an Open Service Gateway Initiative (OSGi) bundle (Alliance, 2003). The WCB adds a set of custom user interface elements and components to an existing content management system, and allows one to create and maintain a specialist website containing situational method knowledge.

Based on the functional overview of the OME, a technical architecture of the system is proposed in Chapter 8 (see Figure 6.4). The technical architecture of the OME is based on components and services because, even though the design and implementation are currently guided by the needs of SPM research, the solution should be generic. The solution should be applicable to other domains by replacing, adding or extending components.

The current architecture describes the technical realization of the lower two layers of the functional framework. Most components directly reflect the functionality described above. A control panel is defined to support the collection of data, including situational data and MFs. For the modeling of MFs, we employ the MetaEdit+ (Tolvanen and Rossi, 2003) meta-modeling tool. This tool enables the definition and usage of domain-specific languages. Communication between MetaEdit+ and the OME environment is structured using XML.

6.3 ELABORATION OF THE OME DESIGN

In the previous section, we have demonstrated how earlier research on software process improvement and situational ME has contributed to the development of a rudimentary design for a KM system for incremental process improvement. This design is subject of constant refinement in order to find a solution that enables us to enhance the effectiveness of process improvement efforts in product software organizations.

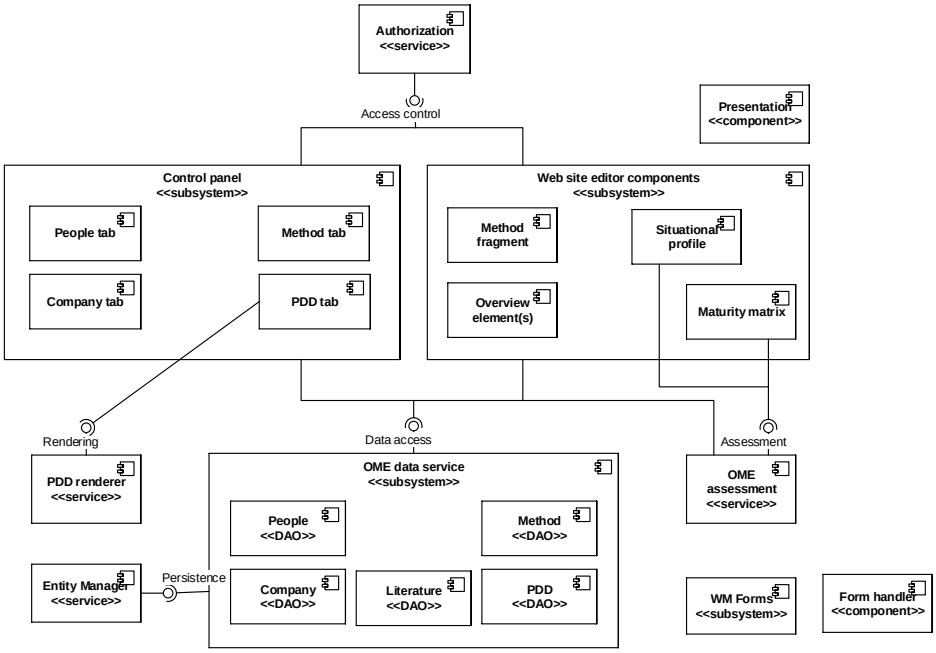


Figure 6.4.: Technical architecture for the OME

Although we are convinced that the main concepts within SPI and SME are generically applicable, research related to the development of the OME is performed within one particular domain. In order to have a clear focus, the research domain is limited to that of SPM. SPM deals with management of requirements, the definition of releases, and the definition of software products in a context where many internal and external stakeholders are involved (van de Weerd, Brinkkemper, Nieuwenhuis, et al., 2006b). The domain is described in the form of a competence model by van de Weerd, Brinkkemper, Souer, et al. (2006). The domain represents a context where the creation and application of situational methods is very relevant due to relatively low maturity of the field, but where knowledge regarding effective method implementations is scarce.

In addition to the earlier described focus on SPM , small to medium enterprises play an important role within this research. Often facing a low budget for process improvement, these companies have no access to expensive consultants and trainers. Furthermore, their size often allows a large amount of flexibility, and many small to medium enterprises are willing to share detailed information.

In order to further study the relevant concepts, we performed seven exploratory case studies. During these case studies, we tried to determine the current approach to process improvement in terms of process maturity, improvement drivers, and improvement steps (or increments). The lessons learned during these case studies have been used to further define the solution space of the OME. In this section, we outline the main lessons learned from these case studies, and how they affect the OME design.

6.3.1 *Case Study Organizations*

The case study organizations were carefully selected from a large network of Dutch product software organizations. We have selected organizations that indicated recent process improvement activity related to SPM and/or Scrum. All of the case studies within this research are small-to-medium enterprises. A brief summary of each organization is given below. For the sake of confidentiality, all names have been changed.

CHATCOMP is a privately-held company, headquartered in Amsterdam. It has local offices in London and San Francisco. The company created an independent, web browser-based instant messaging service in 2003. Within three years, the company grew from 20 employees to over 100 employees. This was accompanied by several extensive changes to the development and product management processes. ChatComp develops two product lines; instant messaging, and real-time messaging for smartphones.

FACILITYCOMP is an international software vendor that produces facility management and real estate management software for organizations (Integrated Workplace Management Systems). Founded in 1984, it currently has a customer base of over 1300, which is supported by more than 325 employees. The company's products are marketed through multiple, international subsidiaries, and a worldwide network of partners. For this case, not maturity profile was created.

SOCIALCOMP is a large social networking site in the Netherlands, focusing mainly on Dutch visitors and members. It was founded in 2004. In 2005, the site reached one million members and eighty million page views from the Netherlands per month. Nowadays, SocialComp has over eleven million

user accounts and serves over 5,8 billion page views per month, despite the advent of social networks such as Twitter and Facebook in the Netherlands.

TIMECOMP is a small independent Dutch software company, founded in 1992. The organization provides qualitative software applications and accompanying services to fulfil the need of achieving a higher efficiency from the utilization of human resources. TimeComp currently has two software products in their portfolio: one solution for time resource management, and one solution for printing and copying facilities for organizations. Currently, the company has 25 employees. For this case, only a partial maturity profile was created.

AGRICOMP was established in 1997 in Wageningen, the Netherlands. Initially, the organization focused on plant breeding companies, but they soon expanded their sights to include the entire range of companies within the agribusiness. Until 2003, AgriComp was purely a tailor-made software development company. Currently, it has evolved into a semi-product software company that has two products in its portfolio. Development effort is shared between tailor-made software and the two software products.

MAILCOMP is part of the global transportation and distribution industry, and dedicated to providing delivery solutions to its customers. The main organization consists of three pillars, known as 'mail', 'packages' and 'e-commerce'. The business unit that was studied in this research is active within the mail division. Due to the nature of its products, MailComp is originally a production-oriented company, with a focus on processing batches of transaction mail. Due to the rise of modern technologies, it is becoming significantly harder for MailComp to maintain its position in the market. Developing and offering software product in addition to their core business is supposed to be the solution for this problem, leading to significant process improvement requirements.

SERVICECOMP was originally founded in 2001 in the Hague, the Netherlands and produces service management products as well as credit management solutions. ServiceComp has years of experience in the development and implementation of IT-solutions for companies in the service-sector. The organization develops standardized, modular web-applications, which are fully customized to the processes, management, users and customers of their clients

6.3.2 *Case Study Findings and Effects on the OME Design*

Each case study has resulted in a detailed description of the process improvement effort, consisting of a description of the initial process, the improvement steps, the resulting process, and the changes in maturity and context. Case studies are most effective when they have a clear focus. Therefore, we have decided to focus on improvements related to (part of) the SPM activities. In addition, we have analyzed the introduction of the Scrum development process, which affects the SPM processes strongly.

We have summarized the results of these case studies in Table 6.1. For each case, we have identified whether the observed improvements were executed in the light of a process improvement framework—such as Capability Maturity Model Integration (CMMI) (CMMI Product Team, 2002)—, what the theme of the improvement was, and whether it was performed in an incremental fashion, or at once. Based on this summary in combination with the more detailed results, we can define several high-level findings that influence the solution space of the OME. These findings have been formulated as generic lessons. However, we are aware that generalization is not possible without further research.

Table 6.1.: Summary of case study improvement efforts

ORGANIZATION	Process Improvement Framework	IMPROVEMENT	STYLE
ChatComp	None	Introduction of Scrum	At once
		Improved Portfolio Management	Incremental
FacilityComp	None	Introduction of Scrum for SPM	Incremental
SocialComp	None, but part of a larger professionalization effort	Introduction of Scrum	At Once
		Professionalization of SPM	Incremental
TimeComp	None	Introduction of Scrum	At Once
		Improved Requirements Engineering	Incremental
AgriComp	None	Professionalization of SPM	Incremental
		Productization	Incremental
MailComp	None	Professionalization of SPM	Incremental
		Productization	Incremental
ServiceComp	None, but part of a larger productization effort	Productization	Incremental

6.3.2.1 *Situational, Experience-Based Process Improvement*

The first lesson that we can learn from these results is that none of the studied organizations have performed the identified process improvements in the context of a process improvement framework. Although several frameworks are available that allow a structured and evidence-based approach to selecting and implementing process improvements, they were either not deemed appropriate to the needs of the organization or too heavy to use, or the persons responsible for the process improvement efforts were not aware of their existence or not trained to use them effectively.

Lesson 1: Process improvement frameworks are often deemed inappropriate due to their size or due to their capability requirements.

An important requirement during the design of the OME is therefore that the approach needs to be flexible; simple and quick when organizations want to obtain a quick overview, but thorough and detailed when more resources are available. The organizations that we studied did not use any framework to guide their process improvement efforts, due to various reasons. Although this can be seen as something positive (many frameworks place a heavy burden on the organization), this also implies that the organizations lack guidance and do not have adequate means to measure whether they have reached their goal. The ad-hoc, trial-and-error approach to process improvement indicates that the organizations are not able to effectively leverage experience and knowledge from similar organizations.

The manner in which the organizational processes were changed can be classified as pragmatic. In most cases, a specific problem was identified or experienced. This problem could for instance be an unreliable requirements management process or an inability to cope with the day to day issues of product management. A significant and growing body of methods specific to SPM problems has been proposed during recent years. However, for the selected SPM processes, none of the responsible persons stated that they relied on scientific literature. In most cases, they were unaware of its existence, or they found the literature hard to put into practice.

Lesson 2: Scientific literature is rarely used directly by practitioners, both due to the fact that it is hard to access and that its language is often complex.

By providing an experience based, situational maturity model and assessment method, we provide the means to guide process improvement efforts effectively. The use of a method base that consists of both experiences from practitioners as

well as methods taken from scientific literature increases the availability and accessibility of relevant knowledge drastically. Methods can be modeled in the form of process-deliverable diagrams using MetaEdit+ (Tolvanen and Rossi, 2003). Any meta-information can be added through the method administration module of the OME. In this manner, a method base is built that can be searched for relevant MFs.

The MFs in the database are to be described by several attributes. This includes the implemented capabilities, the restrictions on situational factors, but also the feedback from people that have used the MFs before. A simple example of a MF with its describing attributes can be found in Table 6.2. It is based on Wieger's prioritization technique (Wiegers, 1999).

What is important in this context is the fact that process owners can vary in the rigidity that they demand from the ME process. Some wish only a partial improvement for a specific area, while others wish to improve their process to the maximum maturity level suggested for them. As stated before, evolutionary improvement is in many cases more prone to success than revolutionary change. This implies that it should be possible to provide improvements in the form of a roadmap when the process is changed rigorously.

Table 6.2.: Example method fragment descriptors for Wieger's prioritization technique

WIEGERS' PRIORITIZATION MATRIX		
CAPABILITIES	SITUATION	RATING
Internal Stakeholder Involvement	# of requirements < 50	Ease of use: 8/10
Prioritization Methodology	Partner involvement >= medium	Satisfaction: 6.5/10
Customer Involvement		Additional classifiers
Cost Revenue Consideration		
Partner Involvement		

During the initial phase, it is the process owner that needs to decide what the extent of the analysis will be, i. e. the focus domain. In many cases, it is not required to analyze the entire process. Instead, only a specific part of the process is looked at, and only for that part improvements are provided. For companies that

do require a major improvement of their process, this should be a possibility. In those cases, the entire process should be analyzed. This group also contains (new) companies that wish to obtain advice without having a process in place yet, or with a process that is to be abandoned altogether. Although the latter will only very rarely happen, it should be taken into account.

The interviews also suggest that there is a variety of wishes regarding the amount of effort that companies are willing to put in, in order to obtain process improvement advice. In the optimal case, companies are willing to provide complete information regarding their current process, deliverables, and situational factors that describe their environment. This means that their entire process needs to be captured in a way suitable for further elaboration. Also, the situational factors need to be captured in some way, either through a questionnaire or by means of an interview. With all data available, the process improvement advice that can be obtained is the most effective. However, capturing the entire process requires significant work from an expert who is able to employ an appropriate modeling technique.

In many cases, capturing full process information requires too much effort. Therefore, it should be possible to provide a process improvement advice based solely on the situational factors and maturity information. This option implies that the advice does not contain any information on how to implement the advice, but only what should be implemented. If a process owner is willing to provide full information regarding (part of) their process, the process should be modeled by an expert, either internal or external. The resulting model should contain detailed information regarding both the process as well as the deliverables. Therefore, Process-Deliverable Diagrams (PDDs) are a very suitable technique for this purpose. Vlaanderen (2010b) show how PDDs can be used to model an SPM process and to capture the current maturity level of a company's product management process. Based on the choices regarding scope and effort, a questionnaire is generated and performed to gather information regarding the situational context. In the area of SPM , the situational analysis can be performed by conducting a questionnaire with a list of all the relevant situational factors as described by Bekkers, van de Weerd, Brinkkemper, et al. (2008).

Lesson 3: Most organizations have a low SPM maturity, indicating a lack of activities that are deemed important by many industry experts.

In addition, we observed that most organizations scored low on our SPM maturity matrix (Bekkers, van de Weerd, Spruit, et al., 2010), mainly with respect

to product planning and portfolio management. We have summarized the results of several maturity assessments in Table 6.3. None of the case companies has used the maturity matrix actively during their process improvement efforts, but the matrix proposes processes that are deemed essential by many industry experts.

Table 6.3.: Maturity matrices summary

CAPABILITY PROCESS	STAGE 1	STAGE 2	STAGE 3	STAGE 4	STAGE 5
Requirements management					
Requirements gathering	A B C D E F	A B C D E F	A B C D E F	A B C D E F	A B C D E F
Requirements identification	A B C D	A B C D	A B C D	A B C D	A B C D
Requirements organizing	A B C	A B C	A B C	A B C	A B C
Release planning					
Requirements prioritization	A B C D E	A B C D E	A B C D E	A B C D E	A B C D E
Release definition	A B C D E	A B C D E	A B C D E	A B C D E	A B C D E
Release definition validation	A B C	A B C	A B C	A B C	A B C
Scope change management	A B C D	A B C D	A B C D	A B C D	A B C D
Build validation	A B C	A B C	A B C	A B C	A B C
Launch preparation	A B C D E F	A B C D E F	A B C D E F	A B C D E F	A B C D E F
Product Planning					
Roadmap intelligence	A B C D E	A B C D E	A B C D E	A B C D E	A B C D E
Coare asset roadmapping	A B C D	A B C D	A B C D	A B C D	A B C D
Product roadmapping	A B C D E	A B C D E	A B C D E	A B C D E	A B C D E
Portfolio management					
Market analysis	A B C D E	A B C D E	A B C D E	A B C D E	A B C D E
Partnering & contracting	A B C D E	A B C D E	A B C D E	A B C D E	A B C D E
Product lifecycle management	A B C D E	A B C D E	A B C D E	A B C D E	A B C D E
Total implemented capabilties	2 (3%)	7 (10%)	20 (29%)	30 (44%)	38 (56%)

6.3.3 *Process Description Generation*

One of the requirements of the SAM is that processes are well-embedded within the organization, and that they are well described in order to avoid loss of know-how when a key stakeholder leaves the organization, and to allow effective knowledge transfer internally. During the case studies, we have found that process descriptions were often not available, and when they were, the descriptions were incomplete or outdated. Some organizations, such as SocialComp explicitly chose not to document their processes for fear of an overly strong focus on formalities.

However, especially in fast growing companies such as SocialComp, this can pose a problem in the future.

Lesson 4: Organizational processes are often not documented or they are outdated; either by choice or by lack of resources.

An important function of the OME is therefore the production of relevant method descriptions that help the organization in implementing the suggested improvements and serve as an effective tool for internal communication. The OME should generate complete method descriptions with explanations of all steps, deliverables and roles. These descriptions aid the process owner during the implementation of the process in the company.

The possibilities for embedding or implementing the process advice vary depending on the amount of information that a company has provided. Templates and tools can be generated and/or updated based on the original and the new method description (expressed in the form of PDDs). The possibilities are limited when only maturity information is known, but considerably increase when full method information is given.

If the company's original work documents are available, then they can be updated to reflect the new deliverables within the method. During this step, original data should be maintained while new columns, sections, formulas, etc. are added to the documents. In case deliverables are not available, templates can be generated based on the generated process description. The generated templates should be directly usable within the new process.

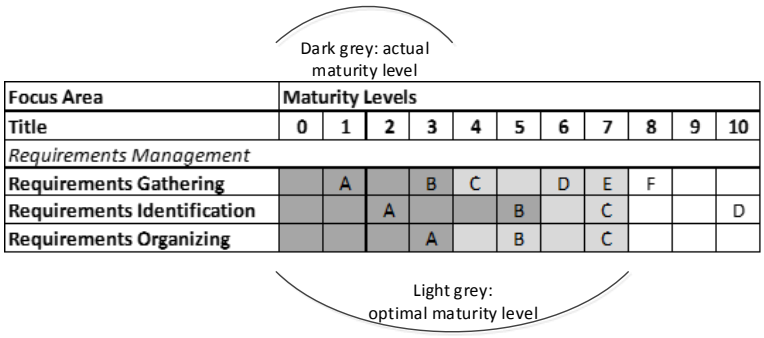
6.3.4 *Improvement Roadmapping*

As can be seen in Table 6.1, improvement of the SPM processes was performed in an incremental fashion in all cases. This fits with theories from the field of ME related to MFs and method increments. The fact that many improvements are implemented step by step can be leveraged and process improvement research should provide fitting solutions, and supporting such steps is one of the main objectives of the OME.

Lesson 5: Process improvements are often implemented incrementally, without a long term process improvement planning.

The first step in the process improvement activity is obtaining an overview of the current situation in terms of implemented capabilities, and situational factors

of the business (unit). The current situation constitutes both the currently employed process as well as more generic aspects of the company at hand. The next phase takes the situational factors and the list of implemented capabilities from the first phase as input, after which it determines how the current process could be improved. In the domain of SPM, this phase has already been described by Bekkers, Spruit, van de Weerd, et al. (2010) in the form of the SAM, but it will be summarized here for the sake of completeness. The need analysis consists of three activities; (1) construction of the current capability profile, (2) calculation of the optimal capability profile, and (3) calculation of an 'areas of improvement' matrix.



Focus Area	Maturity Levels										
Title	0	1	2	3	4	5	6	7	8	9	10
Requirements Management											
Requirements Gathering		A		B	C		D	E	F		
Requirements Identification			A			B		C			D
Requirements Organizing				A		B		C			

Figure 6.5.: Improvement areas in the maturity catrix

The first of these three consists of translating the results from the initial maturity assessment into a form usable for further calculation. The second activity is somewhat more complex. The optimal capability profile is determined by a set of situational factor effects. Several situational indicators have an associated effect. By applying all applicable situational factors effects, an optimal capability profile is obtained that is customized for the current company. The current capability profile and the optimal capability profile are then combined into an Areas of Improvement matrix. This is again a capability matrix, with both previous matrices integrated into it. Between the two matrices, a gap can exist, which we will call the delta. This delta indicates the capabilities that need to be implemented, in order to arrive at the optimal maturity level. An example of such an Areas of Improvement matrix within the SPM domain is shown in Figure 6.5. The actual delta is the light-grey area, as this depicts the difference between the actual and the optimal

maturity level. This set forms the basis for the next phase in the process of method improvement.

Based on the results of the assessment, new MFs can be selected from the method base. These MFs can be integrated into the current method, and in case of large improvements, an improvement roadmap can be generated. Steps are needed since solutions will in many cases be too large for implementation in one iteration. An evolutionary approach has more chance of success as it will likely yield a higher acceptance due to smaller, incremental changes.

Although a long term improvement plan was never created in advance, some cases did show a certain theme to which the identified process improvements could be attached. A strong example is that of ServiceComp, which showed a strong productization drive. Its development is shown in Figure 6.6, which displays the main events and process changes. For each of the stages, we were able to construct a maturity matrix that showed the capabilities relevant to that stage. With a sufficiently filled method base, such experience can be used to create valid improvement roadmaps.

6.3.5 *Centralized Knowledge Repository*

Another interesting observation is the fact that all companies were very forthcoming in providing us with detailed information regarding their current SPM processes, changes and problems that were experienced in the past, and changes that they plan to make in the future. There is a strong case to make for the possibility that this behavior is specific to small and medium size organization, and that larger organizations are less forthcoming due to stronger internal legislation and a more sensitive competitive position. Still, the fact that the organizations at hand were willing to provide such detailed information enables interesting possibilities in the light of the OME.

Lesson 6: Many organizations are willing to share detailed information regarding internal processes for the sake of scientific progress and the advancement of the SPM field.

This observation directly influences our choice to primarily design the OME as a public platform. By not restricting the environment to individual organizations, it allows benchmarking and knowledge dissemination that leverages the knowledge of a broad range of practitioners. Based on what we have heard during the interviews, we believe that the advantages of increased knowledge availability for

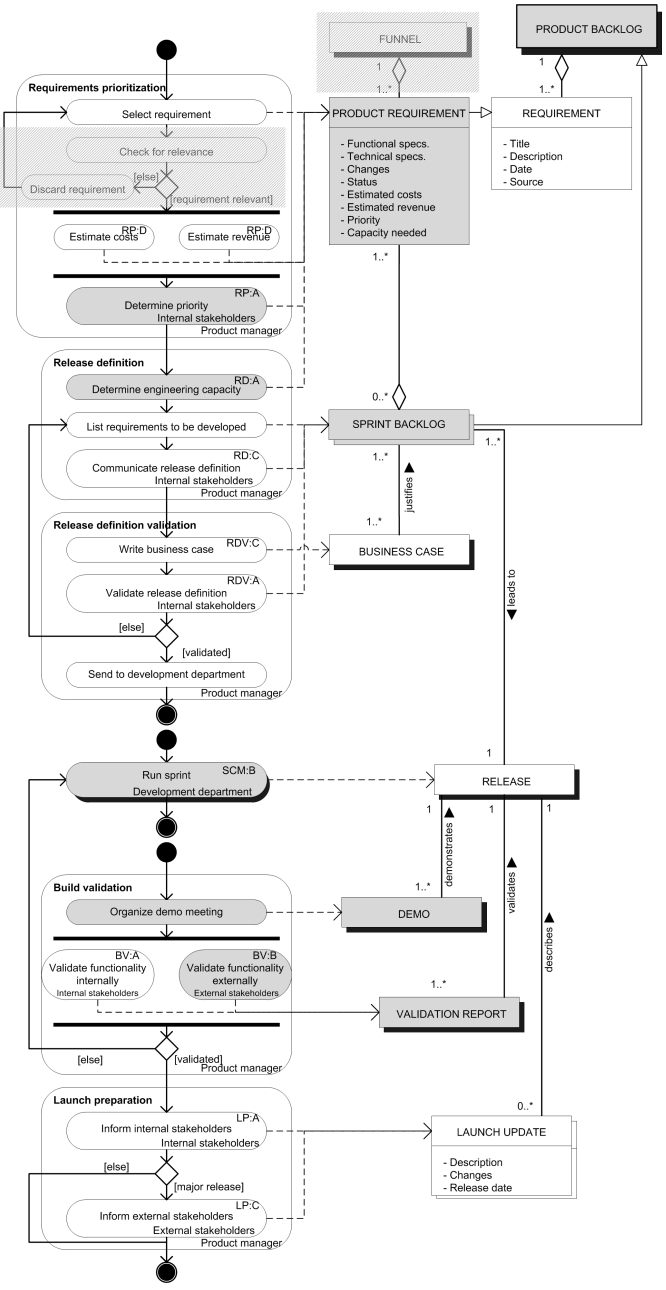


Figure 6.6.: Productization effort at ServiceComp

SPM outweigh privacy and competitive issues. It does however require a strong focus on data protection and fair use.

To enhance reliability of the organizational data, the employed questionnaires can be replaced by performing an interview. In addition, if a process owner is willing to provide full information regarding (part of) the process, the process should be modeled by an expert, either internal or external. In addition, as the number of available methods and techniques in a given area starts to increase, the difficulty of determining the optimal solution for a given problem becomes more complex. This generally increases the demand on the knowledge of the person or group of persons performing the assessment, or on the quality of the process improvement method.

Due to our focus on small to medium enterprises, we design the OME with the idea that process owners are able to use the system themselves. As the system is highly dependent on an extensive method base, partly created and enhanced by its users, knowledge demands on the user are low. However, as process improvement efforts can be time-consuming, we also foresee the use of the OME by process improvement experts (i. e. consultants, ME experts, etc.).

6.3.6 *People-Oriented Design*

Although processes, capabilities and situational factors form a very solid ground for MF selection, we need to take into account that we are dealing with processes in which humans are involved. This means that the resulting process needs to fit with the preferences of the people involved in it. These people need to be able to express these preferences during the selection of alternative MFs. The results from interviews have indicated that product managers are not always willing to accept suggestions made to them by a machine.

Lesson 7: Process owners are reluctant to trust pure machine-based process improvement suggestions.

Therefore, the approach should allow for differences in the amount of freedom that is provided. Users should be at liberty to select MFs that differ from the ones proposed. This ‘freedom-of-choice’ has serious consequences for the OME. In order to make the freedom given to users useful, they need to be provided with a sufficient amount of information for them to base their decision on.

Since every MF can be displayed in the form of a PDD, users can use this diagram to form an initial mental image of its implications. All related activities and

deliverables are readily available in the method fragment. However, in addition to this, we also identified a need for more sources in the form of experience reports. Experience from people in similar situations is highly valued, and would thus be a valuable addition to the process. Based on all of the sources of information combined, users should be able to make a valid and well-argued choice regarding the MFs that should be selected, and thus regarding the changes that should be made to the existing process.

6.4 DESIGN IMPACT

In this paper, we have described the results from seven case studies that were performed with the aim of exploring the concept of incremental process improvement. From the case studies, we were able to extract several interesting findings that are relevant for the design and development of an online environment that can be used to assess an organization's current processes, create an advice based on this assessment, and align the company's tooling infrastructure with the method improvements.

These findings have been used to further define the solution space of the OME. Table 6.4 provides an overview of the major design choices and the observations that they are based on. The areas influenced by these design choices are shown in Figure 6.7.

6.5 DISCUSSION AND FURTHER RESEARCH

The knowledge infrastructure that we have described during the discussion of our research approach has evolved into the OME as presented in this chapter. The advances in the areas of situational ME and incremental process improvement enable us to focus on the analysis and design of the system (see Figure 6.1). The findings from the case studies described above have been used to further refine the solution space for the OME.

From this point onwards, each of the areas of the OME needs to be addressed in detail, putting together the architecture and the resulting system. Expertise in several areas will be needed, as each part of the OME has its specific challenges, from linguistic analysis for method assembly to data-optimization for the method base.

Table 6.4.: Linking observations to OME design choices

DC	OBSERVATION	RESULTING DESIGN CHOICE
1	Process improvement frameworks are often deemed inappropriate due to their size or due to their capability requirements.	Non-prescriptive maturity model with a focus on situationality.
2	Scientific literature is rarely used directly by practitioners, both due to the fact that it is hard to access and that its language is often complex.	Mixed-model method base consisting of experience data and formal MFs.
3	Most organizations score low on the SPM maturity matrix, indicating the lack of activities that are deemed important by many industry experts.	Guidance of the process improvement effort by an industry-proven SPM maturity matrix.
4	Organizational processes are often not documented or outdated; either by choice or by lack of resources.	Meta-model based process description generation, including activity/deliverable definitions, experience reports and PDDs.
5	Process improvements are often implemented incrementally, often without a long term process improvement planning.	Combination of Situational Assessment Method and an incremental ME approach employing method increments and increment roadmaps.
6	Many organizations are willing to share detailed information regarding internal processes for the sake of scientific progress and the advancement of the SPM field.	Open access KM environment, enabling benchmarking and effective knowledge dissemination.
7	Process owners are reluctant to trust pure machine-based process improvement suggestions.	Experience-based process improvement using constant data quality improvement and industry content.

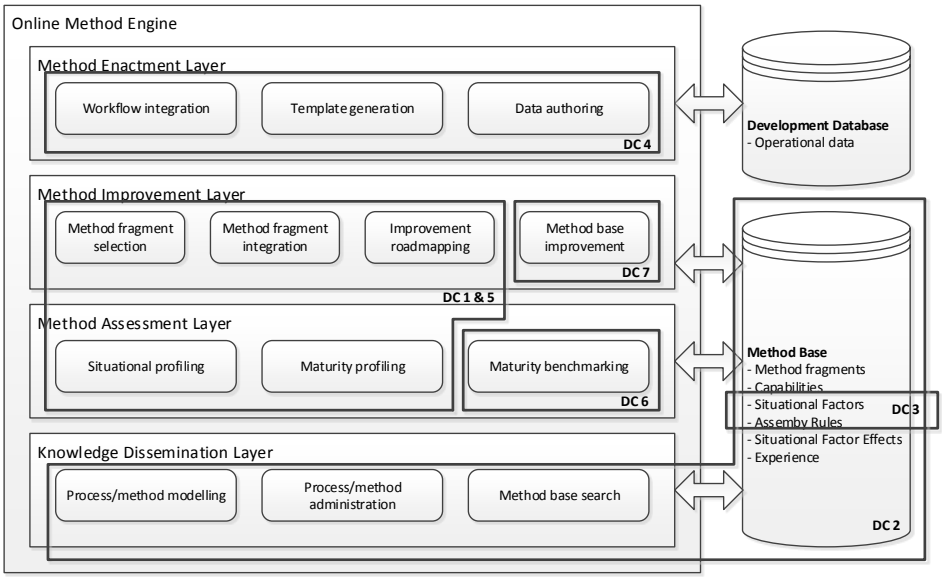


Figure 6.7.: Focus areas of the major design choices

Up until this stage, the research effort has mainly been focused on analyzing the current situation and the need, with a focus on the SPM domain. Furthermore, much work has been performed on the underlying meta-modeling techniques that are used throughout the system. This leaves the remaining areas of process alternative selection, improvement roadmap creation, and increment selection and implementation open for future research. An example of a topic that is currently being researched is the creation of an ontology for SPM that should support method description, analysis and construction. Other open issues are the construction of a technique for aligning organizational tooling to method changes and the classification of MFs.

An important factor that can never be left out during the elaboration is the fact that the purpose of the OME is the improvement of processes. As a consequence, we are always dealing with people that bring habits, experiences, and opinions. This should not be overlooked. Doing so would result in a system that is too rigid, forcing people into ways of working that they will not accept, thereby foregoing the purpose of the system. Unfortunately, the OME that is presented in this paper has not been fully validated yet. As development continues, the user should not be forgotten. At several points in time, the prototype should be validated with practitioners, and corrections should be made according to the results. When done

correctly, this will result in a functional online method engineering environment that aids practitioners with the improvement of their SPM processes. We believe that this solution can increase the maturity of the software industry significantly by providing professionals with the right tools to optimize their processes.

7

IMPROVING SOFTWARE PRODUCT MANAGEMENT: A KNOWLEDGE MANAGEMENT APPROACH

The skills and practices of a product manager can be seen as part of the intellectual capital of a software vendor. From this perspective, it is surprising that knowledge regarding Software Product Management (SPM) is not yet widely available. This impedes many companies from effectively improving their SPM methods. To support knowledge dissemination, method assessment, and method improvement in this and other areas, we propose the Online Method Engine (OME). The solution combines Knowledge Management (KM) principles with incremental process improvement in order to allow software companies to obtain improvement guidance for their SPM methods, specific to their situation. The approach is designed to be domain-independent.

This work was originally published as:

Vlaanderen, K., van de Weerd, I., and Brinkkemper, S. (2013). "Improving Software Product Management: a Knowledge Management Approach." In: *International Journal of Business Information Systems (IJBIS)* 12.1, pp. 3–22

7.1 INTRODUCTION

For each profession, be it manual labor or knowledge work, a certain amount of knowledge and skills is required before results become satisfactory. Education, tutorship, and community based knowledge sharing are all paths to spread this knowledge and these skills to the individuals that require them. Consultants provide the same for organizations. Problems arise when these paths are either absent or not yet well-developed.

An example of where this happens is the area of Software Product Management (SPM). SPM is the process of managing requirements, defining releases, and defining products in a context where many internal and external stakeholders are involved (van de Weerd, Brinkkemper, Nieuwenhuis, et al., 2006b). Due to the complexity of software products, with a large variety of stakeholders, long lists of requirements and a rapidly changing environment, SPM is a complex task and has a huge impact on the success of a software product (Ebert, 2007). However, relatively little scientific work has been performed in this area and practical experience is not yet widespread.

Within the area of Knowledge Management (KM), it is our prime objective to support the creation, development, organization and leverage of knowledge (Wiig, 1997). In the case of SPM , the skills and practices of a product manager can be seen as part of the intellectual capital of a software vendor. However, a problem with this intellectual capital is, as Rus and Lindvall (2002, p. 26) put it, "it has legs and walks home every day". Furthermore, knowledge plays an important role during process improvement efforts. Product managers need to be supported while getting acquainted with their field and improving their current practices. We believe this problem can be solved by implementing the right KM tool. This belief is shared by García, Amescua, Sánchez, et al. (2011), who tackle the problem by developing a software process knowledge repository.

In this paper, we propose a KM system, called the Online Method Engine (OME). The four chief aims of the OME are knowledge dissemination, method assessment, method improvement, and method enactment. We explain why there is a need for such a system, and what our proposed solution is. With the OME, we aim to relieve two of the five major software process improvement risks as defined by Niazi (2011); lack of support and lack of defined Software Process Improvement (SPI) implementation methodology. The solution described in this paper is a continuation of earlier work (as described in Chapter 6).

Section 7.2 addresses several of the related research fields, including Situational Method Engineering (SME), SPM, and software process improvement. In Section 7.3, we give an outline of the proposed solution, followed by three sections that describe the aforementioned problem areas that our solution addresses. In these sections, we will further explain the core concepts with a running example. The final section contains our conclusions and an outline of further research.

7.2 RELATED LITERATURE

7.2.1 *Knowledge Management Systems*

The area of KM has received many publications during the last few decades, many of which take a different approach. Bjornson and Dingsøyr (2008) have analyzed KM literature in the context of Software Engineering (SE). They categorize the literature according to the Earl's schools of KM (Earl, 2001)(see Table 7.1). From our perspective, KM is "a method that simplifies the process of sharing, distributing, creating, capturing and understanding of a company's knowledge" (Davenport and Prusak, 1998).

	TECHNOCRATIC		
	SYSTEMS	CARTOGRAPHIC	ENGINEERING
Focus	Technology	Maps	Processes
Aim	Knowledge bases	Knowledge directories	Knowledge flows
Unit	Domain	Enterprise	Activity

	ECONOMIC	BEHAVIOURAL		
	COMMERCIAL	ORGANIZATIONAL	SPATIAL	STRATEGIC
Focus	Income	Networks	Space	Mindset
Aim	Knowledge assets	Knowledge pooling	Knowledge exchange	Knowledge capabilities
Unit	Know-how	Communities	Place	Business

Table 7.1.: Earl's schools of knowledge management

KM as a whole has several aspects, such as socio-cultural, organizational, and technical. In this paper, we focus mainly on the technical aspect. Lindvall, Rus, and Sinha (2003) defined an architecture model of KM systems. The solution presented in this paper focuses strongly on the development of a knowledge portal and repository with support for competence management and knowledge discovery.

KM and SPI are potentially very close neighbours, due to the knowledge sharing aspect that is inherent in both fields. Recent years have seen some publications from this perspective, including the application of KM for SPI in very small entities (Bin Basri and O'Connor, 2010; Bin Basri and O'Connor, 2011), and the identification of success factors for SPI through experience management by Sharma, Singh, and Goyal (2010).

Through a KM system, we aim to share method knowledge in such a way that organizations are able to assess and improve their current methods. Central terms in this context are *capability* and *maturity*, both applied in an earlier study by Baskerville and Pries-Heje (1999). From this line of research, we can follow the link to the field of SME.

7.2.2 *Method Engineering*

The term 'methodology engineering' was introduced in 1992 by Kumar and Welke, referring to the engineering of information systems development methods, taking into account the uniqueness of a project situation. In his work, Brinkkemper (1996, p. 276) introduces the related term 'method engineering', referring to "the engineering discipline to design, construct and adapt methods, techniques and tools for the development of information systems". With the introduction of this term, he also makes a clear distinction between a method and methodology, where a method is "an approach to perform a systems development project, based on a specific way of thinking, consisting of directions and rules, structured in a systematic way in development activities with corresponding development products" (Brinkkemper, 1996, pp. 275–276) and a methodology is a system of methods in a particular discipline.

To be able to better address the problem that "there is no detailed information systems methodology which is the best in all situations" (Kumar and Welke, 1992), a special type of Method Engineering (ME) has been introduced; Situational Method Engineering. Harmsen, Brinkkemper, and Oei (1994) define a

situational method as "an information systems development method tuned to the situation of the project at hand". Within the field of SME, several different modularization constructs exist (Henderson-Sellers and Gonzalez-Perez, 2008).

In this research, we employ Method Fragments (MFs), which are defined as "... a description of an Information Systems (IS) engineering method, or any coherent part thereof" (Harmsen, 1997, p. 40). Each MF can be classified on three orthogonal dimensions: perspective, abstraction level, and layer of granularity. A MF consists of a process part and a product part. They are described in the form of process-deliverable diagrams. These diagrams describe the relationship between the process on the one hand and the products on the other.

To bridge the gap with the software process improvement literature, part of the ME research community has during the last five years focused on supporting evolutionary process improvement through incremental method evolution. During an exploratory case study, van de Weerd, Brinkkemper, and Versendaal (2007) defined the atomic types of adaptations that can be made to an existing method, called *method increments*. Insight into these method increments enables focused improvement of software development methods based on localized issues and improvement possibilities. In theory, it allows the linkage of locally identified problems to specific changes to the current method. Rossi, Ramesh, Lyytinen, et al. (2004) described the use of a method rationale to support such evolutionary method development with a non-formal element. A similar approach is described by van de Weerd, Brinkkemper, and Versendaal (2010) in the form of method increment drivers.

To support the ME activity, several Computer/Aided Method Engineering (CAME) tools have been developed. Most of these focus on the meta-modeling aspect. One well-known example of such a tool is MetaEdit+ (Tolvanen and Rossi, 2003), which enables the definition and usage of domain-specific languages. This tool was also applied in a more agile context (Berki, 2003). It forms one of the key components of the OME approach.

7.2.3 *Software Product Management*

A large share of software producing companies began with the development of custom software commissioned by customers. This way of working has evolved into the production of product software: products that are not developed for one specific customer, but for an entire market (Ebert, 2007; Xu and Brinkkemper,

2007). Mainly among small-to-medium enterprises, there are a lot of companies that focus on the development of product software. The change from custom software to product software requires significant adaptations in the management of business processes. Instead of dealing with one bidder and one delivery date, companies now have to cope with multiple stakeholders, and different releases and product configurations. In order to manage this effectively, implementing SPM processes in the organization is essential (Carmel and Becker, 1995; Cusumano, 2004).

The original area of product management can be defined as "the discipline and role, which governs a product (or solution or service) from its inception to the market/customer delivery in order to generate biggest possible value to the business" (Ebert, 2007, p. 1). SPMis then "the process of managing requirements, defining releases, and defining products in a context where many internal and external stakeholders are involved" (Gorchels, 2000; van de Weerd, Brinkkemper, Nieuwenhuis, et al., 2006b). Due to the complexity of software products, with a large variety of stakeholders, long lists of requirements and a rapidly changing environment, SPM is a complex task. However, relatively little scientific work has been performed in this area. An attempt to close this gap has been provided by van de Weerd, Brinkkemper, Souer, et al. (2006) in the form of a competence model for SPM. Their work aims at providing a structure for the body of knowledge regarding SPM by identifying and defining the key process areas as well as the internal and external stakeholders, and their relations.

Some recent work related to the specific areas of the framework include the definition of key success factors for pricing software products (Kittlaus and Clough, 2009) and the design of the Quality Performance (QUPER) model, developed for handling non-functional requirements (Berntsson Svensson, Olsson, and Regnell, 2008; Regnell, Berntsson Svensson, and Olsson, 2008). In the fields of life-cycle management and release planning, important work has been performed related to market-driven requirements engineering (Carlshamre and Regnell, 2000) and requirements interdependencies (Carlshamre, Sandahl, Lindvall, et al., 2001). A systematic review of release planning approaches has been given by Svahnberg, Gorschek, Feldt, et al. (2010). Attempts to link long-term product planning and agile development are provided by Vähäniitty and Rautiainen (2008) and Vlaanderen, Brinkkemper, Jansen, et al. (2011). Greer and Ruhe (2004) elaborate on agile release planning by providing an iterative optimization method. Collaboration between product managers and development teams in challenging environments,

such as where no complete requirements are available, is investigated by Fricker, Gorschek, Byman, et al. (2010).

7.3 ONLINE METHOD ENGINE

The idea of a KM system for software process improvement (SPI) in SPM originates from the fact that the domain is still not clearly defined. It is in need of structuring, so that its processes and products can be improved and optimized. A KM system aids in making available the resources that are now difficult or impossible to access. These resources can be methods, techniques, tools and templates.

Brinkkemper (1996) and van de Weerd, Versendaal, and Brinkkemper (2006) proposed initial versions of a KM system for SPM in the form of the Product Software Knowledge Infrastructure (Product Software Knowledge Infrastructure (PSKI)). They identified six main activities that are necessary to create a KM system for SPI in SPM. The 'analysis of current situation' entails an assessment of a company both in a general manner employing situational indicators, as well as focused on the SPM process by using capabilities. The 'analysis of need' phase takes the situational factors and the list of implemented capabilities from the first phase as input, after which it determines how the current process could be improved. During the 'selection of process alternatives', each missing capability has to be connected to a MF that implements the capability. This is followed by the 'embedding of process advice', including the generation of a process (implementation) description and the generation of templates. 'Method base administration' and 'knowledge base improvement' are related activities that are required to ensure the validity and completeness of the KM system.

The solution proposed in this paper is an extension and elaboration of the original proposal. This results in a firm basis for further research and the actual gradual implementation of the KM system. An overview of the KM system, now called 'OME', is shown in Figure 7.1. The OME contains four functional layers. Each layer contains several functional components, shown by the rounded boxes. The approach integrates techniques for sharing knowledge, and assessing, improving and implementing methods.

The bottom layer is related to knowledge gathering and sharing. In the current approach, methods can be *modelled* in the form of process-deliverable diagrams using MetaEdit+ (Tolvanen and Rossi, 2003). Any meta-information can be added

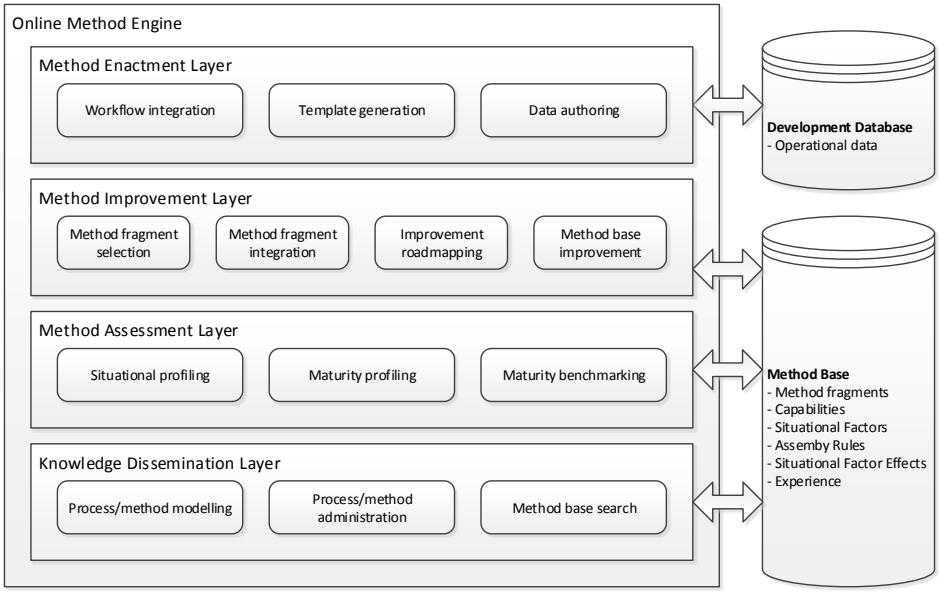


Figure 7.1.: Conceptual model of the Online method engine

through the *method administration* module. In this manner, a method base is built that can be *searched* for relevant MFs.

The method assessment layer deals with qualitative analysis of methods using *situational profiling* and *maturity profiling* as described in Section 7.5. These profiles can be used for direct *benchmarking* against other organizations within industry.

Based on the results of the assessment, new MFs can be *selected* from the method base. These MFs can be *integrated* into the current method, and in case of large improvements, an improvement roadmap can be generated (see Section 7.6).

On the top layer, functionality related to method execution is described. This includes *workflow integration* to synchronize method descriptions with actual process data, *template generation* to update existing tools such as spreadsheets and requirements databases according to the updated method, and *data authoring* to manage company specific process data. The method execution layer will not be further detailed in this paper.

The core concepts will be further explained in the following sections. In order to illustrate the possible application of the OME, we will apply the described concepts to a real-life case study at Planon International (from now on referred to

as Planon). Although the OME is not yet fully implemented, and has therefore not actually been used in the case, the example should provide a better understanding of the main ideas behind the OME. The case is presented as a running example in outlined boxes throughout the text.

> Planon is an international software vendor that produces Facility Management and Real Estate management software for organizations (Integrated Workplace Management Systems). Founded in 1984, it currently has a customer base of over 1300 corporate clients, which is supported by more than 325 employees. The company's products are marketed through six subsidiaries, based in the Netherlands, Belgium, Germany, UK, India and the US, and a worldwide network of partners. The company has, as one of the first known companies, implemented an agile SPM method based on the agile principles in general and the Scrum development method specifically.

7.4 KNOWLEDGE DISSEMINATION

The core function of the OME is knowledge dissemination or sharing. As knowledge regarding SPM methods is not yet widely available, effort is required to fill a method base with relevant techniques, tools and lessons. This method base serves to functions. First of all, it allows process owners to directly find methods that are relevant to them. More importantly though, the method base forms the backbone of any process improvement efforts performed through the OME.

On the right hand of Figure 1, two databases are shown; one for the ME related data such as MFs and Situational Factors (SFs), and one for the method execution related data such as requirements and planned releases. These databases are the knowledge repositories of the OME.

Method descriptions from scientific literature and from practice can be captured and stored in the OME, in order to allow a central knowledge repository for researchers and practitioners. Method knowledge is stored in the form of meta-models, situational profiles, and experience reports. In our approach, we have chosen process-deliverable diagrams (PDD's) along with activity/deliverable descriptions as the main meta-modelling technique.

To safeguard the quality of the knowledge within the OME, the system supports activities related to knowledge base improvement. These activities ensure valid

data and relevant advices based on regular user input. The OME supports user-generated content such as reviews and advice. This implies that, apart from the MFs themselves, the method base is filled with experiences. This is an essential part of the system, and users must be motivated to do so.

One type of user contribution is data corrections, applying especially to the situational factors describing the applicability of MFs. By gathering user data, this applicability can be further refined or even corrected. This has the potential to improve the reliability of the method base, and therefore of the ME approach, drastically.

While data corrections ensure that the quality of the method base is maintained, they do not provide any in depth information to the process owner that needs to decide whether or not a MF is suitable for his business unit. For this, a more direct solution is required. One proposed solution is the possibility of writing reviews for certain MFs. Such reviews can include more subtle remarks that are otherwise not expressible. These remarks can include 'soft' issues, interoperability with other fragments, and performance issues in certain situations.

When all attributes of the MFs that are stored in the method base are combined, the OME can make suggestions based on choices by similar companies. On a more abstract level, the method base contains rules that govern the selection and assembly of MFs, situational factors, capability descriptions and situational factors effects. These data are not constant and are subject to changes resulting from experience, industry changes, etc.

Until 2004, product development at Planon was based on the Prince2 method, after which a switch was made to Scrum, thus following the ideas of the Manifesto for Agile Software Development (Beck, Beedle, van Bennekum, et al., 2001). This means that working software is delivered frequently, changing requirements are welcomed (even late in development), and teams reflect regularly on how to become more effective. Other agile ideas are that the course of a project unfolds in time, decisions are being made in a decentralized team-based way and there is a focus on early feedback.

In the light of knowledge dissemination, the OME can provide clear descriptions of processes and deliverables relevant to Scrum. In addition, experience reports and lessons learned can be provided by the system. Such information helps a process owner in exploring the possibilities available to him.

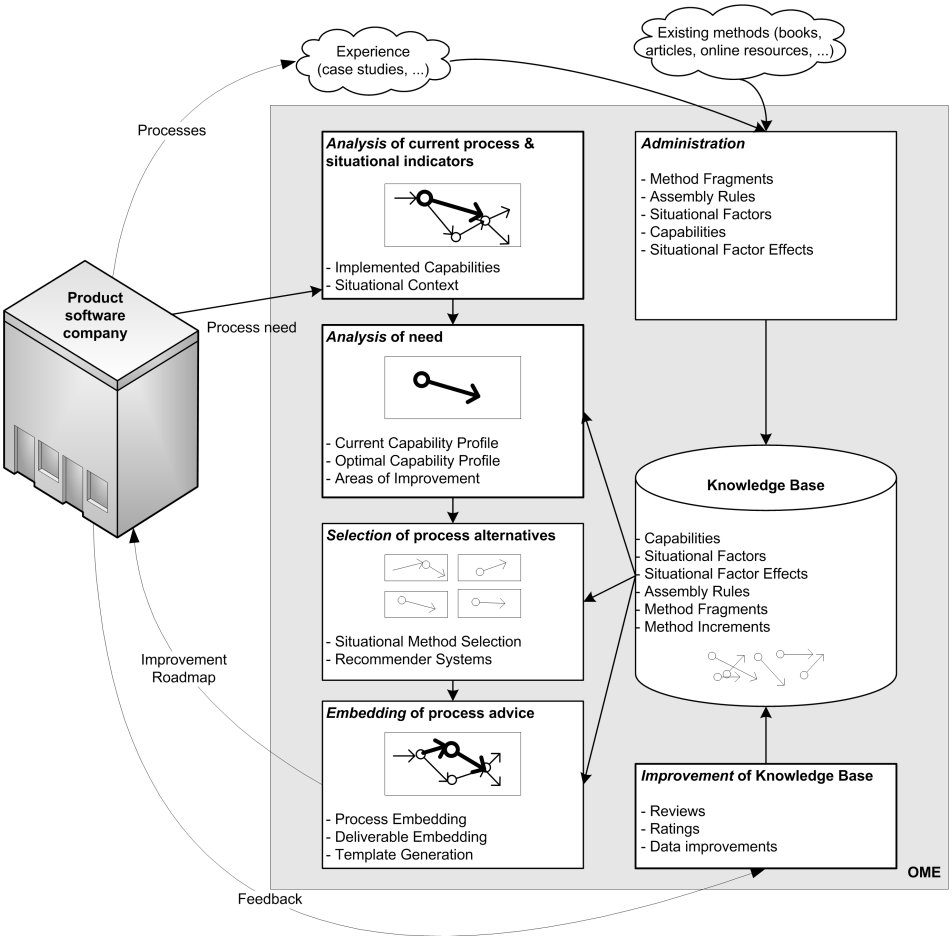


Figure 7.2.: Method improvement using the OME

7.5 SUPPORTING METHOD ASSESSMENT

During the last few years, several of the concepts related to method assessment and improvement in SPM have been expanded and elaborated. Bekkers, van de Weerd, Brinkkemper, et al. (2008) suggested that the general assessment of a company's SPM business function can be performed by filling in a list of situational factors. The assessment of the company's SPM methods can be performed by filling in a capability matrix (Bekkers, Spruit, van de Weerd, et al., 2010). Based on the results of these two assessments, method advice can then be given to the

company. The full process guiding method improvement efforts is depicted in Figure 7.2. In the next subsections, each of the areas shown in this figure will be explained in more detail.

7.5.1 *Analysis of Current Method & Situational Indicators*

The first step in the method improvement activity is obtaining an overview of the current situation in terms of implemented capabilities, situational factors of the business (unit), and in some cases the current method. This approach can be generalized into a form as depicted in Figure 7.3. The situational context can be obtained by conducting a questionnaire with a list of all the relevant situational factors, as described by Bekkers, Spruit, van de Weerd, et al. (2010). As before, the reliability of the data could be improved by replacing the questionnaire with an interview. During the initial phase, it is the company that needs to decide what the extent of the analysis will be, i. e. the focus domain. As the demand on the OME can vary per case, a certain amount of flexibility needs to be incorporated into the method.

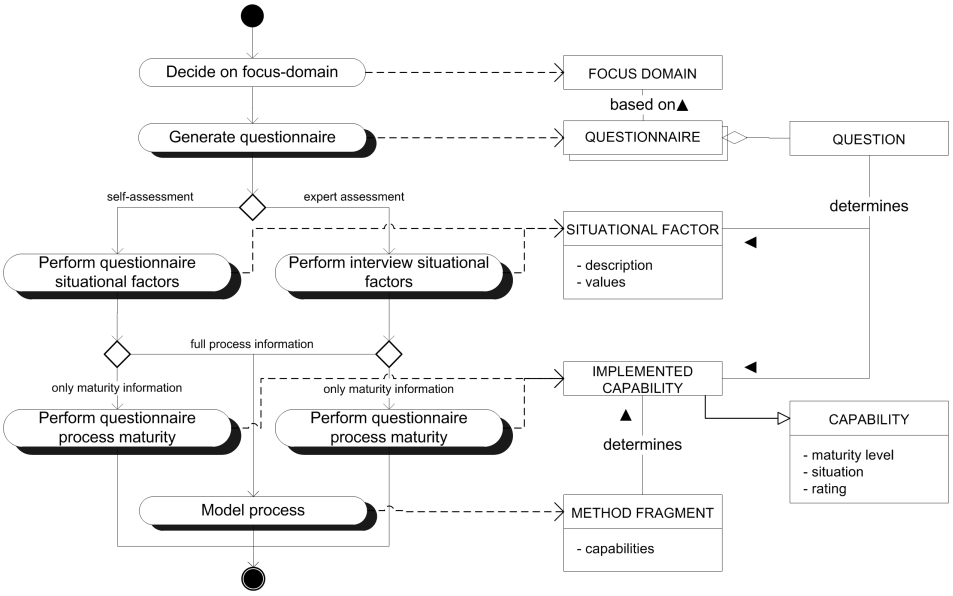


Figure 7.3.: Analysis of current situation

The input during a process improvement effort can vary on two dimension, as illustrated by Figure 7.4. These dimensions are scope and effort.

The scope can vary from the improvement of a single activity (quadrant A and B) to improvement of the entire spectrum of SPM related activities as described in the SPM Competence Model (Bekkers, van de Weerd, Spruit, et al., 2010)(quadrant C and D). Based on the selected scope, a questionnaire is generated and conducted to gather information regarding the situational context. In the area of SPM , the situational analysis has been performed by conducting a questionnaire with a list of all the relevant situational factors as described by Bekkers, Spruit, van de Weerd, et al. (2010). To enhance reliability of the data, the questionnaire could be replaced by performing an interview. Similar solutions can be developed for other areas. Secondly, interviews with various industry experts have indicated that there is a variety of wishes regarding the amount of effort that companies are willing to put in in order to obtain method improvement advice.

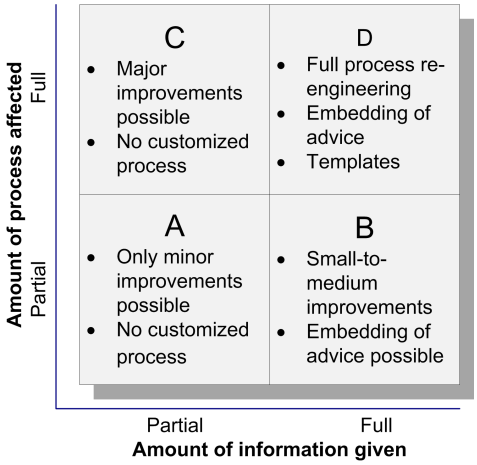


Figure 7.4.: Variations in the input of the OME

In the optimal case, companies are willing to provide complete information regarding their current methods and situational factors (quadrant B and D). The methods can then be modelled by an expert, either internal or external. The resulting model should contain detailed information regarding both the activities as well as the deliverables. Therefore, process-deliverable diagrams are a very suitable technique for this purpose. With all data available, the method improvement advice that can be obtained is the most effective.

In many cases, capturing full method information requires too much effort. Therefore, it should be possible to provide a method improvement advice based solely on the situational factors and maturity information (quadrant A and C). This option implies that the advice does not contain any information on how to implement the advice, but only on what should be implemented. The required information can be gathered through a questionnaire. To enhance reliability of the data, this questionnaire could be replaced by an interview.

7.5.2 *Analysis of Need*

The 'analysis of need' phase takes the situational factors and the list of implemented capabilities from the first phase as input, after which it determines how the current methods could be improved. This phase has already been described by Bekkers, Spruit, van de Weerd, et al. (2010) in the form of the calculation process of the situational assessment method, but it will be summarized here for the sake of completeness.

The phase consists of three activities, depicted in Figure 7.5; construction of the current capability profile, calculation of the optimal capability profile and the calculation of an 'areas of improvement' matrix. The first of these three consists of translating the results from the initial maturity assessment into a form usable for further calculation.

Determining the optimal maturity level is somewhat more complex as it is based on a set of situational factor effects. Several situational indicators have an associated effect. For example, having only one product in the product portfolio could have as an effect that the process 'product line identification' can be omitted. By applying all applicable situational factors effects, one can obtain an optimal capability profile that is customized for the current company.

The current capability profile and the optimal capability profile are then combined into an 'areas of improvement' matrix. This is again a capability matrix, with both previous matrices integrated into it. Between the two matrices, a gap can exist, which can be called the delta. This delta indicates the capabilities that need to be implemented, in order to arrive at the optimal maturity level.

Important in the context of this phase is the fact that product managers can vary in the rigidity that they demand from the ME process. Some only wish a partial improvement for a specific area, while others wish to improve their methods to the maximum maturity level suggested for them. As stated before, evolutionary

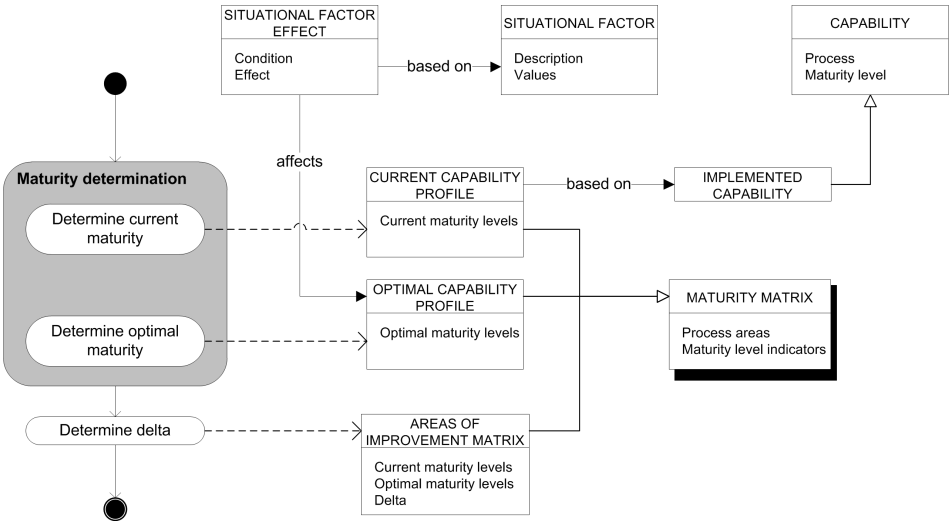


Figure 7.5.: Analysis of need

improvement is in many cases more prone to success than revolutionary change. This implies that it should be possible to provide improvements in the form of an improvement roadmap when the process is changed rigorously.

The result of this phase is thus an 'areas of improvement' matrix, describing either the entire SPM process, or a subset of it. In both cases, the delta equals a set of capabilities that need to be implemented, in order to improve the maturity of the company's SPM process and thus hopefully its efficiency and effectiveness. This set forms the basis for the next phase in the process of method improvement.

7.5.3 *Benchmarking*

Each company or business unit that makes use of the OME also provides some data of its own. For every case, a situational profile and a maturity matrix can be captured. These data can be used to benchmark companies or business units against cases of similar size, industry, market, etc.

Process benchmarking is a rather common tool for comparing the cost structure, profit, success, or other aspects of several companies (Ralston, Wright, and Kumar, 2001). Strategic planning decisions can then be made based on the results.

Pemberton, Stonehouse, and Yarrow (2001) identifies vision, training and education, a problem-solving culture and human resources strategy as some of the

key elements associated with benchmarking and organizational success. In terms of the OME, this means that benchmarking needs to be connected to the other aspects of the system, in order to gain the greatest benefits.

Another benefit of benchmarking is described by van Steenbergen, Schipper, Bos, et al. (2010). They apply benchmarking in order to validate and improve the Dynamic Architecture Maturity Matrix (DAMM), which applies a similar assessment approach as the Maturity Matrix for SPM that is used within the OME.

Until 2007, agile product development was accompanied by non-agile product management. Although several stages of elaboration were employed, no fixed cycles were used. The effect of this was that product managers did not manage to provide development with sufficiently detailed requirements before the start of each sprint. In order to improve this situation, they needed to determine if it was possible to base the management of the product backlog on Scrum principles. This would implicate a continuous adaptation of the product backlog to changing circumstances and a changing environment.

To be able to answer this question, a thorough analysis of the situational factors and the current SPM processes within the organization. Such an analysis would provide the requirements and constraints during the rest of the process improvement effort. In the case of Planon, relevant situational factors would have been the agile nature of the development process, the relatively large amount of incoming requirements, and the fairly large development organization, implying the need for a stricter organization of development work.

The OME would provide such an overview in the form of a list of situational factors and a maturity matrix. These overviews could then be used to determine and validate the room for improvement within the process of Planon. In the case of Planon, there was already a clear knowledge problem. In such a case, the assessment results (especially the situational factors) would provide the boundaries within which a solution could be found.

7.6 GUIDING PROCESS IMPROVEMENT

7.6.1 *Selection of Process Alternatives*

For the next phase, each missing capability has to be connected to a MF that implements the capability (see Figure 7.6). The OME determines which MFs are suitable according to a set of classifiers attached to each method fragment:

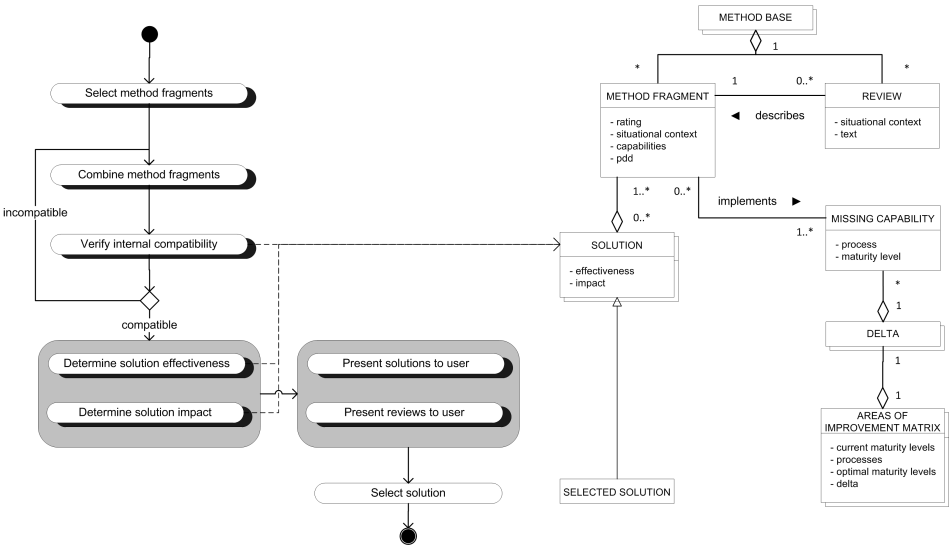


Figure 7.6.: Selection of process alternatives

Capability - The capabilities that a MF implements can be used as an attribute during the initial selection of MF candidates. Both process fragments as well as deliverable fragments can implement capabilities. For instance, a MF describing Wieger's prioritization method can be attributed with the capabilities 'a structured prioritization technique is used' and 'information related to costs and revenue is taken into account during prioritization'.

Situation - The situational factors described by Bekkers, Spruit, van de Weerd, et al. (2010) can be used for further classification. Situational factors should only be used to indicate restrictions on the use of the fragment. This is similar to the situational factor effects as described by Bekkers, Spruit, van de Weerd, et al. (2010). Examples in the case of Planon are the use of an agile development methodology and a high input of new requirements.

Experience - Through the feedback of users, MFs are rated on several aspects, such as effectiveness, complexity, etc. MFs with a very low rating can ignored in most cases, while in other cases, MFs with a high rating are selected over similar MFs with a lower rating.

Although processes, capabilities and situational factors form a very solid ground for MF selection, we need to take into account the fact that we are dealing with processes in which humans are evolved. This means that the resulting process needs to fit with the preferences of the people involved in it. These people need to be able to express these preferences during the selection of alternative MFs. Earlier interviews (see Chapter 6) have indicated that product managers are not always willing to accept suggestions made to them by an autonomous system.

Therefore, product managers or other process owners are presented a set of suitable MFs, along with a sufficient amount of information on which they can base their decision. Based on all of the sources of information combined, product managers or process owners should be able to make a valid and well-argued choice regarding the MFs that should be selected, and thus regarding the changes that should be made to the existing process.

7.6.2 *Creation of Improvement Roadmap*

In many cases, the combined set of process improvements is not suitable for implementation during one process improvement cycle. Therefore, the process improvements have to be elaborated into a set of steps or method increments. An evolutionary approach has more chance of success as it will likely yield a higher acceptance due to smaller, incremental changes.

The splitting of solutions into steps is subject to several conditions. The major condition is the existence of dependencies. For instance, implementing automated requirements registration before a centralized requirements database is in place makes no sense.

In most cases, several capabilities can be implemented at the same time. However, to make iterations or steps more successful, it is wise to make sure that each step has a goal or a theme. Doing so ensures that a set of changes is coherent, making the change-process seem less chaotic to the affected employees. This is important, as they will be the ones performing the new process.

7.6.3 *Embedding of Process Advice*

After the roadmap has been presented to the user and has been accepted, its implementation can start. In case that only maturity information is available, this process is fairly straightforward, as little support can be given. The method increments that have been proposed need to be implemented in the company 'manually', i. e. without any automated support. In order to guide this, process descriptions and templates related to the advice are provided.

If full process information is available, then this process is considerably more complex. This part encompasses the most complex aspect of method engineering, namely the assembly of MFs. During each improvement cycle, the selected method increment needs to be integrated with the existing method.

The OME can generate and/or update templates based on the original and the new process description (expressed in the Process-Deliverable Diagrams (PDDs)). If the company's original work documents are available, than they can be updated to reflect the new deliverables within the process. During this step, original data should be maintained while new columns, sections, formulas, etc. are added to the documents. In case deliverables are not available, templates can be generated based on the generated process description. The generated templates are directly usable within the new process.

In addition, the OME generates full process descriptions with explanations of all steps, deliverables and roles. These descriptions aid the process owner during the implementation of the process in the company.

7.6.4 *Improvement Retrospection*

The data within the method base are not only suitable for new process improvement efforts. A lot can be learned from previous experiences as well. The OME can provide insight into earlier attempts to implement a certain method (fragment). Data from such process improvements have the potential to help others in similar positions to successfully make the same transition.

One of the ways in which insight can be gained is through the evolution of PDDs or, more strictly speaking, process descriptions. When multiple process versions are maintained in the OME, implementation paths, goals, and fall-backs can be identified. Data like this provides insight and valuable lessons to other companies in similar positions.

Historical data can be used to perform quantitative analysis. By measuring the amount of time a certain process change has been in effect, we can obtain information about the quality of these changes. Although not much research has been performed related to the quantitative measurement of process improvement quality, we see strong possibilities in such use of historical data.

In order to cope with agility in SPM , a new categorization emerged on the product backlog and sprint backlogs. Initially, the terms 'concept' and 'theme' began to be used to group items according to the specific topic that they belonged to. Further elaboration of product features was displayed under a products-list within the Product Management Sprint Backlog (PMSB). After several months, this approach consolidated into a stable approach, which has been defined as the 'requirements refinery' in Chapter 2.

Determining up front which solutions will work and which will not is a complex job. Many factors play a role in method improvement success. The situational profile that is used by the OME provides a way to describe the context of a method improvement effort. Using experience from other organisations in similar situations, this context can be used to determine which solutions are likely to work and which are not. Such a question is a typical example of a driver for using the OME to solve a method issue. With the use of existing MFs and experiences from other companies, users can find a solution that is specific to their company.

Data related to the evolution of certain process improvements can also be valuable during similar process improvement efforts. In this case, the data has been used to reconstruct the implementation path of the requirements refinery. Table 7.2 shows the evolution of the grouping in the SPM Sprint backlog, which is one of the most important documents during agile SPM. The vertical axis contains groups that are used in the sprint backlogs to categorize product management tasks. The horizontal axis outlines the date of the sprint backlog, starting from the introduction of Agile SPM in March 2003. At the start, the document has changed often, while in later months the backlog became more stable. Groups that did not satisfy the product managers were removed quickly, while other groups have existed almost from day 1. Such information can be valuable in measuring process improvement quality and in identifying possible problems.

	2007-03	2007-04	2007-05	2007-06	2007-07	2007-08	2007-09	2007-10	2007-11	2007-12	2008-01	2008-02	2008-03	2008-04	2008-05	2008-06	2008-07	2008-08	2008-09	2008-10	2008-11	2008-12	2009-01	2009-02	2009-03	2009-04	2009-05	2009-06	2009-07
Product Knowledge Transfer	x	x	x	x	x	x	x	x	x	x	x																		
Subgroup	x																												
Web Team	x																												
Internal Operating Procedures	x	x	x	x	x																								
EE Team	x																												
Scope&Vision on CRE	x																												
Scalability	x	x																											
Licensing	x	x																											
Roadmap	x	x																											
Products		x	x	x	x	x	x	x	x																				
Subgroup		x	x	x	x	x	x	x	x																				
Product Board		x	x	x	x	x																							
Change Management		x																											
Concepts		x	x	x	x	x	x	x	x	x	x	x	x	x															
Subgroup				x	x	x	x	x	x	x	x	x	x	x															
Supplier Management		x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x								
Unplanned			x																										
Internal Improvements					x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x									
Standard Activities						x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Capital Project Management						x																							
SR Knowledge Transfer							x	x	x																				
Extras								x	x	x	x	x																	
Customer&Implementation Projects								x	x	x	x	x	x	x	x	x	x	x	x	x									
Product Features									x	x																			
Integration										x	x																		
Commited for Release											x	x	x																
Subgroup											x	x	x																
Theme Definition														x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Subgroup														x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Release Launch Support															x	x	x	x	x	x									
Requirements Elaboration															x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Subgroup															x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Development Sprint Preparation															x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Subgroup															x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Concept Definition															x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Subgroup															x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
General Items																					x	x	x	x	x	x	x	x	x
Knowledge Transfer																								x	x	x	x	x	x

Table 7.2.: Evolution of the SPM sprint backlog

7.7 CONCLUSIONS AND FURTHER RESEARCH

The software industry is showing a major shift, from project-driven software development to market-driven product development. With this change, the range of influences, issues and benefits change tremendously. The newly emerging field of SPM is seeing a steady flow of new techniques and methods that are required to deal with this new situation, developed both by academia as well as practitioners. To support this field, we have proposed the creation of a KM system for SPM methods. We think that such a system will increase the maturity of product software companies, allowing the development of products with higher quality and more revenue.

The proposed solution relies on four key areas for supporting practitioners in the area of SPM; knowledge dissemination, method assessment, method improvement, and method execution. Effective knowledge dissemination is achieved by the method-base, which forms the backbone of the approach. It provides a large variety of methods and techniques in combination with experience reports from other professionals.

To support knowledge assessment, the OME combines the context of an organization with a capability-based assessment instrument. It enables practitioners to quickly the current maturity of their processes, and to determine which areas are in need of improvement. This information can be used to focus process improvement effort, and to benchmark the organization's processes with those of other players within the industry.

By combining process assessment results with knowledge stored in the method base, the OME is able to provide context-aware process improvement advice. It supports incremental process improvement by splitting up large process improvement proposals into smaller steps. These are then combined into an improvement roadmap that facilitates the enactment of the proposed improvements. Improvement retrospection is applied to learn from historical data.

Each of the areas of the OME approach requires significant further research. First of all, the concept of method increment is essential in incremental process improvement. Current research aims at providing a strong definition and validation of this concept. Furthermore, a classification of product management activities and deliverables is being created, that will form the basis for computer-aided method engineering. Ultimately, we state that processes can be modelled, altered, expanded and downsized as if they were puzzles. Both the computational as well as the social aspect of this statement will prove very challenging. In this

paper, these topics have hardly been touched upon. However, they will become increasingly clear, and it is important to balance both of them.

Development of the OME is well under way. Currently, the focus lies on creating a web-based environment for gathering and disseminating SPM knowledge. Next steps include the implementation of the process assessment and benchmarking tools. An important factor that can never be left out during the elaboration is the fact that the purpose of the OME is the improvement of SPM processes. As a consequence, we are always dealing with people that bring habits, experiences, and opinions. Overlooking this aspect would result in a system that is too rigid, forcing people into ways of working that they will not accept, thereby foregoing the purpose of the system. Instead, knowledge sharing should be community-based and context-aware. If it is done right, than the OME has great potential value. We believe that this solution can increase the maturity of the software industry significantly by providing requirements managers, product managers and CTO's with the right tools to optimize their SPM process. In turn, product quality will rise and costs will fall.

8

ONLINE METHOD ENGINE: A TOOLSET FOR METHOD ASSESSMENT, IMPROVEMENT, AND ENACTMENT

Software companies keep evolving their methods for software production, due to the continuous changes in the organizational, technological, and societal context. Implementing changes to existing methods is a complex activity, which depends not only on understanding 'what' to alter, but also on defining 'how' to apply the changes. Approaches in the literature are mainly focused on the 'what', provide only partial answers to the 'how' question, and offer no concrete toolset to support change.

In this paper, we propose and discuss a software toolset that realizes the Online Method Engine (OME) concept and provides a concrete answer to the 'how' question by supporting the iterative and incremental assessment, improvement, and enactment of methods for software production. We describe the technical architecture of our toolset that concretely realizes our previously proposed OME concept, provide details on the method enactment mechanism, and report on a qualitative evaluation based on interviews with experienced industrial practitioners in process improvement.

This work was originally published as:

Vlaanderen, K., Dalpiaz, F., van Tuijl, G., et al. (2014). "Online Method Engine: a Toolset for Method Assessment, Improvement, and Enactment. Manuscript accepted for publication." In: *International Journal of Information System Modeling and Design (IJISMD)* 5.3

8.1 INTRODUCTION

The methods that organizations employ for software production are in constant change, as companies strive to adapt their production processes to new technologies, business models, and development paradigms. Implementing changes in existing methods is a time-and cost-consuming activity, whose success is threatened by multiple factors, including resistance to change, fear of ineffectiveness, and resource constraints (Baddoo, 2003). Unsurprisingly, method change and enactment are important subjects of research in software process improvement (Software Process Improvement (SPI)) (vom Brocke and Sinnl, 2011; Feiler and Humphrey, 1993; Gonzalez-Perez and Henderson-Sellers, 2008).

Much research has been conducted on determining 'what' changes to introduce in order to improve process, and thus software, quality. This has resulted in a wide range of frameworks for assessing and improving processes, including the Capability Maturity Model Integration (CMMI) (CMMI Product Team, 2006) and Software Process Improvement and Capability dEtermination (SPICE) (El Emam, 1997) frameworks. While useful, these approaches stay at a high abstraction level and do not answer the important question of 'how' to evolve existing methods to accommodate the needed changes.

This research provides an answer to the 'how' question from a process-oriented perspective, i. e. by investigating how to support the planning and enactment of changes in software production methods. We do not consider here the equally important cultural or human aspects, such as informing personnel about the change, delivering training sessions, etc. We are motivated by the analysis of software process improvement (SPI) success in small and medium software enterprises by Pino, García, and Piattini (2008), which reveals that two crucial factors are (1) the conduction of an iterative and incremental improvement process that introduces changes based on their priority, and (2) the usage of technical tools and platforms that support the improvement process.

In this paper, guided by these two factors, we describe the realization of a knowledge-based tool that supports the generation of process improvement plans for a specific context. Such plans link the process areas that need to be improved to the available process alternatives and the order in which they are implemented.

Our baseline consists of incremental Method Engineering (ME) (van de Weerd, Brinkkemper, and Versendaal, 2007) - a paradigm that promotes the step-wise improvement of process development processes—and the Online Method Engine (OME) (see Chapter 6–Chapter 7), the architecture of a Knowledge Management

(KM) tool that facilitates incremental process improvement by supporting knowledge dissemination as well as method assessment, improvement, and enactment.

We make the following contributions:

A. We detail the technical toolset that realizes the OME conceptual architecture. To do so, we use the well-known 4+1 architectural view model (Kruchten, 1995).

B. We illustrate how our developed toolset is a sound realization of the OME concept.

C. We detail how we support method enactment through the usage of a widely used, off-the-shelf requirements management tool that includes a workflow engine.

D. We report on a qualitative evaluation of our technical toolset that we obtained through interviews with product managers that have expertise in process improvement.

The remainder of the paper is structured as follows. We first discuss related work, showing a technical toolset for method improvement is needed by researchers and practitioners. We present our baseline and illustrate the challenges to be addressed. We describe the realization of the OME concept by showing structural aspects, the process view, and usage scenarios. We illustrate how we support method enactment through a non-trivial mapping to a workflow engine. We report on empirical evaluation of our approach. Finally, we present conclusions and outline future directions.

8.2 RELATED WORK

In Software Product Management (SPM), the main domain of this paper, the skills and practices of a product manager are part of the intellectual capital of a software vendor. However, a problem with this intellectual capital is, as Rus and Lindvall (2002, p. 1) put it, “it has legs and walks home every day”. This belief is shared by García, Amescua, Sánchez, et al. (2011), who tackle the problem by developing a software process knowledge repository developed as a wiki. Similarly, Mishra, Aydin, and Mishra (2013) have developed a basic KM tool that facilitates the capture of method knowledge. However, these approaches focus on knowledge

dissemination only, leaving the issues related to the usage of that knowledge unresolved.

In addition, many current approaches fail to gain traction within industry. While web-based knowledge bases such as the ones described above can help, the knowledge stored within them is often too rigid and does not allow feedback. Within the area of KM, a prime objective is to support not only the creation and organization, but also the development and leverage of knowledge (Wiig, 1997). Mirbel (2007) attempts to improve this by integrating the solution with Communities of Practice. However, the focus remains on knowledge dissemination.

The field of ME offers several proposals related to the modelling, modification, and dissemination of standardized Method Fragments (MFs) using automated tools, such as MERET (Heym and Osterle, 1992), MethodBase (Saeki and Iguchi, 1993), Decamerone (Harmsen and Brinkkemper, 1995), MENTOR (Si-Said, Rolland, and Grosz, 1996), MetaEdit+ (Kelly, Lyytinen, and Rossi, 1996), MERU (Prakash and Gupta, 1998), and Method Editor (Saeki, 2003). Unfortunately, most of these tools are used sparsely in practice, as users are often reluctant to use and trust ME techniques; something that is supported by the results of our evaluation interviews (see Chapter 6). In addition, there is a strong focus on the modelling aspect, and little on method enactment and improvement, with the exception of Method as a Service (Rolland and Plihon, 1998).

The ability to construct, deconstruct, combine, and alter MFs is not enough to provide adequate support during process improvement (Mirbel and Ralyté, 2006). According to several authors, including Ocampo and Munch (2006) and Rossi, Ramesh, Lyytinen, et al. (2004), method rationale is one of the key components of successful method engineering. Process owners are not only interested in what the process should look like, but also in why they should look like that (Karlsson, 2008). Current solutions are limited in their ability to combine the two views. This can result in misguided process evolution, that foregoes the original basic philosophy of a method (Karlsson, 2012).

The need for this evolutionary aspect is gaining recognition. While traditional frameworks for assessing and improving processes, including CMMI (CMMI Product Team, 2002) and SPICE (El Emam, 1997), are useful, these approaches stay at an abstract level and do not answer the important question of 'how' to evolve existing methods to accommodate the needed changes. However, Pino, García, and Piattini (2008) reveal that the conduction of an iterative and incremental improvement process that introduces changes based on their priority is crucial to successful process improvement. Unsurprisingly, new iterative SPI ap-

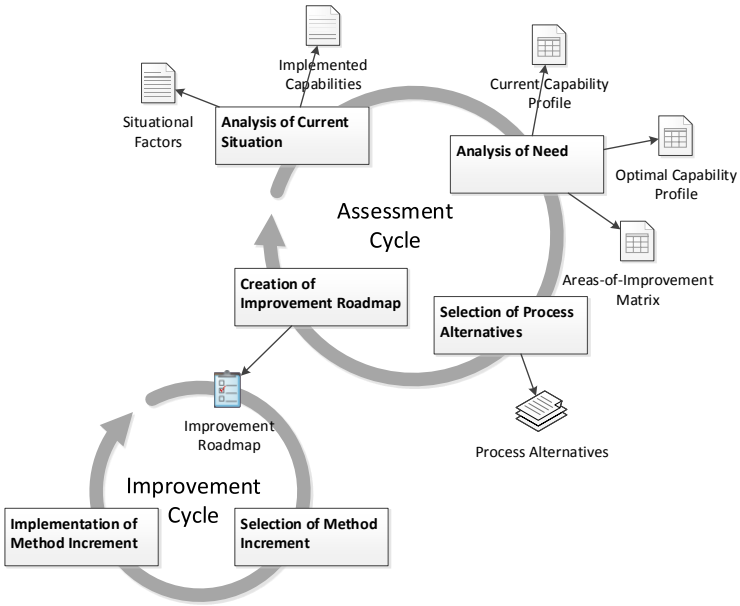


Figure 8.1.: Incremental process improvement

proaches have sprouted lately, such as the SPI-LEAN approach (Petersen and Wohlin, 2010).

Most of the recent work acknowledges the fact that method construction should always take into the account the specific characteristics of the situation in which it is to be applied. Reuse of existing knowledge is bound to factors such as cost, the existing process paradigm, and organizational characteristics such as team composition, and the availability of skills within the organization (Becker, Janiesch, and Pfeiffer, 2007). The results from previous evaluations support this strongly, and these factors are guiding current and future design cycles of the OME.

8.3 RESEARCH APPROACH

8.3.1 *Baseline*

Situational Method Engineering (SME) (Becker and Knackstedt, 2007; Brinkkemper and Harmsen, 1995; Ralyté, 2004) suggests that development methods are constructed and configured depending on the specific situation (characteristics,

environment) of the project under consideration. Methods are assembled starting from existing reusable components, which are tailored to the situation and integrated with the other methods that are already in place in the organization.

A key success factor for SME is that improvement—i. e. the integration or replacement of components—has to be conducted incrementally (Harmsen, Brinkkemper, and Oei, 1994; Pino, García, and Piattini, 2008), through a series of small steps that fit well with the situation at hand. Our research over the past few years has focused on supporting evolutionary process improvement through incremental method evolution (van de Weerd, Brinkkemper, and Versendaal, 2007). The starting point was the definition of the atomic types of adaptations, called increments (van de Weerd, Brinkkemper, and Versendaal, 2007).

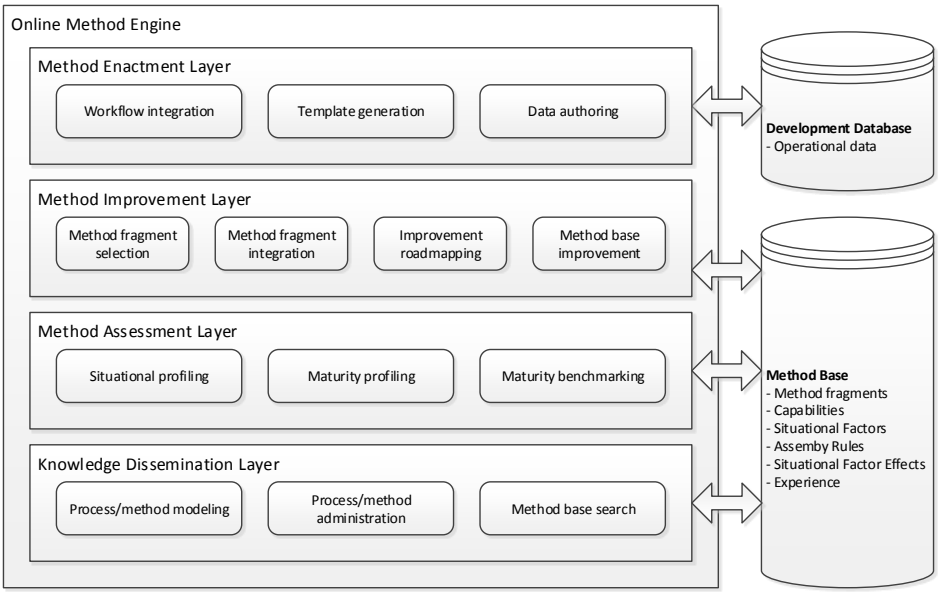


Figure 8.2.: Conceptual model of the OME

In Chapter 6, we have proposed a high-level approach (see Figure 8.1) that supports incremental method evolution. This is based on the situational assessment method by Bekkers, Spruit, van de Weerd, et al. (2010), influenced by process improvement approaches such as the Deming cycle of Plan, Do, Check and Act (Deming, 2000), and the Quality Improvement Paradigm (Basili, 1993).

The approach consists of an outer process improvement cycle (assessment) that triggers an inner cycle (improvement). Process improvement starts with the analy-

sis of the current processes, which outputs in a maturity profile. The organization's situational context (Bekkers, van de Weerd, Brinkkemper, et al., 2008) is used to determine an optimal maturity profile to reach. The process improvement effort aims to bridge the gap between the current maturity level and the optimal one. While doing so, suitable MFs are combined into a new method that is integrated with existing processes in the organization.

The OME is a KM system that supports incremental process improvement. The proposal of the OME was motivated by the need of a framework that makes available the necessary resources for incremental process improvement (methods, techniques, tools, and templates). The conceptual architecture of the OME is shown in Figure 8.2. A more detailed description is provided as part of the research contribution presented in this paper.

8.3.2 *Problem Statement*

The design process of the OME can be described in terms of the Systems Development Research Process (SDRP) by Nunamaker Jr., Chen, and Purdin (1990) (see Figure 8.3). The first design has been presented in the form of the product software knowledge infrastructure (Product Software Knowledge Infrastructure (PSKI)) that was proposed by Brinkkemper (1996) and van de Weerd, Versendaal, and Brinkkemper (2006). The PSKI included activities that, starting from the analysis of current situation, resulted in the implementation of changes in the existing process. The conceptual architecture resulted from a new design cycle, incorporating the results from a qualitative evaluation round. This evaluation has been described in an earlier paper (see. Chapter 6), resulting in a set of observations that were used to revise the original conceptual design.

While the conceptual architecture was considered as promising (see. Chapter 6), our empirical experience has identified a number of major obstacles that affect the field in general, and the applicability of the OME in particular:

- Process improvement framework mismatch: process improvement frameworks are too heavyweight and require very experienced analysts to be effective.
- Science/practice gap: existing approaches are not turned into practical solutions by practitioners, which do not access the literature and deem it as too complex.

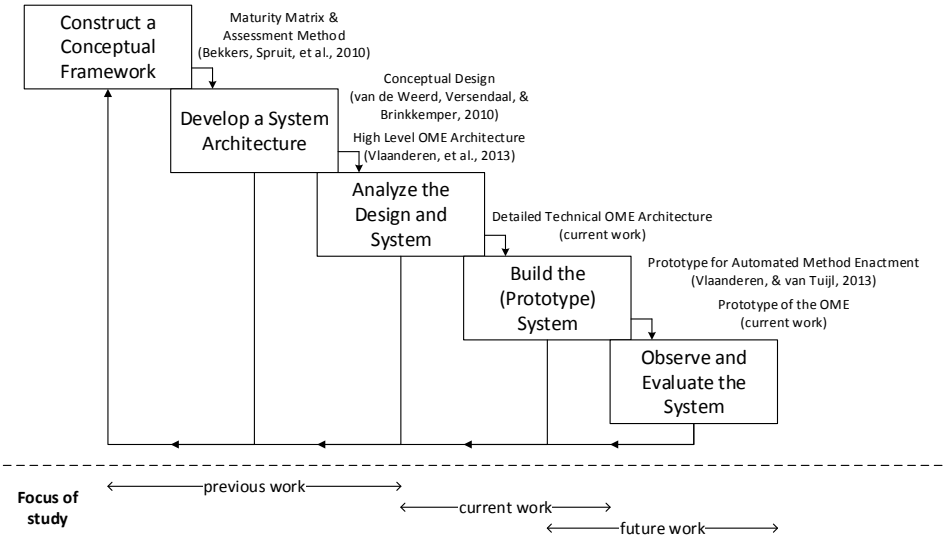


Figure 8.3.: Systems development research process, annotated with earlier and current work

- Low SPM maturity: most organizations score low on the SPM maturity matrix, and are thus unprepared to develop their own process improvement tools.

The problem that we address here is to overcome the listed obstacles, for they prevent the adoption of method improvement initiatives in industry. Our proposed solution is to describe a realization of the OME concept, which can be used as such by practitioners, or can be taken as a reference for developing custom frameworks by larger organizations.

8.4 REALIZATION OF THE OME: STATIC VIEW

In this and the next two sections, we use the 4+1 architectural view model (Kruchten, 1995) to illustrate our realization of the OME concept. In this section, we start with the static view of the architecture, which includes the logical, development, and physical views.

8.4.1 *Logical View*

This view corresponds to the original conceptual architecture in Figure 8.2. The OME contains four functional layers. Each layer includes several functional components, shown by the rounded boxes. The approach integrates techniques for sharing knowledge, and assessing, improving, and implementing methods.

The bottom layer is related to knowledge gathering and sharing. In the current approach, methods can be modeled in the form of process-deliverable diagrams using MetaEdit+ (Tolvanen and Rossi, 2003). Any meta-information can be added through the method administration module. In this manner, a method base is built that can be searched for relevant MFs.

The method assessment layer deals with qualitative analysis of methods using situational profiling and maturity profiling. These profiles can be used for direct benchmarking against other organizations. Based on the results of the assessment, new MFs can be selected from the method base. These MFs can be integrated into the current method, and in case of large improvements, an improvement roadmap can be generated.

The top layer—method enactment—includes workflow integration to synchronize method descriptions with actual process data, template generation to update existing tools such as spreadsheets and requirements databases (development database in Figure 8.2) according to the updated method, and data authoring to manage company-specific process data.

8.4.2 *Development View*

The OME is implemented on top of the web content management system GX WebManager (Souer and Mierloo, 2008). Its feature set contains all the content management functionality required, e. g. web-based forms, authorization, versioning, and workflow management. At the same time, it is flexible and extensible, for it consists of components and services that are connected at startup time.

Functionality related to the OME is developed as a set of hot-pluggable components and services based on the concept of an open service gateway initiative (OSGi) bundle . These components and services add a set of custom user interface components to the content management system, which allow one to create and maintain a specialist website containing situational method knowledge.

Table 8.1.: Data types within the OME

OME CONCEPTS	DATA TYPE
Process-Deliverable Diagrams	XML-Documents
Situational Profiles, Maturity Profiles	Relational Data
Alternative Metamodel Formats (e. g. Visio)	Binary Files

The technical architecture of the OME is based on components and services because, even though the design and implementation are currently guided by the needs of SPM research, we aim to ensure generality. The solution is designed to be applicable to other domains by replacing, adding or extending components that are domain-specific, such as the capability model, the list of situational factors, and the available tool connectors.

The individual components' design follows the model-view-controller design pattern and is based on the Spring MVC framework . Most component types have multiple views, so we can provide customizable presentations for the knowledge in the repository, where editors configure component instances in 'edit view' before they get published.

The research data is a mixture of XML-documents, objects, relational data and possibly binary files (see Table 8.1 for an overview of the main data types currently handled by the OME). The Java Content Repository (JCR) (JSR-170) implementation in WebManager can handle these different types of data through a single, unified API.

WebManager (van Berkum, Brinkkemper, and Meyer, 2004; Souer, Joor, Helms, et al., 2011) builds on standard technologies and products (e. g. Java, Apache Tomcat, Apache Felix, SQL) that can be scaled easily. This enables the gradual extension of the OME from a small research prototype to a publicly available resource. The OME depends on various WebManager services, merges with its user interface, and builds on the underlying repository.

The users, organizations, and methods can be maintained through a set of UI components. These components access the data layer through a set of Data Access Objects (DAOs) that is connected to an entity manager. Another set of UI components is used to implement all functionality related to the scenarios described below. This set can be used by a website editor to publish selected information

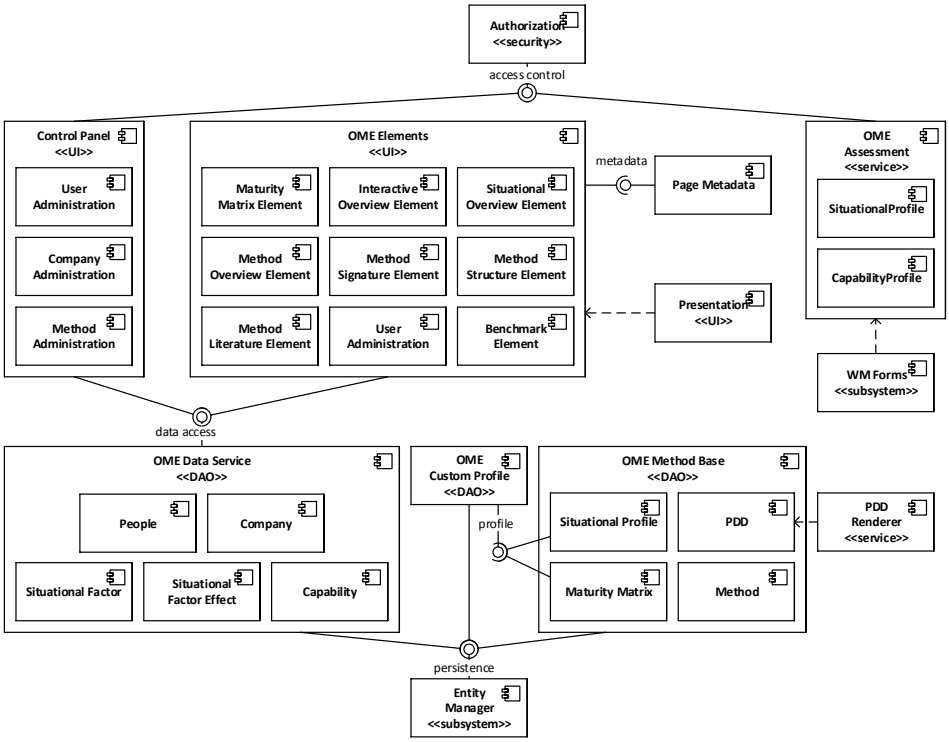


Figure 8.4.: Component diagram of the OME

from the method base on the website in a suitable form. The assessment results are handled by a set of services that link the data in the method base to an extended user profile that is capable of maintaining all data relevant to a specific user or organization. The UML 2 component diagram (Object Management Group, 2005) in Figure 8.4 shows all relevant components and services. The diagram includes both generic WebManager specific components, as well as OME specific components.

8.4.3 *Physical View*

The implementation of the OME conceptual architecture addresses issues that go beyond the domain of process improvement. An important goal of our implementation is to provide a repository of MFs and assessment results that serves

to improve the overall quality of SPMprocess and increase the understanding of best practices and the characteristics of process improvement efforts.

The conceptual architecture does not detail where this data resides and who has access to it. Our implementation differentiates between data in the public domain and data in the private/corporate domain. The former category includes generic fragments from the literature, analytical tools (e. g. capabilities and situational factors), and rules originating from empirical research that relate these components (e. g. situational factor effects). The latter category consists of organization-specific data, such as enacted methods and assessment results.

By design, the OME's functionality goes beyond mere process improvement support. An important goal is the acquisition of empirical data that can be used to infer hypotheses related to successful combinations of situational context and MFs. This introduces privacy-and competition-related issues. Although many organizations are willing to provide confidential data for academic purposes, there are concerned with safeguarding their competitive position. This concern is reflected in the physical design of the OME.

The deployment diagram in Figure 8.5 shows our implementation, with the major nodes and their relationships. The dashed vertical line determines the boundary between the publicly available nodes (left-hand side) and the private organization-specific nodes (right-hand side). There are two types of frontend and backend servers, public and private. The organization-specific backend server connects not only to the organization-specific method base (from retrieving private methods and assessments), but also to the public method base, that contains generic situational factors, capabilities, situational factor effects, and methods.

8.4.4 *Mapping to Conceptual Architecture*

As described above, one of the goals of this paper is to demonstrate how we implemented the conceptual architecture shown in Figure 8.2. The static views detailed in the previous sections provide a thorough overview of the various techniques we used and the design choices we made. In order to link the previously designed conceptual architecture to its technical counterpart, we have outlined each conceptual component to the relevant technical components in Table 8.2.

The left-hand side of the tables lists all conceptual components of Figure 8.2. The right-hand side shows the techniques that we employed to satisfy the requirements for that component. While some of these techniques link directly

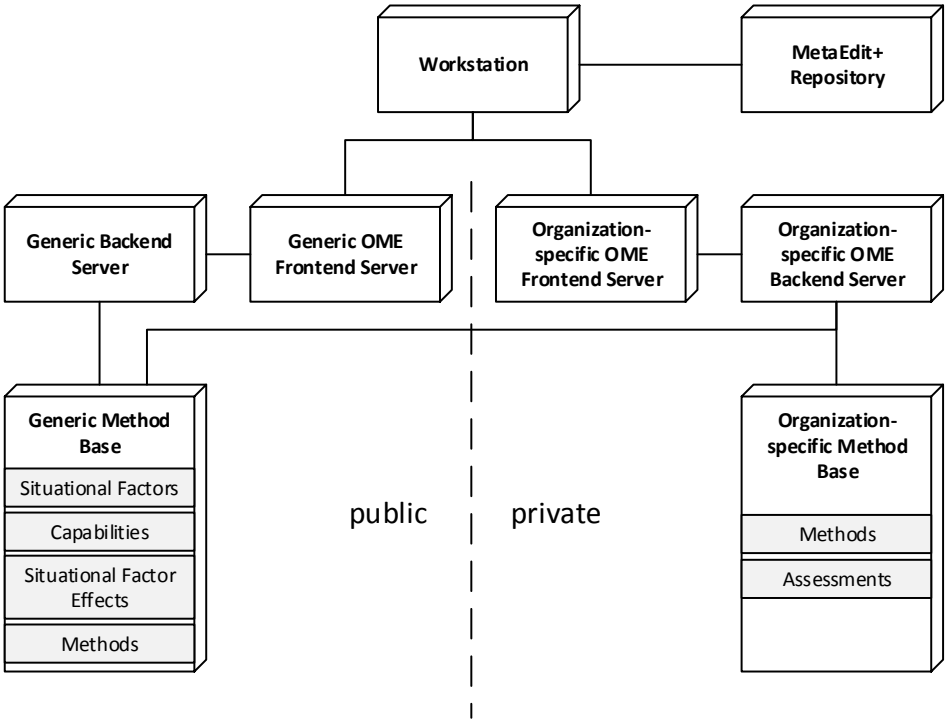


Figure 8.5.: Deployment diagram of the OME

to components shown in Figure 8.4, others refer to external tools or techniques from the ME domain. For example, several components refer to the Situational Assessment Method, described by Bekkers, Spruit, van de Weerd, et al. (2010).

In the following sections, we describe the role that these components play during a process improvement activity in more detail.

8.5 REALIZATION OF THE OME: DYNAMIC VIEW

We describe the dynamic aspects of our implementation. We first present the overall process view, and then focus on the five main usage scenarios that the OME allows for.

Table 8.2.: Mapping the conceptual architecture to the prototype realization

CONCEPTUAL COMPONENT	IMPLEMENTED THROUGH
Method Modeling	MetaEdit+ / Visio
Method Administration	Method Administration Panel
Method Base Search	Method Overview
	Facetted Search
Situational Profiling	Situational Profile (SAM)
Maturity Profiling	Capability Profile (SAM)
Maturity Benchmarking	Capability Profile with Aggregated Data
Method Fragment Selection	Areas of Improvement Matrix (SAM)
	Method Base Facetted Search
Method Fragment Integration	Method Assembly Techniques
Improvement Roadmapping	Method Increment Planner
Method Base Improvement	Situational Profile (SAM)
	Capability Profile (SAM)
	Feedback Form
Workflow Integration	Method Enactment Mechanism
	Link with CASE tools
Template Generation	Method Enactment Mechanism
Data Authoring	Not implemented yet

8.5.1 *Process View*

The OME supports a set of distinct goals that practitioners are interested in achieving during process improvement. These goals are tightly related to the layers in the logical view, and are in line with the incremental process improvement paradigm in Figure 8.1. We describe how the process view—illustrated via the Process-Deliverable Diagram (PDD) (van de Weerd and Brinkkemper, 2008) in Figure 8.6—realizes these goals.

The figure outlines the processes, the artifacts that are created and used, and their interdependencies and relations. The process side describes a sequential path

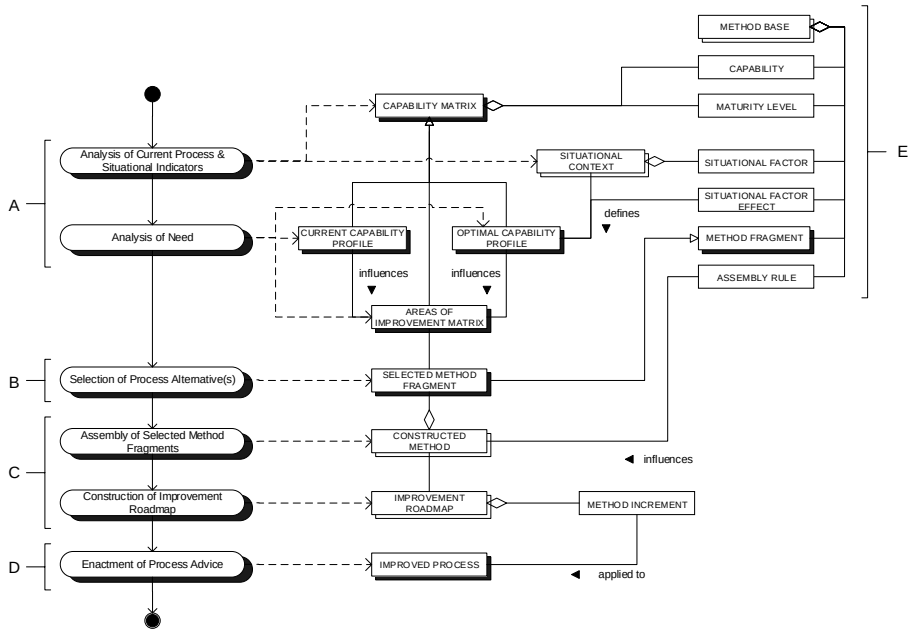


Figure 8.6.: Meta-process view of the OME

through the functionalities of the OME. However, complexity arises from the fact that each step is typically performed iteratively, and that the results of each step result in modifications of the data in the method base through the use of feedback mechanisms.

8.5.1.1 *Method Assessment*

The OME process starts with the analysis of the current processes and situational indicators. This activity results in filling in a capability matrix and a situational context. This is followed by an analysis of the need, which determines the current capability profile, the optimal profile to be achieved through the process improvement effort, and the areas of improvement. These processes are detailed through Scenario A below.

8.5.1.2 *Method Discovery*

The process follows with a selection of process alternatives for reaching the optimal profile. This requires the manual discovery of relevant MFs that are compatible with a specific organizational context, as described in scenario B.

8.5.1.3 *Method Improvement*

The chosen fragments are assembled within a constructed method, and the definition of an improvement roadmap that defines a plan for integrating the new fragments. This part of the OME process is detailed in Scenario C.

8.5.1.4 *Method Enactment*

The OME main process terminates with the enactment of process advice, which results in the improved process. The process should lead to a situation in which the organization meets (or is very close to) the optimal capability profile. This phase is described in Scenario D.

8.5.1.5 *Method Administration*

In addition to the processes in Figure 8.6, the OME also supports a general-purpose process that concerns the administration of methods. This is detailed in Scenario E below.

8.5.2 *Scenarios*

We detail the five major usage scenarios of the OME. They are closely tied to the layers in the OME conceptual model, and further refine the process view described above.

8.5.2.1 *Scenario A: Method Assessment*

The method assessment in OME provides a structured overview of the quality of an organization's SPM process, providing the hammer and screwdriver during a method improvement effort. During the last few years, several of the concepts related to method assessment and improvement in SPM have been proposed and elaborated. (Bekkers, van de Weerd, Brinkkemper, et al., 2008) suggest that the

general assessment of an organization or a specific business unit within that organization can be performed using a set of situational factors. The assessment of an organization's SPM methods can be performed using a set of capabilities, which have been obtained through extensive empirical analysis (Bekkers, Brinkkemper, van den Bemd, et al., 2012). Based on the results of these two assessments, a maturity matrix can be determined that outlines the current maturity of the organization (see Figure 8.7).

Maturity	0	1	2	3	4	5	6	7	8	9	10
Requirements management											
Requirements gathering		A		B	C		D	E	F		
Requirements identification			A			B		C			D
Requirements organizing				A		B		C			
Release planning											
Requirements prioritization			A		B	C	D			E	
Release definition			A	B	C				D		E
Release definition validation					A			B		C	
Scope change management				A		B		C		D	
Build validation					A			B		C	
Launch preparation		A		B		C	D		E		F
Product planning											
Road map intelligence				A		B	C		D	E	
Core asset roadmapping					A		B		C		D
Product roadmapping			A	B			C	D		E	
Portfolio management											
Market analysis					A		B	C	D		E
Partnering & contracting						A	B		C	D	E
Product life cycle management					A	B			C	D	E

Figure 8.7.: Example method assessment result in the OME

8.5.2.2 *Scenario B: Method Discovery*

Each method is described in a separate page of the system. These pages contain all data relevant to the particular method, as described in the previous scenario. In addition, several tools are provided that generate overviews of the available methods. With these tools, the method base becomes a fully-searchable database of method knowledge, allowing method administrators or process owners to quickly find relevant methods.

The main entry point for this activity is a form that allows faceted search, enabling to user to filter the available methods based on business function, situational relevance, implemented capabilities, and keywords. Throughout the system, users can perform contextual searches that result in an overview of methods that are relevant to the current system state. For example, during the improvement activity, users can easily access methods that are similar to the methods that have been proposed by the system, in order to gain a more complete view and enabling them to make well-informed choices.

8.5.2.3 *Scenario C: Method Improvement*

The assessment results that are obtained during scenario A are the main input for the improvement activities supported by the OME. In the current implementation, these activities include the selection of MFs that could improve the assessed process, the integration of the selected MFs into the current process, and the creation of a roadmap that supports the enactment of the improved process description.

A selection of relevant MFs is determined based on the situational profile of the organization in combination with the set of missing capabilities. This set is the delta between the currently implemented capabilities and the capabilities at or below a maturity level indicated by the user. This implies an explicit choice regarding the relevant business functions and the goal maturity level.

The system queries the method base using the set of requested capabilities and the situational factors as constraints. The results are shown to the user in summarized form. When multiple fragments are available with a similar signature (i. e. similar capabilities and situational constraints) then the user is given a choice between these fragments. Based on the meta-information of the MFs, such as description and rationale, the user can select the fragments that he deems suitable.

If the user has provided a process description of the current process in the form of a PDD, preferably with the help of a method engineer, then the selected MFs can be integrated into that process. This integration is performed based on the dependencies between the deliverables described by the MFs. Any issues related to this integration, such as duplicate capabilities and incompatible MFs are reported to the user, who can then make appropriate changes to the set of selected MFs.

The integrated process is then broken down again into atomic pieces based on the dependencies between deliverables. The resulting set of atomic fragments is prioritized based on factors such as the maturity level of the capabilities they

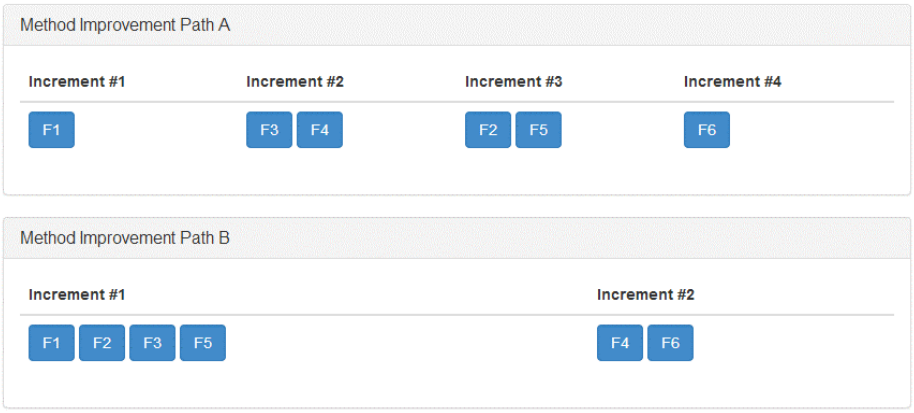


Figure 8.8.: Method improvement paths

relate to. This prioritized list is used to determine one or more paths that support the implementation of the proposed changes, as illustrated by Figure 8.8. These paths (depicted by the large boxes) consist of multiple increments (depicted by the columns) each containing a set of one or more fragments (depicted as dark boxes), allowing an incremental improvement style. A sidebar shows details regarding the changes contained within each steps, such as added or removed activities and deliverables.

8.5.2.4 *Scenario D: Method Enactment*

Enacting process improvements is a complex undertaking influenced by many factors such as organizational culture, technological constraints, and management support. Such a problem cannot be solved by a tool, as it is largely a social challenge. However, we believe that the OME can provide some support related to the enactment of changes from a technological perspective. For this purpose, the OME incorporates a mechanism for translating the method changes to tool changes.

The mechanism requires a mapping between the system and one or more tools that are used within the process. Each increment is translated to a set of changes to the tool. Examples of such changes are added screen or changed properties to specific models. The results of each transformation are displayed to the user for review. A detailed description of this process is provided in the next section.

8.5.2.5 *Scenario E: Method Administration*

The method base contains the nails and screws of the OME. It is a fundamental concept of method engineering, and it consists of a set of MFs that can be combined into a mature and situational SPM method, in addition to experience related to the applicability and usability of those MFs.

In the OME, new methods enter the method base through an administration panel. Each method is described in terms of a name, a goal, a free-format description, the situation in which it is applicable, the capabilities that it helps implementing, and relevant literature.

A method can be further detailed in terms of a PDD. When this is the case, the method administrator separately draws the diagram using the MetaEdit+ modeling tool, which can be imported into the OME (see the spotlight), and linked to the appropriate method.

8.6 TECHNICAL ZOOM-IN: METHOD ENACTMENT

We detail how the incremental method enactment components of the OME realization have been implemented by interfacing with the Jira requirements management tool. Jira integrates a workflow engine that we use to support method enactment.

A thorough explanation of the process that we followed to select an appropriate SPM tool is provided by Vlaanderen and Tuijl (2013). The main criteria for our choice were the support of workflow management, the capability to support documents, simple-to-use configuration, modifiability, and that it is widely known and used in the industry.

8.6.1 *Conceptual Mapping*

The mapping fromPDDMFs to Jira's data model is shown in Figure 8.9. This mapping is essential to allow using Jira's workflow engine for method enactment.

The left-hand side of Figure 8.9 shows a simplified PDD fragment. The right-hand side of the same figure shows the essential elements of Jira's data model. The arrows link the MF meta-model to the data model. Table 8.3 provides a textual explanation of the relationships between both fragments.

The diagram name of the fragment that a user creates is used to create an issue type in Jira. For instance, an initial PDD for a fragment called 'Requirement

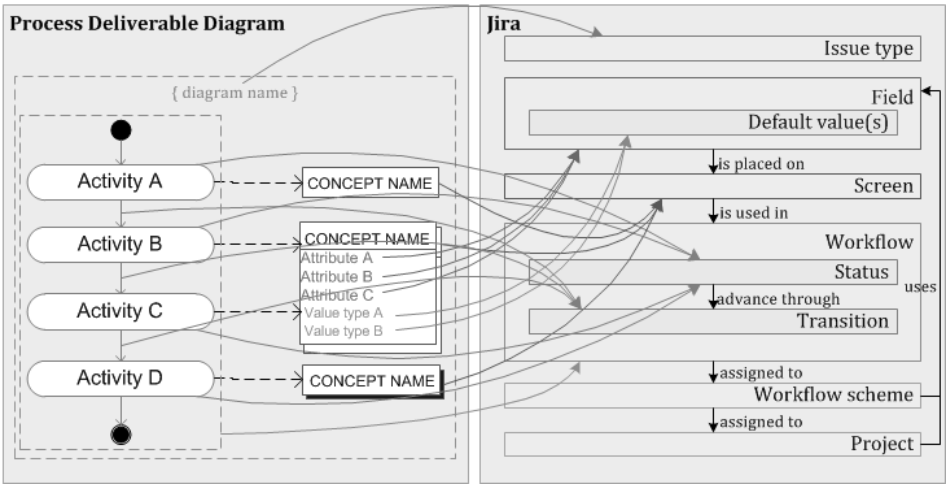


Figure 8.9.: Mapping of PDDs to Jira's data model

management process' would result in an issue type in Jira. All the data that is entered in Jira in relation with this issue type belongs to that specific fragment. When the PDD of the fragment is updated, the existing issue type will not be affected. Issue types can be used in multiple projects.

The complete process side of the fragment is mapped to a workflow in Jira. A workflow is represented by a name and the entire workflow is based on an XML structure. Structurally, workflows in Jira are similar to processes in PDDs. A workflow consists of a status (of an issue) and transitions between statuses. A status is a step in the workflow through that, when completed, advances the workflow. For instance, after completing activity A in a workflow and marking this as complete, the workflow advances to next status (i. e. the next activity). Each status is linked to a screen so that a status can present information to the user.

Jira advances from a status to the next by following the path defined by transitions. PDD transitions are mapped 1-to-1 to Jira transitions. The name of the transition is based on the upcoming activity in the PDD.

The name of a PDD concept is used to create a screen in Jira. For instance, given a concept called REQUIREMENT, a screen is created with the name REQUIREMENT. A screen contains fields that are based on attributes of the PDD concept. If a concept does not contain attributes, the screen does not have any input fields. The screen is linked to a status that can contain many other functions such as attaching a document or adding comments to a particular status.

Table 8.3.: Mapping of method fragments to Jira

#	PDD FRAGMENT	RELATIONSHIP	JIRA FRAGMENT	USAGE DOMAIN
1	Diagram name	is used to create an	Issue type	per project
2	Process side	will produce a	Workflow	per project
3	Activity	is used to create a	Status	in a workflow
4	Transition	is used to create a	Transition	in a workflow
5	Concept	is used to create a	Screen	per project
6	Attribute	is used to create a	Field	per project
7	Value type	will be used as (a)	Default value(s)	per field

8.6.2 *Technical Description*

After a fragment has been created in MetaEdit+, a MERL script (the MetaEdit+ scripting language; Tolvanen and Rossi (2003)) exports the fragment to an external tool. This tool extracts the relevant meta-data from the fragment, including the fragment name and its version, and converts the fragment to an intermediate structure based on the PDD meta-model as defined by van de Weerd and Brinkkemper (2008).

Once all relevant data has been extracted, it is shown in a user friendly manner to the user, who can verify data consistency. Once the data is verified, the enactment process is triggered. For each relevant database table of Jira, three objects are created based on the Model/View/Controller (MVC) design pattern; a Value Object (VO), a DAO, and a Controller Object (CO). These objects form the connection between MetaEdit+'s data structure and Jira's data structure.

Through a set of prepared SQL statements, the data fragment is then stored in Jira's configuration database. Prepared statements are used to increase safety and avoid SQL injection (i. e. attacking the database deliberately through query manipulation). We used Persistent Domain Object (PDO) in conjunction with MySQL 5.0 because this allows us to abort the entire querying execution process when a single query fails (it supports transactions with commit/rollback commands).

Figure 8.10 shows the method increment and tool increment based on increment #4 in van de Weerd, Brinkkemper, and Versendaal (2010). On the left side of Figure 8.10, the fields for the respective concepts of increment #4 are shown. The fields in Jira are based on the attributes of the concepts REQUIREMENTS

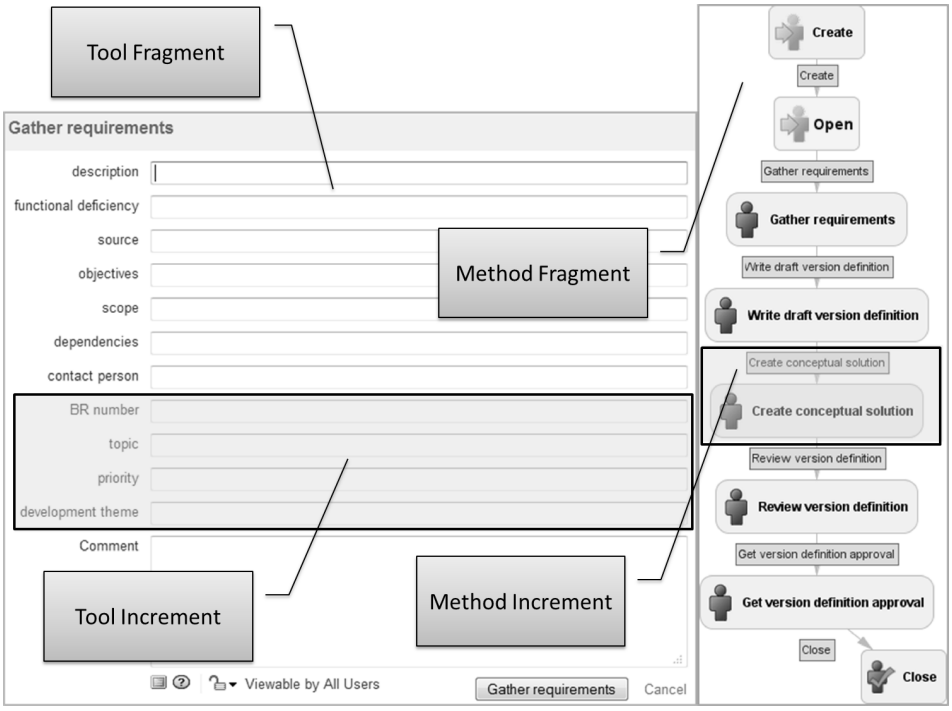


Figure 8.10.: Adapted method fragment in Jira

DOCUMENT and REQUIREMENT. On the right hand side of Figure 8.10, the workflow is shown that is in accordance with the workflow in fragment #4. The shaded areas denote the method and tool increment.

8.7 EVALUATION

Any design activity is a process involving multiple iterations of design generation and design testing (Hevner, March, Park, et al., 2004). The work described in this paper represents one of these cycles. We report here on a qualitative evaluation of the overall implementation and the enactment mechanism.

8.7.1 *Expert Evaluation of the OME Prototype*

8.7.1.1 *Definition*

For this evaluation, we are mainly interested in the opinion of practitioners regarding the implementation, since the conceptual design has been evaluated in earlier work, as described above. For this reason, we structured the interviews around the following questions:

- How well does the OME implementation support the described scenarios?
- Is adequate information provided at key points?
- Is this an effective approach for searching, visualizing, and managing MFs?
- Would this concept be feasible in a real-life setting?

8.7.1.2 *Planning*

We selected five product managers from product software organizations in the Netherlands for the evaluation of the current implementation of the OME. These product managers were not previously familiar with the OME concept.

8.7.1.3 *Execution*

The evaluation is structured around the five scenarios described above. We interviewed the participants in separate sessions, to prevent any contamination of opinions. Each session lasted approximately 45 minutes. First, the main concepts were explained. Then, the implementation was demonstrated according to the five scenarios. After each scenario, the participant's interpretation was checked against the original intent. Once it was verified that the interpretation and intent were aligned, we obtained feedback through a semi-structured interview.

During the interviews, the participants were encouraged to provide additional comments, in response to the screens that they were shown. This provided us with some interesting ideas and insights for future design iterations.

8.7.1.4 *Results*

Most participants envisioned the method base as a collaborative system, where practitioners could share experience and thus develop a higher standard of product

management processes and perhaps some sort of standardization. None of the participants opposed the idea of sharing these experiences, on the condition that all information would be anonymized.

The style of continuous, iterative improvement was in general highly appreciated. The overall process (assessment, interpretation, improvement, and enactment) was deemed relevant and realistic, and the fact that process owners would have appropriate tools for process improvement was much appreciated. They also liked very much the possibility of retrospection, i. e. looking back at earlier assessment and reflecting on the changes. This confirms the results from our earlier study (van Stijn, Vlaanderen, Brinkkemper, et al., 2012). The participants indicated a wish for improved support in this area, in the form of a historical overview of earlier assessments and improvements.

Scenario A

The assessment forms posed no significant problems to the participants, but the maturity matrix was found difficult to interpret. We showed a condensed version, which everyone found easy to interpret, and an expanded version, which is the original form and shows more detail, but which was harder to interpret. The interviewees suggested introducing improved visual aids such as a legend and detailed information regarding the capability levels.

Scenario B

With respect to the method base and the way in which MFs are presented, the participants expressed concerns regarding the origins of MFs. A common perception was that high-quality method descriptions would be critical to any successful implementation of the OME. In addition, some participants indicated the need for more detailed data such as common usage scenarios of the MFs, templates, roles, and meta-data such as complexity.

Scenario C

The main concern regarding the improvement scenario was that participants were not convinced that the system would be capable of reflecting and incorporating the complex situation of ‘the field’. Soft factors such as team culture, costs, and the existence of ad-hoc processes were indicated as confounding factors that were insufficiently addressed in the design.

Regarding the improvement roadmap, the experts mentioned the need for more detailed information concerning the contents of each improvement step as well as the rationale behind the scenarios. This last point was related to the need of prioritization of improvement suggestions, as indicated by a few participants. A

recurring comment was the importance of the distinction between quick wins and improvements with a larger impact on resources.

Scenario D

The enactment mechanism was only briefly discussed during this evaluation round. Due to its complexity, this part was evaluated in a separate study. The results are described in a separate section below.

Scenario E

The administration of the method base was not deemed relevant to the profile of the participants. Therefore, we focused on functionality related to the discovery and usage of knowledge.

8.7.1.5 *Interpretation*

The results of this evaluation round provide an early, high level of the areas of the system that require thorough analysis during further elaboration. A limitation during any prototype evaluation is the expectancy of the participant, for whom it is difficult to separate the proposed functionality and workflow from the visual design aspects. The evaluation results of scenario A and B suggest the need for an analysis of information needs. However, no major architectural changes seem to be required.

Feedback for all scenario's confirmed the earlier observation that practitioners are skeptical towards suggestions provided by automated, knowledge-based reasoning. A real life setting will always be constrained by conflicts amongst stakeholders, limited resources, and the need to balance short and long term goals. In the context of MF integration and planning, participants indicated that the proposal has merits, but that they need assurance that the results are reliable and robust.

8.7.2 *Technical Evaluation of the Enactment Mechanism*

8.7.2.1 *Definition*

The goal of the technical evaluation of the enactment mechanism is to obtain concrete insight into the capabilities and shortcomings of the current solution. We do so by simulating a series of successive method increments and their enactment in the SPM tool as described above.

8.7.2.2 *Planning*

The data for this evaluation consists of a base method, and four method increments that describe changes to this base method. To emphasize the notion of mimicking a real-life setting, we selected the base method and method increments from a previously conducted case study (van de Weerd, Brinkkemper, and Versendaal, 2010) at a vendor of ERP software (see Table 8.4 for an overview). The increments show the evolution of a requirement management method that advances from a very simple method to a more elaborate approach. The increments, along with a textual elaboration of each increment, can be found in the paper by van de Weerd, Brinkkemper, and Versendaal (2010).

Table 8.4.: Overview of method increments

INCREMENT	GOAL
0	Introduction requirements document
1	Introduction design document
2	Introduction version definition
3	Introduction conceptual solution
4	Introduction requirements database, division market and business requirements, and introduction of product families

8.7.2.3 *Execution*

We executed the enactment mechanism with the method increments listed above as input. After enacting each method increment, we compared the resulting tool configuration to the method description. Although the enactment tool is deterministic in nature, this approach allowed us to discern subtle shortcomings in both the realization of the tool as well as the constraints that can be posed by an industrial tool.

8.7.2.4 *Results*

During the development of the prototype, we encountered some issues that still inhibit a fully automated approach to method enactment.

STANDARDIZATION OF PDDS. While the PDD format has been clearly defined and formalized (van de Weerd and Brinkkemper, 2008) in practice we encountered different dialects of PDDs. This is a limitation in that many existing fragments cannot be transformed, due to their deviation from the standard.

SUPPORT FOR ABSTRACTION LEVELS. PDDs support the visualization of different levels of aggregation and abstraction within the process. The ability to implement these levels highly depends on the workflow engine. This has been an obstacle in Jira; as a result, our technique currently does not support all types of activities.

USAGE OF FORKS, JOINS AND DECISIONS. The use of forks, joins and decisions is very common in processes. However, many workflow engines do not fully support these constructs. While it supports forks and joins, Jira does not explicitly support decision points; fortunately, one can add conditions to transitions to mimic this behavior.

GENERALIZATIONS, ASSOCIATIONS AND AGGREGATIONS. We could not find a way through which we could make sound use of generalizations, associations and aggregations.

8.7.3 *Expert Evaluation of the Enactment Mechanism*

8.7.3.1 *Definition*

Our interviews focused on assessing the rationale behind the approach, the feasibility of the enactment mechanism, and the required adjustments.

8.7.3.2 *Planning*

We conducted 5 semi-structured interviews with product managers from different medium/large-scale organizations in the Netherlands. These participants were not the same people who evaluated the OME prototype.

8.7.3.3 *Execution*

We posed a total of 14 questions. 7 of these were aimed at eliciting information directly related to the companies' current situation. The other 7 focused on the

evaluation of the enactment mechanism. A demo of the enactment mechanism was shown to the product manager in the form of a video that showed the mechanism in action . During the video, a verbal explanation was provided to make things clearer to the product manager. At the end of each interview, time was allocated to document any kind of other information that was provided by the product manager, i. e. remaining concerns and/or opinions.

8.7.3.4 *Results*

Each of the interviewees agreed on the fact that process adaptation is an important and continuous activity throughout the organization or department. Each organization struggled in the beginning with their business process(es), and each company reinvented the wheel numerous times before a standard process was in place. Out of the 5 companies, 4 had process models in place, mostly at a high level of abstraction. The remainder company did not have any model in place due to its small size. Over time, each company shifted from having a very detailed process model to a more high-level process.

The interviewees agree that a major issue with the current approach is rigidity. A successful enactment mechanism needs to be flexible, supporting non-linear, complex methods, and allowing exceptions. This includes the support for decisions, multiple stakeholders, and a flexible stance towards the relation between the process model and reality. Most interviewees agreed that any enactment approach that forces people to work in a specific way would be met with much resistance, which is a result that is in line with recent work in software process improvement (Baddoo, 2003; Sulayman, Urquhart, Mendes, et al., 2012).

The interviewees also commented on the fact that the current approach does not deal with all possible process model changes. It focuses mainly on the addition of steps and/or deliverables, while leaving out the deletion of process elements.

8.8 CONCLUSIONS AND OUTLOOK

We described the toolset that implements of the OME concept and reported on the feedback that we obtained through semi-structured interviews with practitioners that had experience with process improvement.

The experience that is reported in this paper—which consists of technical mechanisms, design choices, and feedback from the field concerning the OME—constitutes an important source of information for researchers and practitioners

in the field of process improvement. We envisage that this information can be leveraged in order to reuse successful design choices and technologies, and to create awareness of possible limitations and obstacles.

Our future work aims to evolve the OME implementation from a research prototype to a product that can be effectively used in the industry. This will involve identifying which technologies can be reused, and which ones need to be replaced; building a large public method base, also through incentives for organizations to contribute; and integrating our toolset with existing information systems and KM systems, in order to facilitate its adoption.

The evaluation that we performed in the context of this research has a broad scope, yielding high-level results. These results are useful in guiding future research, mainly in the area of MF integration and planning. As we proceed implementing the OME, frequent and thorough evaluation of each subsystem is required in order to alleviate the issues identified in this study.

Part V

CONCLUSIONS AND FUTURE WORK

9

CONCLUSIONS

9.1 CONCLUSIONS AND CONTRIBUTIONS

In this dissertation we have proposed novel techniques to support incremental process improvement, which we expressed in Chapter 1 as the following main research objective:

Design and develop a knowledge-centric software system to support incremental process improvement in software production. — *Main Research Objective*

In order to reach this objective, we have followed a process moving from the exploration of improvement paths within the software industry and the development of fundamental techniques for describing and employing method increments, to the design and realization of the Online Method Engine (OME), which employs the core concepts of our research. Each of these phases relate to one of three sub-objectives defined in Chapter 1. We will use this same structure to discuss the main research results and the contributions described in this dissertation.

Study the practices of software process improvement in the domain of software production. — *Research Objective A*

Our first objective was to investigate Software Process Improvement (SPI) efforts in the software production domain. This phase is essential to understanding process improvement efforts in an industrial setting. The results of this phase form the basis for the development of techniques and tools that support incremental process improvement.

Research Question A.1 *How can the Scrum development method be adapted to the context of SPM processes?*

The challenge of aligning the various processes surrounding software development is present within any software producing organization. Especially between software development teams and software product management teams, where there is a strong need for frequent communication and a strong synchronization of work, misalignment can cause significant problems in terms of productivity and software quality.

In the case of Scrum, which is by design a software development method, we deal with short development cycles and frequent feedback loops within the development teams. However, these loops require a constant input of up-to-date, detailed requirements by the product managemer. At the same time, the product manager needs the input from the development teams in order to refine the requirements and estimate the workload. When their processes are not aligned, these principles become jeopardized.

The first contribution described in this dissertation is a method description for extending the agile Scrum development method to the Software Product Management (SPM) domain. The method description is an adapted version of the standard Scrum model, presented in this form of a Process-Deliverable Diagram (PDD). The method includes provisioning for the gradual breakdown of high level functionality descriptions into smaller items that fit within a development sprint, called the *Requirements Refinery* (see Figure 9.1).

Analysis of 28 SPM sprint backlogs, extracted during a case study at a large software organization, has resulted in two lists of standard backlog items, one regarding development related activities and one related to SPM related activities. This overview can be used as a baseline for applying the Scrum method to SPM.

We address the challenges of aligning the high-level process concerned with software production with the detailed low-level process that describes development based on four interviews and an analysis of internal documents, resulting in five key lessons for aligning the two processes, shown below. These lessons reflect the challenges that were observed during the gradual introduction of the agile SPM approach, thus increasing our understanding of designing a more holistic software development approach:

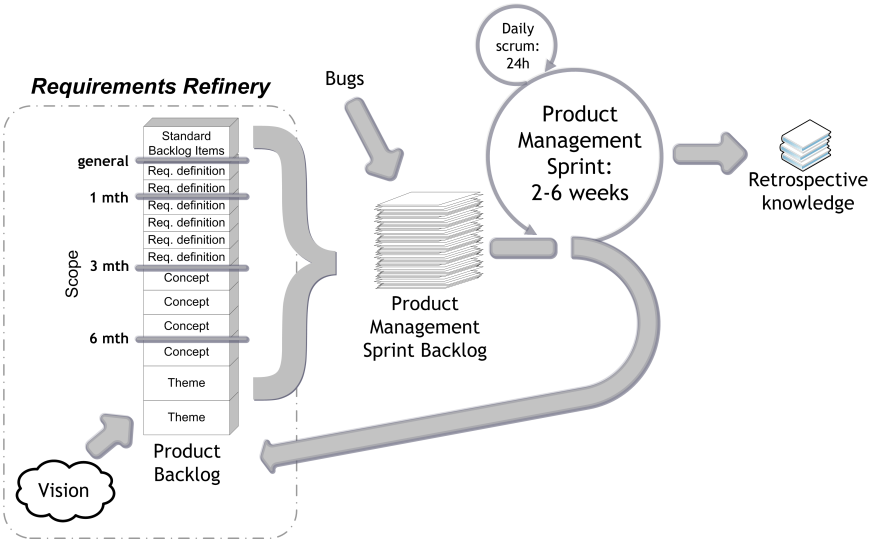


Figure 9.1.: Requirements Refinery

- Alternate sprint cycles for SPM and development ensure that the backlogs are up-to-date.
- Complex requirements are in need of structured detailing.
- Daily Scrum meetings are essential.
- Backlog administration requires discipline.
- Early collaboration promotes reuse and integration.

According to which paths can Scrum be implemented?

Research Question A.2

When introducing new methods, organizations have a choice between a radical change to the new way of working or a gradual move towards the target approach. Radical changes can have benefits in terms of cost and process complexity, but they place a high burden on the employees.

Based on an analysis of four case companies, we identify and illustrate the differences between a gradual and a disruptive approach to the introduction of Scrum within a software producing organization. The results of this analysis provide insight into the differences in motivation, execution, and outcome of the different approaches.

Each case study is described in terms of three phases related to the introduction within the organization: preparation, implementation, and cus-

tomization. For each phase, we describe the most important decisions and their outcomes. Through these case study descriptions, we provide concrete insight into common practices and issues related to method adoption strategy.

We decompose Scrum into a set of method fragments, and link these method fragments to a set of method increments for each case organization. The resulting overview demonstrates the capabilities of using method increments for historical process improvement analysis, and provides an initial understanding of how to use method increments for planning an incremental process improvement effort.

Research Objective B

Develop foundational techniques for tool-supported software process improvement.

The second objective of the research described in this dissertation was the development of techniques that can be used for computer-aided process improvement. The focus of these techniques lay on the support for capturing method increments and using these method increments for the purpose of planning. These techniques are an essential addition to existing approaches for method description, method assembly, and method assessment.

Research Question B.1

How can multiple increments that are part of a larger evolutionary process improvement be documented effectively?

As a pre-requisite for reasoning on method changes, we need a technique for capturing historical data related to these changes, their content, and the obtained effects. For this purpose, we propose a template for capturing relevant historical data related to (series of) atomic method changes in the form of the method increment case description template.

The template is structured around a single, coherent method change. It captures the goal of this change, the atomic steps that are embodied within the method change, the relevant contextual factors, and the effects on the process in terms of changes to the capability maturity. In addition, it enables the users to provide an evaluation of the method change in terms of perceived success or failure, and the reasons for that outcome.

Although the template can be used both in a descriptive as well as a prescriptive manner, the focus lies on the former. It is a means to build up a knowledge base of method changes by coupling context, action, and result.

How can we determine the most appropriate path for implementing process changes, based on the constraints of the stakeholders? — *Research Question B.2*

Although incremental introduction of process improvements can facilitate the organizational change and relieve the burden of employees, it is no trivial tasks to determine an adequate plan for the introduction of complex methods. Organizations need to take into account existing processes, limited resources for process improvement efforts, and both short and long term goals. These factors impact the available options for introducing new processes and work documents.

We propose an algorithm and a method for generating a set of scenarios that describe the step-wise introduction of a set of related method changes. Based on a set of hard (must have) and soft (nice to have) constraints, the algorithm generates both optimal as well as sub-optimal plans. It takes into account dependencies between deliverables, costs for implementing specific changes, and dependencies between capabilities—based on the SPM competence model (Bekkers, van de Weerd, Spruit, et al., 2010).

The main contribution of this approach is the possibility of generating multiple realistic plans, allowing an organization to focus on small changes that deliver value to the organization quickly, while providing the freedom to choose a particular approach that best fits the organizational needs.

Design and develop a system that supports process improvement efforts based on the notion of incremental process improvement. — *Research Objective C*

In Chapter 1, we proposed an integrated design science research approach (see Figure 1.3). The first step of this model, problem identification, is reflected by the third and final objective of this dissertation, i. e. the development of a software system that integrates our knowledge of incremental process improvement into knowledge-centric support tool. Throughout the following sections, we address how the remaining components in that model have been addressed in this research.

What are the objectives for a knowledge centric software system supporting incremental process improvement? — *Research Question C.1*

Throughout the years, several systems have been proposed for the support of method fragment modelling, method discovery (method

bases), method assembly, and process assessment. Unfortunately, such tools are often not widely used. Research shows that practitioners deem them too complex, or that their value is not directly obvious. Van de Weerd, Versendaal, and Brinkkemper (2006) have proposed a proposed a knowledge infrastructure with the aim of supporting product software organizations during process improvement efforts. This Product Software Knowledge Infrastructure (PSKI) forms the basis for the tool proposed in this dissertation, which promotes knowledge sharing and aims to increase benefit by linking method knowledge directly to the organizational context, and by supporting resource-intensive tasks such as improvement planning and method enactment (cf. Figure 1.3 — define the objectives for a solution).

Based on the results from seven case studies, we have identified the seven lessons, shown below, that were used to further elaborate the original PSKI proposal. This has resulted in the conceptual framework for the OME (cf. Figure 1.3 — construct a conceptual framework).

- Process improvement frameworks are often deemed inappropriate due to their size or due to their capability requirements.
- Scientific literature is rarely used directly by practitioners, both due to the fact that it is hard to access and that its language is often complex.
- Most organizations score low on the SPM maturity matrix, indicating the lack of activities that are deemed important by many industry experts.
- Organizational processes are often not documented or outdated; either by choice or by lack of resources.
- Process improvements are often implemented incrementally, often without a long term process improvement planning.
- Many organizations are willing to share detailed information regarding internal processes for the sake of scientific progress and the advancement of the SPM field.
- Process owners are reluctant to trust pure machine-based process improvement suggestions.

What is an adequate functional architecture for a knowledge centric software system supporting incremental process improvement? *Research Question C.2*

The conceptual framework forms the basis for the elaboration of a functional architecture for the OME (cf. Figure 1.3 — develop a system architecture). This functional view describes the major usage paths of the system—i. e. dissemination of method knowledge, assessment and improvement of implemented methods, and the enactment of these improvements—in more detail.

Each functional component of the architecture is elaborated through a PDD. In addition, we demomnstrate potential usage by retrospectively applying the approach to an earlier case study.

What is a feasible technical architecture for a knowledge centric software system supporting incremental process improvement? *Research Question C.3*

We complement the functional architecture for a knowledge-centric SPI system with a technical architecture, that shows how the design can be realized (cf. Figure 1.3 — analyze and design the system). The technical architecture describes technical solutions for the major functionalities.

The functional and technical architecture design are implemented in a prototype of the OME (cf. Figure 1.3 — build the prototype system). The core of this prototype is formed by several concepts and techniques presented in earlier research, including a method base of Method Fragments (MFs), and a Capability Maturity Model for SPM. On top of this baseline, we implement the most important techniques proposed in this dissertation.

We evaluate the prototype during several iterations based on interviews with software product managers (cf. Figure 1.3 — observe and evaluate the system). These evaluation rounds provide input through the design and realization of the prototype.

The prototype is intended to support software producing organizations during process improvement efforts by giving them access to a large collection of MFs and by aiding them in the adaptation and enactment of their processes.

9.2 IMPLICATIONS

9.2.1 *Implications for theory*

The application of existing methods to new contexts is a constant challenge in many organizations. It is rare that methods can be implemented without change in new organizations, due to various factors including existing process, organizational capabilities, and the market sector. Furthermore, methods can often be applied in more ways than originally intended. The application of Scrum to the SPM domain described in Chapter 2 is a clear example of such a case. Although existing literature provides techniques to configure and adapt method changes (Jecker, Janiesch, Knackstedt, et al., 2007; Karlsson and Ågerfalk, 2009), our research on the adaptation of Scrum to the SPM context and the description of Scrum implementation paths provide a better understanding of the factors that play a role in such changes.

The approach has shown to be of interest to researchers and practitioners from various domains and regions. To date, it is one of few proposals described in literature that deal with the topic. The alternated sprints are a partial solution to the misalignment between the evolving release scope of agile development and the need for release planning, as described by Dzamashvili-Fogelström, Numminen, and Barney (2010), by enabling product managers and software development teams to reflect more often and more effectively.

However, the requirements refinery itself is subject to the same implications as other methods. Implementations within other organizations will likely require and induce new adaptations, in line with the specific organizational context. The case studies in Chapter 3 show that there are various approaches to the implementation of Scrum. Although we have in no way described every possible implementation path, the description of revolutionary and gradual implementation paths shows us the necessity of acknowledging this variety. The development of the method increment case description provides us with the means to do so.

By applying automated reasoning techniques to the problem of improvement planning, we show that systems such as the OME can be developed beyond the state of a knowledge repository. Instead, we apply clear semantics to method fragments, method increments and context, enabling the

generation of recommendations within clear and rational constraints. Based on the method described in Chapter 5, the technique can be developed to include more complex constraints, and possibly be applied to other method engineering approaches such as the MAP approach (Ralyté, Maiden, Rolland, et al., 2005) and method components (Wistrand and Karlsson, 2004) as well.

The findings from the various case studies are in line with theory related to process improvement for knowledge work (Davenport, 2010). Product managers often respond negatively to strict processes, and are reluctant to accept changes when they are not involved in the SPI process. With the OME and the underlying process improvement approaches, we enable product managers to become more involved in the SPI process, resulting in the combination of process and practice that is promoted by Davenport (2010). The ability to capture process improvements and their implications on a smaller, more situational scale allows us to moves away from the prescriptive model that many traditional approaches, such as CMMI (CMMI Product Team, 2010) employ. Instead, it forces an analytical, reflective stance.

9.2.2 *Implications for practice*

The further elaboration of the method increment concept and case description allows for better re-use of experience with process improvement efforts, by allowing the recording of the results from applying specific method fragments in specific contexts. During the case studies performed in the context of this dissertation, one apparent benefit was the acitivity of retrospectively analyzing the SPM and development processes. This activity forced practitioners to pause and reflect on the changes that their organizations had applied during the previous few years.

Although the approach described in this dissertation is lightweight and provides quick results, the potential benefit of building a situational knowledge base, where MFs are linked to Situational Factors (SFs) and the positive and negative outcomes, is one that needs to fully prove itself yet. The architecture of the OME has been designed to be independent of the application domain. However, every domain does need its own tools for e. g. process assessment and enactment. For the elaboration of the OME,

we have always focused on the SPM domain. While there is quite some methodical knowledge available in the research and practitioners community, this is a domain with hardly any formal education and relatively little knowledge sharing within the community. The OME and the underlying techniques have the potential to help organizations find and apply the relevant knowledge in a rapidly changing environment. The OME enables broader use of the Situational Assessment Method (SAM) as developed by Bekkers, Spruit, van de Weerd, et al. (2010). This approach has been applied in a wide variety of cases, providing a thorough assessment of SPM processes at little cost.

During the past years, we have constantly used the knowledge gained from our work on SPM to educate product managers, through a bi-yearly course for professionals. This course consists of 10 sessions and deals with various topics such as requirements management, product planning, strategic management, and marketing. Nearly 150 professionals have completed the course.

9.3 LIMITATIONS AND FUTURE WORK

Challenges for the OME — The OME represents our vision of situational method engineering applied to practice. It entails an interactive knowledge infrastructure, allowing organizations to effectively measure their current process maturity, and obtain and implement process improvements in an incremental fashion. The core of the system is the method base, appended with modules for knowledge dissemination, method assessment, method improvement, and method enactment.

For the OME to become fully functional, we need additional research in several areas. In its current iteration, the aspects related to the method base and process assessment have been implemented. We are also able to link assessment results to method fragment proposals. In addition, we have proposed techniques in this dissertation related to the planning of improvement steps and the enactment of method increments. However, these last two aspects are not fully understood, recquiring more design cycles and additional evaluation of the proposed techniques. A more practical concern is the filling of the method base using scientific literature and case study results. This work is currently underway.

Usability of CAME tools — Interest in Computer/Aided Software Engineering (CASE) and Computer/Aided Method Engineering (CAME) tools has grown because of the important role they play in supporting the software development process. Studies show these complex and sophisticated tools have a positive impact on quality and productivity but they have been slow to be adopted by industry; this is partially explained by the difficulty of learning to use the tool. Previous research has shown that the ability to effectively use a CASE or CAME tool depends on a multitude of factors, including personal style, motivation, and the design of the tool. More research is needed regarding the usability and learnability of CAME tools, including the OME.

Process improvement for knowledge work — The main application domain of this dissertation research has been SPM. The activities within this domain are often of an abstract and creative nature, such as the translation of business strategy and customer requests into a product vision. This type of work is often called knowledge-work. Davenport (2010, p. 1) suggests that "that traditional, engineering-based approaches to knowledge work are incompatible with the autonomy and work approaches of many knowledge workers". In line with this statement, we argue that the higher the abstraction level of a job, the harder it becomes to effectively apply existing method engineering techniques to them. Unfortunately, method engineering literature insufficiently deals with processes related to knowledge work.

Method engineering techniques describe how to construct a situational method, specific to the context of one organization. In most cases, the approach focusses on the method to-be and not on the method as-is, except for an assessment of the current method 'maturity'. This ignorance regarding the existing situation can be a major inhibitor of process improvement Davenport (2010, p. 1). The current method needs to be the starting point of process improvement efforts, not only in terms of maturity, but also in terms of activities, deliverables and capabilities.

Concepts related to quality, efficiency, and success become increasingly more open to interpretation and therefore increasingly hard to measure and control when the level of abstraction increases. More research is needed regarding the relation between tasks within the SPM and software development domains in terms of the level of abstraction that they work on

and the level of creativity involved. The results should then be related to measurability, controllability, and learnability.

Giving meaning to fragments — Retired US Marines officer Paul van Riper has been attributed with the quote: "In the act of tearing something apart, you lose its meaning" (Gladwell, 2007, p. 125). The quote was obviously never uttered in the context of method engineering, but it does represent an actual problem. When methods are broken up into smaller fragments, they might lose their effectiveness and efficacy. This would imply that there is a limit to breaking up methods. The concepts of method increments (van de Weerd, Brinkkemper, and Versendaal, 2007) and method rationale (Rossi and Tolvanen, 2000) have the potential to preserve the meaning of even these small parts, but building up this knowledge is hard.

The purpose of this dissertation research has been to provide additional concepts and techniques that enable us to structure and enhance the conduct of process improvement. We have enhanced the method engineering domain with techniques that enable the description and prescription of sub-steps throughout a process improvement effort, incorporating not only knowledge about changed activities, deliverables and roles, but also experience related to organizational culture, context-specific contingency factors, and other intangible knowledge. Even though the potential usage of method increments has been identified, we have not yet investigated all its facets. For instance, what is the exact scope of an increment? At what point does an increment become too big for effective implementation, and when is it too small? Such questions, related to the concepts of modularization and coherence (Ågerfalk, Brinkkemper, Gonzalez-Perez, et al., 2007), arise when trying to apply the concept to real-life situations. One can answer these questions by observing the steps that software companies take during process improvement projects. Under the assumption that method increments should be implementable and are therefore similar in size to the steps that organizations take while improving their process, the characteristics of a method increment can be derived from the characteristics of such steps.

Measuring process improvement success — A possible approach to extracting knowledge regarding the applicability of specific methods is to measure process retention. Organization often returning to their old habits

after a process improvement project has ended. Not all process improvements 'stick', possibly indicating that the ones that do are successful improvements, contributing to the company's well-being. One should take a very critical stance when applying this approach, as the measurement leaves room for many interprations. Long process retention can have many reasons, ranging from a strong top-down management push of an ineffective process to a strong fit with the business unit's culture. On the other hand, short process retention does not have only one explanation either. Some possible reasons are quick growth of the business unit, reorganization, and dissatisfaction with the process change.

Despite these reasons, process retention has the potential to be an interesting success measure. Although it will not give definitive answers, it might be a good indicator for certain issues within a company or business unit. For instance, strong management push is not always easy to measure due to social reasons. However, a low process retention along with a low process satisfaction rating might be able to pinpoint the problem nonetheless.

Having a reliable success measure is an important step in the validation of incremental method evolution research. Without knowing the effect of applied process improvements, there is no way of telling if method increments are an effective means to process improvement or not.

Part VI

APPENDIX

A

DEMONSTRATION OF THE ONLINE METHOD ENGINE

During the past decade, much research has been performed in the areas of Method Engineering (ME) and Software Process Improvement (SPI). As a result of this research, we are developing the Online Method Engine (OME). The OME is a Knowledge Management (KM) system that provides support during process improvement initiatives, using a set of assessments, an extensive method base, and automatic method assembly mechanisms. The system is designed around four main acivities: knowledge dissemination, method assessment, method improvement, and method enactment. In this paper, we demonstrate the core components and the main scenarios for the usage of the OME.

This work was originally published as:

Vlaanderen, K., Spruit, S., Dalpiaz, F., et al. (2014). "Demonstration of the Online Method Engine." In: *Proceedings of the International Conference on Advanced Information Systems Engineering Forum (accepted for publication)*. Ed. by Nurcan, S. and Pimenidis, E. Thessaloniki, Greece

A.1 INTRODUCTION

In an attempt to describe and reuse software development practices, several researchers have proposed method bases to capture this knowledge (Jarke, Gallersdörfer, Jeusfeld, et al., 1995; Ralyté, 1999). The method fragments in these method bases are being configured and re-engineered using several techniques (Nehan and Deneckère, 2007; Ralyté, Rolland, and Plihon, 1999), and implemented through a variety of tools (Karlsson and Ågerfalk, 2012). Some of these techniques are in turn applied to the Method Engineering (ME) activity itself (Hug, Front, and Rieu, 2008).

The Online Method Engine (OME) is the implementation of scientific concepts in the field of ME and Software Process Improvement (SPI), in combination with concepts from the field of Knowledge Management (KM). The main goal of the OME is to provide a system that facilitates the sharing of knowledge related to method fragments for software development, assessing methods that are being used in practice, improving methods based on assessment results, and adopting (improved) methods in practice. The main philosophy behind the OME is that incremental (or step-wise) improvement of processes reduces risks related to process change and improves the change of success (Baddoo, 2003; Diaz and Sligo, 1997; Pino, Pedreira, García, et al., 2010).

Following the four goals of the OME, the system is based on four layers of functionality; knowledge dissemination, method assessment, method improvement, and method enactment. Each layer increasingly relies on the previous ones, and all of the layers rely on a central method base, which forms the backbone of the system. The quality of the system is ensured by a constant process of validation and review.

A.2 ARCHITECTURE

Traditionally, method bases contain fragments of method knowledge, which reside on the M2 level according to the Meta/Object Facility (MOF) framework (Object Management Group, 2011). In the context of the OME, the method base contains some additional elements. In the first place, it contains a set of Situational Factors (SFs) that can be used to describe an organizational unit. These SFs describe the specifics of the organizational context in terms that are unrelated to the specific

method implementation. The SFs can be used for other purposes, but this will be described later on.

Complimentary to the SFs, the method base contains a set of capabilities that can be used to characterize a specific method implementation in a specific domain. The capabilities relate directly to a domain model, which captures the relevant process areas in a specific domain.

The SFs and the capabilities are related through a domain-dependent mechanism called the Situational Factors Effect (SFE). A SFE describes how a specific value of a situational indicator typically influences the relevance of a capability (Bekkers, Spruit, van de Weerd, et al., 2010). These SFEs are also captured in the method base.

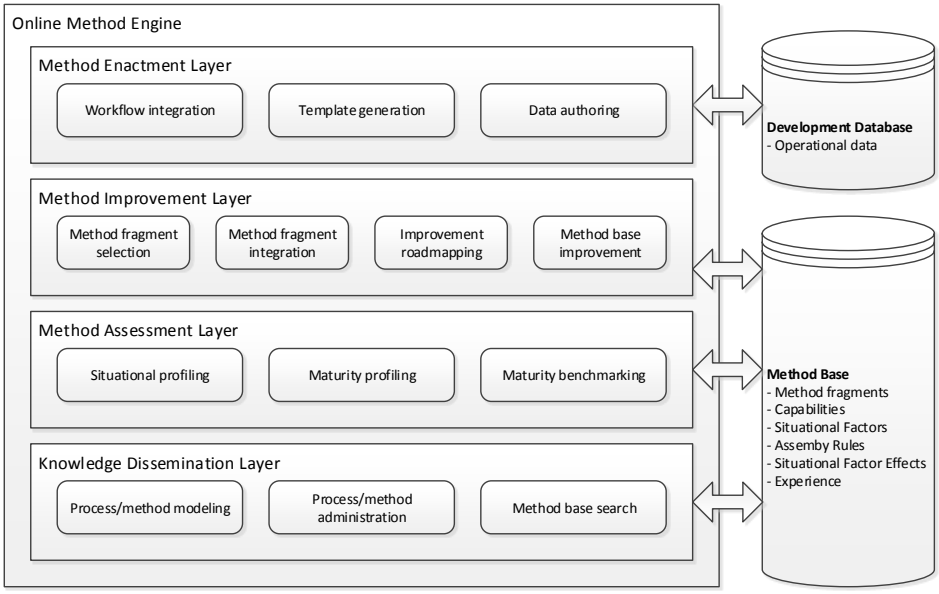


Figure A.1.: Functional architecture of the OME

The final core element of the method base consists of a set of Method Fragments (MFs). MFs describe a coherent piece of method knowledge that facilitates reaching a specific goal during software development. A MF is described in terms of activities and deliverables. Within the OME, MFs are currently modeled using Process-Deliverable Diagram (PDD) (van de Weerd and Brinkkemper, 2008),

which are a combination of a Unified Modeling Language (UML) activity diagram and a UML class diagram, connected through a set of links.

A.3 SCENARIOS

A.3.1 *Scenario A: Method Assessment*

Maturity	A	B	C	D	E	F
Requirements management						
Requirements gathering	Implemented	Implemented	Missing	Missing	Missing	Irrelevant
Requirements identification	Implemented	Implemented	Missing	Irrelevant		
Requirements organizing	Implemented	Missing	Missing			
Release planning						
Requirements prioritization	Implemented	Implemented	Irrelevant	Implemented	Missing	
Release definition	Implemented	Missing	Missing	Irrelevant	Irrelevant	
Release definition validation	Missing	Missing	Irrelevant			
Scope change management	Implemented	Missing	Irrelevant	Irrelevant		
Build validation	Implemented	Implemented	Missing			
Launch preparation	Implemented	Implemented	Implemented	Irrelevant	Missing	Missing
Product planning						
Road map intelligence	Implemented	Implemented	Implemented	Missing	Missing	
Core asset roadmapping	Implemented	Implemented	Missing	Irrelevant		
Product roadmapping	Implemented	Implemented	Missing	Irrelevant	Missing	
Portfolio management						
Market analysis	Implemented	Implemented	Implemented	Implemented	Missing	
Partnering & contracting	Implemented	Implemented	Implemented	Missing	Irrelevant	
Product life cycle management	Implemented	Missing	Missing	Irrelevant	Irrelevant	

Figure A.2.: Example areas of improvement report

In general, a process improvement effort starts with an assessment of the current processes. The assessment approach employed within the OME is based on the Situational Assessment Method (SAM) (Bekkers, Spruit, van de Weerd, et al., 2010). This method consists of three phases; data collection, calculcation, and feedback. Data collection is performed through two questionnaires; one to determine the organizational context, and one to determine the current capabilities of the organization. During the calculcation phase, the former results are transformed into a

Current Capability Profile (CCP) and the latter into an Optimal Capability Profile (OCP). The delta between these two profiles results in an Areas of Improvement Matrix (AIM). During the feedback phase, an evaluation is performed that is used to improve the quality of the knowledge base (i. e. the method base).

The SAM is realized through two forms within the OME. The first form is used to capture the organizational context. It consist of 24 questions spread out over 5 pages. Each question has a short description, a set of answers, and possibly some help text to indicate the type of answer expected. The second form is used to capture the current capabilities. This form contains 68 questions, which are spread out over 4 pages. Each page represents a business function, i.e. a layer from the Software Product Management (SPM) competence model (van de Weerd, Brinkkemper, Nieuwenhuis, et al., 2006b). The results of the questionnaires can be reviewed using two seperate reports; a situational profile report and a capability maturity report. The combined results are shown through an areas of improvement report, which shows both a condensed as well as an expanded version of the AIM. An example of the condensed *areas of improvement* report is shown in Figure A.2.

A.3.2 *Scenario B: Method Discovery*

The results of the method assessment activity indicate the areas within the current process that are open for improvement. This is done in the form of a set of capabilities that are divided over various process areas, such as requirements prioritization or product roadmapping, and ordered based on an associated maturity level. This makes it possible to match these missing capabilities, or a subset of them, to existing methods that are stored in the method base.

The OME facilitates method discovery by providing tools to search the method base based on several filters. These filters include the focus area, the business function, the relevant capabilities, and keywords. The result is a set of MFs that fit these filters. Each result can be expanded into a detailed description of the MF. The main components of these descriptions are a textual summary of the MF, a detailed description that can contain both text and illustrations, a list of the relevant SFs and capabilities, a PDD, descriptions of the main activities and deliverables, and a list of reference. An example excerpt of a method description for the Echo approach Lee, Guadagno, and Jia, 2003 is shown in Figure A.3.

Within the OME, the user is aided in this discovery process as much as possible. Elements within the PDD and items within the situational factor and capability

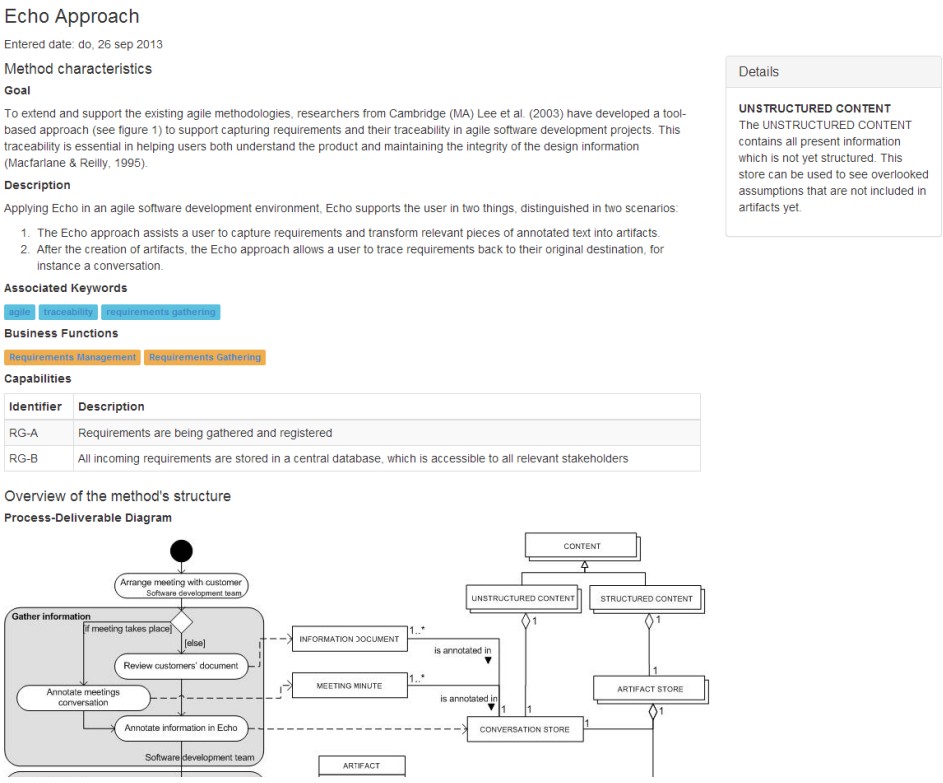


Figure A.3.: Example method fragment description

lists are hyperlinks to more detailed descriptions. This makes it possible to obtain a quick overview, while allowing the user to go into more depth or to find related MFs.

A.3.3 *Scenario C: Method Improvement*

An important goal of the OME is to assist during a process improvement effort. In many cases, there are various solutions available to solve a specific problem, such as requirements prioritization. Not all of these solutions are applicable to any given situation, and it is hard for the method engineer to determine which solutions are likely optimal. The knowledge available within the method base allows for an automated approach to selecting appropriate MFs. This selection can be based

on the structural aspects of the MF (activities, deliverables), required capabilities, and situational factors.

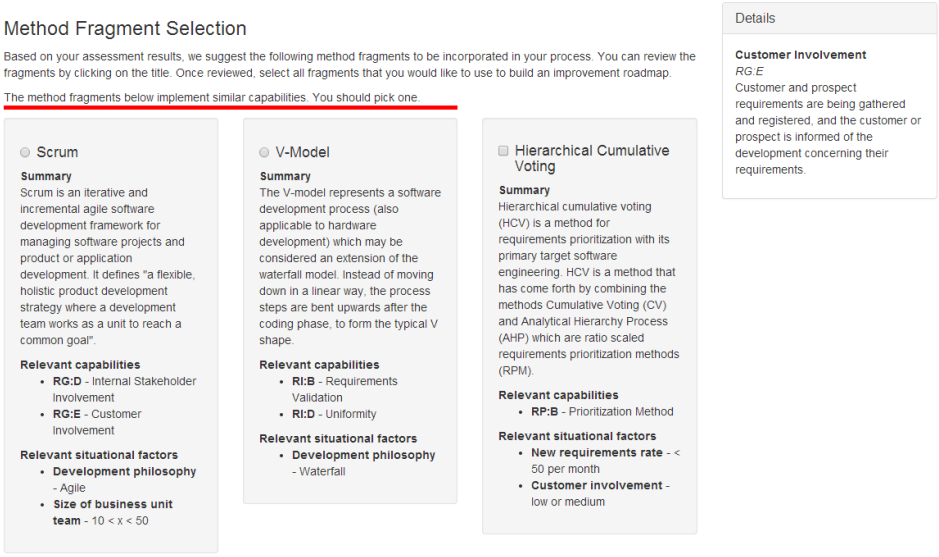


Figure A.4.: Method fragment selection

The OME can propose a set of relevant MFs based on the requirements of the user. It is possible to select MFs that focus on one or more specific business functions, that implement capabilities up to a certain maturity level, or for the entire process. The selected MFs are presented to the user so that they can be reviewed using the tools described above (under method discovery). A prototype of the selection step is shown in Figure A.4.

Once the user has selected the MFs that he deems relevant, these fragments need to be assembled. In most cases, this means that they need to be integrated within the existing process. This activity can be partially performed automatically based on the structural aspects of the MFs. In many cases, conflicts will arise during this assembly. For instance, multiple MFs can include similar or incompatible deliverables. A report of these issues is presented to the user for further review.

The overarching goal of the OME is to allow for incremental improvement. This is a very important characteristic, as the implementation of many process changes at the same time is often unrealistic and unfeasible within an organization. Process changes are always prone to resistance among employees, unforeseen

complications, and changes within the environment. Therefore, the system generates a series of implementation plans (see Chapter 5). These plans consist of a series of increments. Each increment consists of a set of small changes, such as inclusion or removal of activities and deliverables. Plans are generated based on a set of parameters, including the available resources and temporal constraints such as the need for a certain capability to be implemented within a certain amount of increments.

The generated plans are presented to the user as a set of timelines. Each timeline contains the proposed increments with a summary of their contents. Once again. the user can review these changes to gain a detailed understanding of their contents and impact (see Figure A.5 for an example of these timelines).

A.3.4 *Scenario D: Method Enactment*

The final activity within a process improvement effort, apart from the review, is the enactment of process changes. Although this is an activity that is mainly dependend on social and managerial aspects, it is possible to support parts of it through

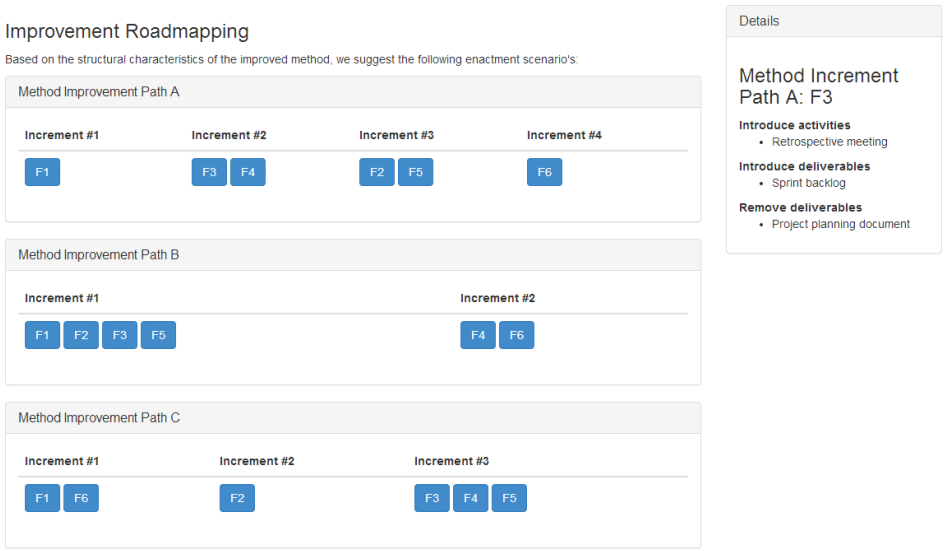


Figure A.5.: Improvement roadmapping

automated means. Within the context of the OME, enactment support focuses on the generation of templates and the automated migration of development tools.

A.3.5 *Scenario E: Method Administration*

All functionality within the OME is based on the contents of the method base. This method base consists of MFs, capabilities, SFs, SFEs, and experience reports. For the creation of MFs, we employ specialized tools instead of developing functionality within the OME itself. MetaEdit+ (Smolander and Lyytinen, 1991) is used to model the PDDs, which can be annotated with relevant capabilities. Textual descriptions can be created with appropriate textual editors. References are stored in the BibTex format.

A.4 DISCUSSION AND FUTURE RESEARCH

Both the conceptual design as well as the initial prototype of the OME have been validated in earlier studies (see Chapter 6–Chapter 8). In its current iteration, the OME does not fully support all of the described scenarios. Development follows the layers as described in Figure A.1 from bottom to top. The method base has been adequately implemented, as is functionality related to method discovery. The method assessment layer has also been realized, making it possible to assess current methods and link the results to MF proposals.

On the method improvement layer, we have realized partial planning functionality. This makes it possible to generate a set of improvement plans based on a goal method. However, more research is needed to incorporate the removal and replacement of MFs, and to support more complex situations including improvement based on an existing method.

For the method enactment layer, no functionality has been implemented sofar. Techniques are currently under development to translate method changes into concrete enactment actions.

BIBLIOGRAPHY

Aaen, I. et al. (2001). "A Conceptual MAP of Software Process Improvement." In: *Scandinavian Journal of Information Systems* 13.1, pp. 81–101.

Abrahamsson, P. et al. (2003). "New directions on agile methods: a comparative analysis." In: *Proceedings of the International Conference on Software Engineering*. IEEE, pp. 244–254.

Ågerfalk, P. J. et al. (2007). "Modularization Constructs in Method Engineering: Towards Common Ground?" In: *Situational Method Engineering: Fundamentals and Experiences* 244, pp. 359–368.

Aharoni, A. and Reinhartz-Berger, I. (2011). "Semi-Automatic Composition of Situational Methods." In: *Journal of Database Management* 22.4.

Alliance, O. (2003). *OSGI Service Platform*. Release 3. IOS Press.

Ashrafi, N. (2003). "The Impact of Software Process Improvement on Quality: in Theory and practice." In: *Information and Management* 40, pp. 677–690.

Automotive Special Interest Group (2010). "Automotive SPICE Process Assessment Model." In: *Final Release, v4* 4, p. 146.

Baddoo, N. (2003). "De-Motivators for Software Process Improvement: an Analysis of Practitioners' Views." In: *Journal of Systems and Software* 66, pp. 23–33.

Bajec, M. et al. (2007). "Software Process Improvement Based on the Method Engineering Principles." In: *Situational Method Engineering: Fundamentals and Experiences*. Vol. 244. Springer, pp. 283–297.

Barney, J., Wright, M., and Ketchen, D. J. (2001). "The Resource-Based View of the Firm: Ten Years After 1991." In: *Journal of management* 27.6, pp. 625–641.

Basili, V. R. (1993). "The Experience Factory and its Relationship to Other Improvement Paradigms." In: *Proceedings of the European Software Engineering Conference*, pp. 68–83.

Basili, V. R. and Green, S. (1994). "Software Evolution at the SEL." In: *Ieee Software* 11.4, pp. 58–66.

Baskerville, R. L. and Pries-Heje, J. (1999). "Knowledge Capability and Maturity in Software Management." In: *ACM SIGMIS Database* 30.2, pp. 26–43.

Bass, L., Clements, P., and Kazman, R. (2012). *Software Architecture in Practice*. SEI Series in Software Engineering. Pearson Education.

Beck, K. (1999). *Extreme programming explained: embrace change*. Boston, USA: Addison-Wesley Longman Publishing Co., Inc.

Beck, K. et al. (2001). *Agile Manifesto*.

Becker, J., Janiesch, C., and Pfeiffer, D. (2007). "Reuse Mechanisms in Situational Method Engineering." In: *Situational Method Engineering: Fundamentals and Experiences*. Vol. 244, pp. 79–93.

Becker, J. and Knackstedt, R. (2007). "Configurative Method Engineering–on the Applicability of Reference Modeling Mechanisms in Method Engineering." In: *Proceedings of the America Conference on Information Systems*, pp. 1–12.

Bekkers, W. et al. (2008). "The Influence of Situational Factors in Software Product Management: An Empirical Study." In: *Proceedings of the International Workshop on Software Product Management*. Washington, DC, USA: IEEE Computer Society, pp. 41–48.

Bekkers, W. et al. (2010). "A Framework for Process Improvement in Software Product Management." In: *Proceedings of the European Conference on Software Process Improvement (EuroSPI)*, pp. 1–12.

Bekkers, W. et al. (2010). "A Situational Assessment Method for Software Product Management." In: *Proceedings of the European Conference on Information Systems*. Ed. by Alexander, T., Turpin, M., and Deventer, J. van. Pretoria, South-Africa, pp. 22–34.

Bekkers, W. et al. (2012). "Evaluating the Software Product Management Maturity Matrix." In: *International Requirements Engineering Conference*. IEEE, pp. 51–60.

Berander, P. (2007). "Evolving Prioritization for Software Product Management." PhD thesis. Ronneby: Blekinge Institute of Technology, p. 250.

Berki, E. (2003). "Formal Metamodelling and Agile Method Engineering in Meta-CASE and CAME Tool Environments." In: *Proceedings of the 1st South-East European Workshop on Formal Methods*. Ed. by Tigka, K. and Kefalas, P. Thessaloniki, Greece, pp. 170–188.

Berkum, M. van, Brinkkemper, S., and Meyer, A. (2004). "A Combined Runtime Environment and Web-Based Development Environment." In: *Proceedings of the International Conference on Advanced Information Systems Engineering*, pp. 307–321.

Berntsson Svensson, R., Olsson, T., and Regnell, B. (2008). "Introducing Support for Release Planning of Quality Requirements—An Industrial Evaluation of the QUPER Model." In: *Proceedings of the International Workshop on Software*

Product Management. Ed. by Ebert, C. et al. Barcelona, Spain: IEEE, pp. 18–26.

Bettis, R. A. and Hitt, M. A. (1995). "The New Competitive Landscape." In: *Strategic management journal* 16.S1, pp. 7–19.

Bin Basri, S. and O'Connor, R. V. (2010). "Evaluation on Knowledge Management Process In Very Small Software Companies : A Survey." In: *Proceedings of the Knowledge Management International Conference*, pp. 1–6.

– (2011). "Knowledge Management in Software Process Improvement : A case study of very small entities." In: *Knowledge Engineering for Software Development Life Cycles: Support Technologies and Applications*. UK: IGI Global, pp. 1–12.

Bjornson, F. and Dingsøyr, T. (2008). "Knowledge Management in Software Engineering: a Systematic Review of Studied Concepts, Findings and Research Methods Used." In: *Information and Software Technology* 50.11, pp. 1055–1068.

Boehm, B. (2006). "A View of 20th and 21st Century software Engineering." In: *Proceeding of the 28th international conference on Software engineering - ICSE '06*, p. 12.

Boehm, B. and Turner, R. (2004). *Balancing Agility and Discipline: a Guide for the Perplexed*. Boston, MA, USA: Pearson Education, p. 194.

– (2005). "Management Challenges to Implementing Agile Processes in Traditional Development Organizations." In: *IEEE Software* 22.5, pp. 30–39.

Booch, G. (1995). *Object Solutions: Managing the Object-Oriented Project*. Addison-Wesley.

Booch, G., Rumbaugh, J., and Jacobson, I. (1999). *The Unified Modeling Language User Guide*. MA: Addison-Wesley, Reading.

Borjesson, A. and Mathiassen, L. (2004). "Successful Process Implementation." In: *IEEE Software* 21.4, pp. 36–44.

Brinkkemper, S. (1996). "Method Engineering: Engineering of Information Systems Development Methods and Tools." In: *Information and Software Technology* 38.4, pp. 275–280.

Brinkkemper, S. and Harmsen, F. (1995). "Configuration of Situational Process Models: an Information Systems Engineering Perspective." In: *Software Process Technology* 913, pp. 193–196.

Brinkkemper, S., Saeki, M., and Harmsen, F. (1998). "Assembly Techniques for Method Engineering." In: *Advanced Information Systems Engineering*. Vol. 1413, pp. 381–400.

Brinkkemper, S., Saeki, M., and Harmsen, F. (1999). “Meta-Modelling Based Assembly Techniques for Situational Method Engineering.” In: *Information Systems* 24.3, pp. 209–228.

Brinkkemper, S. et al. (2008). “Process Improvement in Requirements Management: a Method Engineering Approach.” In: *Proceedings of Requirements Engineering: Foundation of Software Quality*.

Brocke, J. vom and Sinnl, T. (2011). “Culture in business process management: a literature review.” In: *Business Process Management Journal* 17.2, pp. 357–378.

Brooks, F. P. (1995). *The Mythical Man-month: Essays on Software Engineering*. Essays on software engineering. Addison-Wesley.

Carlshamre, P. and Regnell, B. (2000). “Requirements Lifecycle Management and Release Planning in Market-Driven Requirements Engineering Processes.” In: *Proceedings of the International Workshop on Database and Expert Systems Applications*. Ed. by Ibrahim, M. T., Küng, J., and Revell, N. London, UK: IEEE, p. 961.

Carlshamre, P. et al. (2001). “An Industrial Survey of Requirements Interdependencies in Software Product Release Planning.” In: *Proceedings of the IEEE International Symposium on Requirements Engineering*. Ed. by Nuseibeh, B. Toronto, Canada: IEEE, p. 84.

Carmel, E. and Becker, S. (1995). “A Process Model for Packaged Software Development.” In: *IEEE Transactions on Engineering Management* 42.1, pp. 50–61.

Cass, A. and Volcker, C. (2000). “SpiCE for SPACE: a Method of Process Assessment for Space Projects.” In: *SPICE 2000 Conference*.

Churchman, C. W. (1967). “Guest Editorial: Wicked Problems.” In: *Management Science* 14.4.

CMMI Product Team (2002). *Capability Maturity Model Integration (CMMI)*. Tech. rep. December 2001. Pittsburgh: Carnegie Mellon Software Engineering Institute, p. 647.

– (2006). *CMMI ® for Development , Version 1 . 2*. Tech. rep. August.

– (2010). *CMMI® for Development, Version 1.3 CMMI-DEV, V1.3*. Tech. rep. November. Pittsburgh, p. 468.

Cockburn, A. (1997). “Structuring Use Cases with Goals.” In: *Technology* 84121.801, pp. 1–13.

Cohn, M. and Ford, D. (2003). “Introducing an Agile Process to an Organization.” In: *Computer* 36.6, pp. 74–78.

Conboy, K. (2009). “Agility from First Principles: Reconstructing the Concept of Agility in Information Systems Development.” In: *Information Systems Research* 20.3, pp. 329–354.

Cossentino, M. et al. (2006). “A Metamodelling Based Approach for Method Fragment Comparison.” In: *Proceedings of the 11th International Workshop on Exploring Modeling Methods in Systems Analysis and Design (EMMSAD06)*.

Cusumano, M. (2008). “The Changing Software Business: Moving from Products to Services.” In: *Computer* January, pp. 20–27.

Cusumano, M. (2004). *The Business of Software: What Every Manager, Programmer, and Entrepreneur Must Know to Thrive and Survive in Good Times and Bad*. New York, NY, USA: The Free Press, p. 352.

Danait, A. (2005). “Agile Offshore Techniques - a Case Study.” In: *Procedings of the Agile Conference*, pp. 214–217.

Davenport, T. H. (1993). *Process Innovation: Reengineering Work Through Information Technology*. New York, NY, USA: Ernst & Young.

Davenport, T. H. (2010). “Process Management for Knowledge Work.” In: *Handbook on Business Process Management 1*. Ed. by Brocke, J. vom and Rosemann, M. 2005. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 17–35.

Davenport, T. H. and Prusak, L. (1998). *Working Knowledge: How Organizations Manage What They Know*. 1st. Boston, MA, USA: Harvard Business Press, p. 199.

Deming, W. E. (2000). *Out of the Crisis*. MIT Press, p. 523.

Deneckère, R. et al. (2008). “From Method Fragments to Method Services.” In: *Proceedings of EMMSAD’08*. Ed. by Halpin, T., Proper, H., and Krogstie, J. Montpellier, France.

Diaz, M. and Sligo, J. (1997). “How Software Process Improvement Helped Motorola.” In: *Software, IEEE* 14.5, pp. 75–81.

Dingsøyr, T. et al. (2006). “Developing Software with Scrum in a Small Cross-Organizational Project.” In: *Lecture notes in computer science*. Springer, pp. 5–15.

Dorling, A. (1993). “SPICE: Software Process Improvement and Capability Determination.” In: *Software Quality Journal* 2.4, pp. 209–224.

Dybå, T. and Dingsøyr, T. (2008). “Empirical Studies of Agile Software Development: a Systematic Review.” In: *Information and Software Technology* 50.9-10, pp. 833–859.

Dzamashvili-Fogelström, N., Numminen, E., and Barney, S. (2010). "Using Portfolio Theory to Support Requirements Selection Decisions." In: *Proceedings of the International Workshop on Software Product Management*.

Earl, M. (2001). "Knowledge Management Strategies: Toward a Taxonomy." In: *Journal of Management Information Systems* 18.1, pp. 215–233.

Ebert, C. (2007). "The Impacts of Software Product Management." In: *Journal of Systems and Software* 6.80, pp. 850–861.

El Emam, K. (1997). *SPICE: The Theory and Aractice of Software Process Improvement and Capability Determination*. Los Alamitos, CA, USA: IEEE Computer Society Press.

Feiler, P. and Humphrey, W. (1993). "Software Process Development and Enactment: Concepts and Definitions." In: *International Conference on the Software Process*. Pittsburgh, PA: Software Engineering Institute.

Fitzgerald, B., Hartnett, G., and Conboy, K. (2006). "Customising Agile Methods to Software Practices at Intel Shannon." In: *European Journal of Information Systems* 15.2, pp. 200–213.

Fricker, S. and Glinz, M. (2010). "Comparison of Requirements {Hand-Off}, Analysis, and Negotiation: Case Study." In: *Proceedings of the International Conference on Requirements Engineering*, pp. 167–176.

Fricker, S. et al. (2010). "Handshaking with Implementation Proposals: Negotiating Requirements Understanding." In: *IEEE Software* 27.2, pp. 72–80.

García, J. et al. (2011). "Design Guidelines for Software Processes Knowledge Repository Development." In: *Information and Software Technology* 53, pp. 834–850.

Gerevini, A. and Long, D. (2005). *Plan Constraints and Preferences in {PDDL3} - the Language of the Fifth International Planning Competition*. Tech. rep. University of Brescia.

Ghallab, M., Nau, D., and Traverso, P. (2004). *Automated Planning: Theory & Practice*. Elsevier.

Giarratana, M. S. and Fosfuri, a. (2007). "Product Strategies and Survival in Schumpeterian Environments: Evidence from the US Security Software Industry." In: *Organization Studies* 28.6, pp. 909–929.

Gladwell, M. (2007). *Blink: The Power of Thinking Without Thinking*. Back Bay Books, p. 296.

Glazer, H. et al. (2008). "CMMi or Agile: Why Not Embrace Both!" In:

Gonzalez-Perez, C. and Henderson-Sellers, B. (2007). “Modelling Software Development Methodologies: a Conceptual Foundation.” In: *Journal of Systems and Software* 80.11, pp. 1778–1796.

– (2008). “A Work Product Pool Approach to Methodology Specification and Enactment.” In: *Journal of Systems and Software* 81.8, pp. 1288–1305.

Gorchels, L. (2000). *The Product Manager’s Handbook: The Complete Product Management Resource (2nd edition)*. 2nd. Columbus, OH, USA: McGraw-Hill, p. 304.

Gorschek, T. and Wohlin, C. (2006). “Requirements abstraction model.” In: *Requirements Engineering* 11.1, pp. 79–101.

Greer, D. and Ruhe, G. (2004). “Software Release Planning: an Evolutionary and Iterative Approach.” In: *Information and Software Technology* 46.4, pp. 243–253.

Gries, D. (1981). *The Science of Programming*. Monographs in Computer Science Series. Springer-Verlag.

Grundy, J. C. and Venable, J. R. (1996). “Towards an Integrated Environment for Method Engineering.” In: *Proceedings of the IFIP WG 8.1 Conference on Method Engineering*. Chapman and Hall, pp. 45–62.

Gupta, D. and Dwivedi, R. (2012). “Method Configuration from Situational Method Engineering.” In: *ACM SIGSOFT Software Engineering Notes* 37.3.

Gupta, V. et al. (2013). “Requirement Reprioritization: A Multilayered Dynamic Approach.” In: *International Journal of Software Engineering and Its Applications* 7.5, pp. 55–64.

Guzélian, G. and Cauvet, C. (2007). “SO2M: Towards a Service-Oriented Approach for Method Engineering.” In: *Proceedings of the International Conference in Computer Science*.

Harmsen, F. (1997). “Situational Method Engineering.” PhD thesis. Universiteit Twente.

Harmsen, F. and Brinkkemper, S. (1995). “Design and Implementation of a Method Base Management System for a Situational CASE Environment.” In: *Proceedings of the Second Asia-Pacific Software Engineering Conference (APSEC’95)*. Washington, DC, USA: IEEE Computer Society, p. 430.

Harmsen, F., Brinkkemper, S., and Oei, J. L. H. (1994). “Situational Method Engineering for Informational System Project Approaches.” In: *Proceedings of the IFIP WG8.1 Working Conference on Methods and Associated Tools for the Information Systems Life Cycle*. New York, NY, USA: Elsevier Science Inc., pp. 169–194.

Harter, D. (2000). “Effects of Process Maturity on Quality, Cycle Time, and Effort in Software Product Development.” In: *Management Science* 46.4, pp. 451–466.

Henderson-Sellers, B. (2002). “Process Metamodelling and Process Construction: Examples Using the OPEN Process Framework (OPF).” In: *Annals of Software Engineering* 14.1, pp. 341–362.

– (2003). “Method Engineering for OO Systems Development.” In: *Communications of the {ACM}* 46.10, pp. 73–78.

Henderson-Sellers, B. and Gonzalez-Perez, C. (2008). “Comparison of Method Chunks and Method Fragments for Situational Method Engineering.” In: *Proceedings of the Australian Conference on Software Engineering*. Washington, DC, USA: IEEE Computer Society, pp. 479–488.

Henderson-Sellers, B., Gonzalez-Perez, C., and Ralyté, J. (2007). “Situational Method Engineering: Fragments or Chunks?” In: *Proceedings of CAiSE*. Vol. 7, pp. 11–15.

Henderson-Sellers, B., Simons, A., and Younessi, H. (1998). *The OPEN Toolbox of Techniques*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co.

Hevner, A. R. and Chatterjee, S. (2010). “Design Science Research Frameworks.” In: *Design Research in Information Systems*. New York, NY, USA: Springer, pp. 23–31.

Hevner, A. R. et al. (2004). “Design Science in Information Systems Research.” In: *MIS Quarterly* 28.1, pp. 75–105.

Heym, M. and Osterle, H. (1992). “A semantic data model for methodology engineering.” In: *Proceedings of the International Workshop on Computer-Aided Software Engineering*. IEEE, pp. 142–155.

Hietala, J., Kontio, J., and Jokinen, J. (2004). “Challenges of Software Product Companies: Results of a National Survey in Finland.” In:

Hofmann, H. et al. (2007). *CMMi for Outsourcing: Guidelines for Software, Systems, and IT Acquisition*. Addison-Wesley Professional.

Hug, C., Front, A., and Rieu, D. (2008). “A Process Engineering Method based on a Process Domain Model and Patterns.” In: *Proceedings of Méthodes Avancées de Développement de Systèmes d’Information*.

Humphrey, W. (1989). *Managing the Software Process*. Ed. by Habermann, N. 1st. Massachusetts: Addison-Wesley Publishing Company.

ISO/IEC (2007). *Software Engineering — Metamodel for Development Methodologies*.

Iversen, J. H., Mathiassen, L., and Nielsen, P. A. (2004). "Managing Risk in Software Process Improvement: An Action Research Approach." In: *Mis Quarterly* 25.3, pp. 395–433.

Jansen, S. and Brinkkemper, S. (2007). "Applied Multi-Case Research in a Mixed-Method Research Project: Customer Configuration Updating Improvement." In: *Information Systems Research Methods, Epistemology, and Applications*. Ed. by Cater-Steel, A. and Al-Hakimi, L. Utrecht: IGI Global, pp. 120–139.

Jarke, M. et al. (1995). "ConceptBase - a Deductive Object Base for Meta Data Management." In: *Journal of Intelligent Information Systems* 4.2, pp. 167–192.

Jecker, J. et al. (2007). "Facilitating change management with configurative reference modelling." In: *International Journal of Information Systems and Change Management* 2.1, p. 19.

Kanellis, P., Lycett, M., and Paul, R. J. (1999). "Evaluating Business Information Systems Fit: From Concept to Practical Application." In: *European Jouurnal on Information Systems* 8.1, pp. 65–76.

Kano, N. et al. (1984). "Attractive Quality and Must-Be Quality." In: *The Journal of the Japanese Society for Quality Control* 14.2, pp. 39–48.

Karlsson, F. (2008). "Method Tailoring as Negotiation." In: *Proceedings of CAiSE Forum*. Berlin, Germany: Springer, pp. 1–4.

– (2012). "Longitudinal use of method rationale in method configuration: an exploratory study." In: *European Journal of Information Systems* 22.6, pp. 690–710.

Karlsson, F. and Ågerfalk, P. J. (2009). "Towards Structured Flexibility in Information Systems Development: Devising a Method for Method Configuration." In: *Journal of Database Management* 20.3, pp. 51–75.

– (2012). "MC Sandbox: Devising a Tool for Method-User-Centered Method Configuration." In: *Information and Software Technology* 54.5, pp. 501–516.

Karlsson, F. and Wistrand, K. (2006). "Combining Method Engineering with Activity Theory: Theoretical Grounding of the Method Component Concept." In: *European Journal of Information Systems* 15.1, pp. 82–90.

Kelly, S., Lyytinen, K., and Rossi, M. (1996). "MetaEdit+: a Fully Configurable Multi-User and Multi-tool CASE and CAME Environment." In: *Proceedings of the Conference on Advanced Information Systems Engineering*, pp. 1–21.

Kerzner, H. R. (2013). *Project Management: a Systems Approach to Planning, Scheduling, and Controlling*. Wiley.

Kittlaus, H.-B. and Clough, P. N. (2009). *Software Product Management and Pricing: Key Success Factors for Software Organizations*. Berlin, Germany: Springer, p. 231.

Knuth, D. (1997). *The Art of Computer Programming. Volume 1: Fundamental Algorithms*. Third edit. Reading, Massachusetts: Addison-Wesle, p. 650.

Komi-Sirvio, S. (2004). "Development and Evaluation of Software Process Improvement Methods." PhD thesis. University of Oulo.

Kruchten, P. (1995). "Architectural Blueprints—The "4+ 1" View Model of Software Architecture." In: *IEEE Software* 12.November, pp. 42–50.

Krzanik, L. and Jouni, S. (2002). "Is My Software Process Improvement Suitable for Incremental Deployment?" In: *Proceedings of the International Workshop on Software Technology and Engineering Practice*. London, UK, pp. 76–87.

Kuilboer, J. and Ashrafi, N. (2000). "Software Process and Product Improvement: an Empirical Assessment." In: *Information and software technology* 42.1, pp. 27–34.

Kumar, K. and Welke, R. (1992). "Methodology Engineering R: a Proposal for Situation-Specific Methodology Construction." In: *Challenges and strategies for research in systems development*. John Wiley & Sons, Inc. New York, NY, USA: John Wiley & Sons, Inc., p. 269.

Kuvaja, P. et al. (1994). *Software Process Assessment and Improvement - The Bootstrap Approach*.

Lee, A. S. and Baskerville, R. L. (2003). "Generalizing Generalizability in Information Systems Research." In: *Information Systems Research* 14.3, p. 221.

Lee, C., Guadagno, L., and Jia, X. (2003). "An Agile Approach to Capturing Requirements and Traceability." In: *International Workshop on Traceability in Emerging Forms of Software Engineering*.

Lee, G. and Xia, W. (2010). "Toward Agile: an Integrated Analysis of Quantitative and Qualitative Field Data on Software Development Agility." In: *Mis Quarterly* 34.1, pp. 87–114.

Leenen, W. et al. (2012). "Transforming to Product Software: A Case Study on the Evolution of Software Product Management Processes during the Stages of Productization." In: *Proceedings of the International Conference on Software Business*. Ed. by Cusumano, M., Iyer, B., and Venkatraman, V. Boston, MA, USA: Springer, pp. 40–54.

Leone, N. et al. (2006). "The DLV System for Knowledge Representation and Reasoning." In: *ACM Transactions on Computational Logic* 7.3, pp. 499–562.

Lindvall, M., Rus, I., and Sinha, S. S. (2003). "Software systems support for knowledge management." In: *Journal of Knowledge Management* 7.5, pp. 137–150.

Livermore, J. a. (2008). "Factors that Significantly Impact the Implementation of an Agile Software Development Methodology." In: *Journal of Software* 3.4, pp. 31–36.

Maglyas, A., Nikula, U., and Smolander, K. (2011). "What do we Know About Software Product Management? a Systematic Mapping Study." In: *Software Product Management (IWSPM), 2011 Fifth International Workshop on*. IEEE, pp. 26–35.

Mann, C. and Mauer, F. (2005). "A Case Study on the Impact of Scrum on Overtime and Customer Satisfaction." In: *Agile Development Conference*, pp. 70–79.

March, S. T. and Smith, G. F. (1995). "Design and Natural Science Research on Information Technology (Invited Paper)." In: *Decision Support Systems (Special issue on WITS '92)* 15.4, pp. 251–266.

McConnell, S. (1994). *Code Complete: A Practical Handbook of Software Construction*. 2nd editio. Microsoft Press, p. 914.

McFeeley, B. (1996). *IDEAL: A User's Guide for Software Process Improvement*. Tech. rep. Pittsburgh, PA: Carnegie Mellon University.

McGarry, F. et al. (1994). *Software Process Improvement in the NASA Software Engineering Laboratory*. Tech. rep. Pitssburgh: Software Engineering Institute.

Miles, R. and Snow, C. (1978). "Organizational Strategy, Structure, and Process." In: *Academy of management ...* 3.3, pp. 546–562.

Mirbel, I. (2007). "Connecting Method Engineering Knowledge: a Community Based Approach." In: *Situational Method Engineering: Fundamentals and Experiences*. Vol. 244. Springer, pp. 176–192.

Mirbel, I. and Ralyté, J. (2006). "Situational Method Engineering: Combining Assembly Based and Roadmap-Driven Approaches." In: *Requirements Engineering* 11.1, pp. 58–78.

Mishra, D., Aydin, S., and Mishra, A. (2013). "Situational Requirement Method System: Knowledge Management in Business Support." In: *New Trends in Databases and Information Systems* 185, pp. 349–359.

Moe, N. B. and Aurum, A. (2008). "Understanding Decision-Making in Agile Software Development: A Case-study." In: *Proceedings of the Conference on Software Engineering and Advanced Applications*. IEEE, pp. 216–223.

Nehan, Y.-R. and Deneckère, R. (2007). "Component-Based Situational Methods: a Framework for Understanding SME." In: *Situational Method Engineering: Fundamentals and Experiences*. Vol. 244. Springer, p. 161.

Nerur, S., Mahapatra, R., and Mangalaraj, G. (2005). "Challenges of Migrating to Agile Methodologies." In: *Communications of the ACM* 48.2, pp. 72–79.

Niazi, M. (2011). "An Exploratory Study of Software Process Improvement Implementation Risks." In: *Journal of Software Maintenance and Evolution: Research and Practice*.

Nunamaker Jr., J. F., Chen, M., and Purdin, T. D. (1990). "Systems Development in Information Systems Research." In: *Journal of Management Information Systems - Special issue on management support systems* 7.3, pp. 631–640.

Object Management Group (2005). *Unified modeling language 2.0*.

– (2011). *Meta Object Facility (MOF^{TM})*. Tech. rep. Object Management Group.

Ocampo, A. and Munch, J. (2006). "Process Evolution Supported by Rationale: An Empirical Investigation of Process Changes." In: *Lecture Notes in Computer Science 3966*. Berlin/Heidelberg: Springer, pp. 334–341.

OMG (2008). *Software & Systems Process Engineering Meta-Model Specification (SPEM)*. Tech. rep. April, p. 236.

Omran, A. (2008). "AGILE CMMI from SMEs perspective." In: *Information and Communication Technologies: From Theory to Applications, 2008. ICTTA 2008. 3rd International Conference on*. IEEE, pp. 1–8.

Palmer, S. and Felsing, J. (2002). *A Practical Guide to Feature-Driven Development*. Prentice Hall.

Paulk, M. C. and Curtis, B. (1993). "The Capability Maturity Model for Software." In: *IEEE Software* 10.4, pp. 18–27.

Paulk, M. C. et al. (1995). *The Capability Maturity Model: Guidelines for Improving the Software Process*. Boston: Addison-Wesley.

Peffers, K. et al. (2007). "A Design Science Research Methodology for Information Systems Research." In: *Journal of Management Information Systems* 24.3, pp. 45–77.

Pemberton, J. D., Stonehouse, G. H., and Yarrow, D. J. (2001). "Benchmarking and the Role of Organizational Learning in Developing Competitive Advantage." In: *Knowledge and Process Management* 8.2, pp. 123–135.

Petersen, K. and Wohlin, C. (2010). "Software Process Improvement Through the Lean Measurement (SPI-LEAM) Method." In: *Journal of Systems and Software* 83.7, pp. 1275–1287.

Pettersson, F. et al. (2004). “Packaging Software Process Improvement Issues: a Method and a Case Study.” In: *Software Practice and Experience* 34.14, pp. 1311–1344.

Pichler, M., Rumetshofer, H., and Wahler, W. (2006). “Agile Requirements Engineering for a Social Insurance for Occupational Risks Organization: A Case Study.” In: *14th IEEE International Requirements Engineering Conference*, pp. 251–256.

Pino, F. J., García, F., and Piattini, M. (2008). “Software Process Improvement in Small and Medium Software Enterprises: a Systematic Review.” In: *Software Quality Journal* 16.2, pp. 237–261.

Pino, F. J. et al. (2010). “Using Scrum to Guide the Execution of Software Process Improvement in Small Organizations.” In: *Journal of Systems and Software* 83.10, pp. 1662–1677.

Pittman, M. (1996). “Lessons Learned in Managing Object-Oriented Development.” In: *IEEE Software* 10.1, pp. 43–53.

Prakash, N. and Gupta, D. (1998). “An Architecture for a CAME Tool.” In: *Information Modelling and Knowledge Bases X*. Ed. by Kangassalo, H., pp. 200–219.

Pyzdek, T. and Keller, P. A. (2003). *The Six Sigma Handbook*. Vol. 486. McGraw-Hill New York, NY.

Racheva, Z., Daneva, M., and Buglione, L. (2008). “Supporting the Dynamic Reprioritization of Requirements in Agile Development of Software Products.” In: *Proceedings of the International Workshop on Software Product Management*, pp. 49–58.

Ralston, D., Wright, A., and Kumar, J. (2001). “Process Benchmarking as a Market Research Tool for Strategic Planning.” In: *Marketing Intelligence & Planning* 19.4, pp. 273–281.

Ralyté, J. (1999). “Reusing Scenario Based Approaches in Requirement Engineering Methods: CREWS Method Base.” In: *International Workshop on Database and Expert Systems Applications (DEXA)*. Dexa 1999. Ieee, pp. 305–309.

– (2004). “Towards Situational Methods for Information Systems Development: Engineering Reusable Method Chunks.” In: *Proceedings of the International Conference on Information Systems Development*.

Ralyté, J., Deneckère, R., and Rolland, C. (2003). “Towards a Generic Model for Situational Method Engineering.” In: *Advanced Information Systems Engineering*. Springer, pp. 1029–1029.

Ralyté, J. and Rolland, C. (2001). "An Approach for Method Reengineering." In: *Proceedings of the 20th International Conference on Conceptual Modeling*. Springer-Verlag, pp. 267–283.

Ralyté, J., Rolland, C., and Ayed, M. B. (2005). "An Approach for Evolution-Driven Method Engineering." In: *Information Modeling Methods and Methodologies*, pp. 80–202.

Ralyté, J., Rolland, C., and Plihon, V. (1999). "Method Enhancement with Scenario Based Techniques." In: *Proceedings of CAiSE*, pp. 103–118.

Ralyté, J. et al. (2005). "Map-driven Modular Method Re-engineering: Improving the RESCUE Requirements Process." In: *Proceedings of the International Conference on Advanced Information Systems Engineering*, pp. 209–224.

Ralyté, J. et al. (2008). "A Knowledge-Based Approach to Manage Information Systems Interoperability." In: *Information Systems* 33.7-8, pp. 754–784.

Regnell, B., Berntsson Svensson, R., and Olsson, T. (2008). "Supporting Roadmapping of Quality Requirements." In: *Software, IEEE* 25.2, pp. 42–47.

Regnell, B. et al. (2001). "An Industrial Case Study on Distributed Prioritisation in Market-Driven Requirements Engineering for Packaged Software." In: *Requirements Engineering* 6.1, pp. 51–62.

Robertson, J. and Robertson, S. (1998). *Volere Requirements Specification Template Edition 6.0*. Tech. rep. Atlantic Systems Guild.

Rolland, C. (2009). "Method Engineering: Towards Methods as Services." In: *Software Process: Improvement and Practice* 14.3, pp. 143–164.

Rolland, C. and Plihon, V. (1998). "Specifying the Reuse Context of Scenario Method Chunks." In: *Proceedings of CAiSE*, pp. 191–218.

Rolland, C., Prakash, N., and Benjamen, A. (1999). "A Multi-Model View of Process Modelling." In: *Requirements Engineering* 4.4, pp. 169–187.

Rossi, M. and Tolvanen, J.-P. (2000). "Method Rationale inMmethod Engineering." In: *Proceedings of the Hawaii International Conference on System Sciences*. Hawaii, pp. 1–10.

Rossi, M. et al. (2004). "Managing Evolutionary Method Engineering by Method Rationale." In: *Journal of the Association for Information Systems* 5.9, pp. 356–391.

Runeson, P. and Höst, M. (2008). "Guidelines for Conducting and Reporting Case Study Research in Software Engineering." In: *Empirical Software Engineering* 14.2, pp. 131–164.

Rus, I. and Lindvall, M. (2002). "Knowledge Management in Software Engineering." In: *IEEE Software* 19.3, pp. 26–38.

Saeki, M. (1995). “Object-oriented meta modelling.” In: *Object-Oriented and Entity-Relationship Modeling* 1021, pp. 250–259.

– (2003). “CAME: The first step to automated method engineering.” In: *Proceedings of the Workshop on Process Engineering for Object-Oriented and Component-Based Development*. Ed. by Crocker, R. and Steele Jr., G. L. Anaheim, California, USA: ACM, pp. 7–18.

Saeki, M. and Iguchi, K. (1993). “A Meta-Model for Representing Software Specification & Design Methods.” In: *Proceedings of the IFIP WG8.1 Working Conference on Information System Development Process*.

Si-Said, S., Rolland, C., and Grosz, G. (1996). “MENTOR: a Computer Aided Requirements Engineering Environment.” In: *Proceedings of the International Conference on Advanced Information Systems Engineering*. Springer, pp. 22–43.

Salo, O. (2006). “Enabling Software Process Improvement in Agile Software Development Teams and Organisations.” In: *Vtt Publications* 618.618, p. 153.

Salo, O. and Abrahamsson, P. (2008). “Agile Methods in European Embedded Software Development Organisations: a Survey on the Actual Use and Usefulness of Extreme Programming and Scrum.” In: *IET Software* 2.1, p. 58.

Sawyer, P., Sommerville, I., and Viller, S. (1997). “Requirements Process Improvement Through the Phased Introduction of Good Practice.” In: *Software Process: Improvement and Practice* 3.1, pp. 19–34.

Schmalensee, R. (2000). “Antitrust Issues in Schumpeterian Industries.” In: *American Economic Review*, pp. 192–196.

Schwaber, K. (1995). “Scrum Development Process.” In: *OOPSLA Business Object Design and Implementation Workshop*. Vol. 27. April 1987. Citeseer, pp. 10–19.

– (2004). *Agile Project Management with Scrum*. Redmond, WA, USA: Microsoft Press.

Schwanninger, C. et al. (2009). *Model Driven Engineering Languages and Systems*. Ed. by Schürr, A. and Selic, B. Vol. 5795. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 685–689.

Scotland, K. and Boutin, A. (2008). “Integrating Scrum with the Process Framework at Yahoo! Europe.” In: *Proceedings of the AGILE conference*. IEEE, pp. 191–195.

Sharma, N., Singh, K., and Goyal, D. (2010). “Software Process Improvement through Experience Management : An Empirical Analysis of Critical Success.”

In: *Proceedings of the International Conference on Information Systems, Technology and Management*. Springer, pp. 386–391.

Sharma, N., Singh, K., and Goyal, D. (2011). "Experience Based Software Process Improvement: Have We Found the Silver Bullet?" In: *Information Intelligence, Systems, Technology and Management* 141.2, pp. 71–80.

Shih, C.-C. and Huang, S.-J. (2010). "Exploring the Relationship Between Organizational Culture and Software Process Improvement Deployment." In: *Information & Management* 47.5-6, pp. 271–281.

Sidky, A. S. (2007). "A Structured Approach to Adopting Agile Practices: The Agile Adoption Framework." In:

Simon, H. A. (1956). "Rational Choice and the Structure of the Environment." In: *Psychological review* 63.2, p. 129.

Smith, M. E. (2002). "Success Rates for Different Types of Organizational Change." In: *Performance Improvement* 41.1, pp. 26–33.

Smits, H. and Pshigoda, G. (2007). "Implementing Scrum in a Distributed Software Development Organization." In: *Management*. Agile 2007, pp. 371–375.

Smolander, K. and Lyytinen, K. (1991). "MetaEdit—a Flexible Graphical Environment for Methodology Modelling." In: *International Conference on Advanced Information Systems Engineering*, pp. 168–193.

Souer, J. and Mierloo, M. V. (2008). "A Component Based Architecture for Web Content Management: Runtime Deployable WebManager Component Bundles." In: *Proceedings of the International Conference on Web Engineering*, pp. 366–369.

Souer, J. et al. (2011). "Identifying commonalities in web content management system engineering." In: *International Journal of Web Information Systems* 7.3, pp. 292–308.

Staples, M. et al. (2007). "An Exploratory Study of Why Organizations Do Not Adopt CMMI." In: *Journal of systems and software* 80.6, pp. 883–895.

Stapleton, J. (1999). "DSDM: Dynamic Systems Development Method." In: *Proceedings of the Technology of Object-Oriented Languages and Systems*. Washington, DC, USA: IEEE Computer Society, p. 406.

Steenbergen, M. van et al. (2010). "The Dynamic Architecture Maturity Matrix : Instrument." In: *Proceedings of the International Conference ServiceWave*. Ed. by Dan, A., Gittler, F., and Toumani, F. Berlin, Germany: Springer, pp. 48–61.

Stijn, P. van et al. (2012). "Documenting Evolutionary Process Improvements with Method Increment Case Descriptions." In: *Proceedings of the European Conference on System & Software Process Improvement and Innovation (EuroSPI)*. Ed.

by Winkler, D., O'Connor, R. V., and Messnarz, R. Vienna, Austria: Springer, pp. 193–204.

Sulayman, M. et al. (2012). "Software Process Improvement Success Factors for Small and Medium Web companies: A Qualitative Study." In: *Information and Software Technology* 54.5, pp. 500–479.

Svahnberg, M. et al. (2010). "A Systematic Review on Strategic Release Planning Models." In: *Information and Software Technology* 52.3, pp. 237–248.

Team, C. P. (2010). "CMMI for Services, version 1.3." In:

Teece, D. and Pisano, G. (1994). "The Dynamic Capabilities of Firms: an Introduction." In: *Industrial and Corporate Change* 3.3, pp. 537–556.

Theuns, M., Vlaanderen, K., and Brinkkemper, S. (2012). "Exploring the Relationship Between Scrum and Release Planning Activities." In: *Software for People: Fundamentals, Trends and Best Practices*. Ed. by Maedche, A., Botzenhardt, A., and Neer, L. Berlin, Germany: Springer, pp. 239–256.

Thiagarajan, R. K. et al. (2002). "BPML: A Process Modeling Language for Dynamic Business Models." In: p. 239.

Tolvanen, J.-P. and Rossi, M. (2003). "MetaEdit+: Defining and Using Domain-Specific Modeling Languages and Code Generators." In: *Proceedings of the Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA03)*. Ed. by Crocker, R. and Steele Jr., G. L. Anaheim, California, USA: ACM.

Turner, R. and Jain, A. (2002). "Agile Meets CMMI: Culture Clash or Common Cause?" In: *Extreme Programming and Agile Methods—XP/Agile Universe 2002*. Springer, pp. 153–165.

Vähäniitty, J. and Rautiainen, K. T. (2008). "Towards a Conceptual Framework and Tool Support for Linking Long-Term Product and Business Planning with Agile Software Development." In: *Proceedings of the International Conference on Software Engineering*. Ed. by Schäfer, W., Dwyer, M. B., and Gruhn, V. Leipzig, Germany: ACM, pp. 25–28.

Vähäniitty, J. et al. (2011). *Towards Agile Product and Portfolio Management*. Ed. by Heikkilä, V., Rautiainen, K., and Vähäniitty, J. Espoo, Finland, p. 227.

Venners, B. (2003). *Programming is Gardening, not Engineering. A Conversation with Andy Hunt and Dave Thomas, Part VII*.

Vlaanderen, K. (2010a). "Improving Software Product Management Processes: a detailed view of the Product Software Knowledge Infrastructure." Master Thesis. Utrecht, the Netherlands: Utrecht University.

Vlaanderen, K. (2010b). “Model-Driven Assessment in Software Product Management.” In: *International Workshop on Software Product Management (IWSPM)*. Ed. by Fricker, S. and Aurum, A. Sydney, Australia: Springer, pp. 17–25.

Vlaanderen, K., Brinkkemper, S., and Weerd, I. van de (2011). “Evolutionary Process Improvement Using Method Increments.” In: *Proceedings of the International Working Conference on Requirements Engineering: Foundation for Software Quality (RefsQ) - Doctoral Symposium*. Ed. by Paech, B. and Gervasi, V. Essen, Germany: Springer, pp. 234–241.

Vlaanderen, K., Dalpiaz, F., and Brinkkemper, S. (2014). “Finding Optimal Plans for Incremental Method Engineering. Manuscript accepted for publication.” In: *International Conference on Advanced Information Systems Engineering*.

Vlaanderen, K., Stijn, P. van, and Brinkkemper, S. (2012). “Growing into Agility: Process Implementation Paths for Scrum.” In: *Proceedings of the International Conference on Product-Focused Software Development and Process Improvement (PROFES)*. Ed. by Dieste, O., Jedlitschka, A., and Juristo, N. Madrid, Spain: Springer, pp. 116–130.

Vlaanderen, K. and Tuijl, G. V. (2013). “Incremental Method Enactment for Computer Aided Software Engineering Tools.” In: *Workshop for Exploring Modelling Methods for Systems Analysis and Design (EMMSAD)*. Ed. by Nurcan, S. et al. Barcelona: Springer, pp. 370–384.

Vlaanderen, K., Valverde, F., and Pastor, O. (2008). “Improvement of a Web Engineering Method Applying Situational Method Engineering.” In: *Proceedings of the International Conference on Enterprise Information Systems (ICEIS)*. Ed. by Cordeiro, J. and Filipe, J. 1. Barcelona, Spain, pp. 1–14.

– (2009). “Model-Driven Web Engineering in the CMS Domain: A Preliminary Research Applying SME.” In: *Enterprise Information Systems (Selected papers of the 10th International Conference on Enterprise Information Systems)*. Ed. by Cordeiro, J. and Filipe, J. Volume 19. Berlin, Germany: Springer, pp. 226–237.

Vlaanderen, K., Weerd, I. van de, and Brinkkemper, S. (2011). “The Online Method Engine: From Process Assessment to Method Execution.” In: *Proceedings of the IFIP WG8.1 Working Conference on Method Engineering*. Ed. by Ralyté, J., Mirbel, I., and Deneckère, R. Paris, France: Springer, pp. 108–122.

– (2012). “On the Design of a Knowledge Management System for Incremental Process Improvement for Software Product Management.” In: *International Journal of Information System Modeling and Design (IJISMD)* 3.4, pp. 46–66.
– (2013). “Improving Software Product Management: a Knowledge Management Approach.” In: *International Journal of Business Information Systems (IJBIS)* 12.1, pp. 3–22.

Vlaanderen, K. et al. (2009). *Agile Product Management at Planon*. Tech. rep. Utrecht University.

Vlaanderen, K. et al. (2011). “The Agile Requirements Refinery: Applying SCRUM Principles to Software Product Management.” In: *Information and Software Technology (IST)* 53.1, pp. 58–70.

Vlaanderen, K. et al. (2014). “Demonstration of the Online Method Engine.” In: *Proceedings of the International Conference on Advanced Information Systems Engineering Forum (accepted for publication)*. Ed. by Nurcan, S. and Pimenidis, E. Thessaloniki, Greece.

Vlaanderen, K. et al. (2014). “Online Method Engine: a Toolset for Method Assessment, Improvement, and Enactment. Manuscript accepted for publication.” In: *International Journal of Information System Modeling and Design (IJISMD)* 5.3.

Weerd, I. van de and Brinkkemper, S. (2008). “Meta-Modeling for Situational Analysis and Design Methods.” In: *Handbook of Research on Modern Systems Analysis and Design Technologies and Applications*. Information Science Publishing. Chap. III, pp. 35–54.

Weerd, I. van de, Brinkkemper, S., and Versendaal, J. (2007). “Concepts for Incremental Method Evolution : Empirical Exploration and Validation in Requirements Management.” In: *Proceedings of the International Conference on Advanced Information Systems Engineering*. Springer, pp. 469–484.
– (2010). “Incremental Method Evolution in Global Software Product Management: A Retrospective Case Study.” In: *Information and Software Technology* 52.7, pp. 720–732.

Weerd, I. van de, Versendaal, J., and Brinkkemper, S. (2006). “A product software knowledge infrastructure for situational capability maturation: Vision and case studies in product management.” In: *Proceedings of the International Working Conference on Requirements Engineering: Foundation for Software Quality*. Ed. by Regnell, B., Kamsties, E., and Gervasi, V. Luxemburg, Luxemburg: Essener Informatik Beiträge, pp. 1–16.

Weerd, I. van de et al. (2006). "A situational implementation method for web-based content management system-applications: method engineering and validation in practice." In: *Software Process: Improvement and Practice* 11.5, pp. 521–538.

Weerd, I. van de et al. (2006a). "On the Creation of a Reference Framework for Software Product Management: Validation and Tool Support." In: *Proceedings of the International Workshop on Software Product Management*. Ed. by Brinkkemper, S., Ebert, C., and Versendaal, J. Minneapolis, USA: IEEE, pp. 3–12.

– (2006b). "Towards a Reference Framework for Software Product Management." In: *14th IEEE International Requirements Engineering Conference (RE'06)*, pp. 319–322.

Weinberg, G. M. (1997). *Quality Software Management: Anticipating change*. Quality Software Management. Dorset House Pub.

Wiegers, K. (1999). "First things first: prioritizing requirements." In: *Software Development* 7.9, pp. 1–6.

Wiig, K. M. (1997). "Knowledge Management: An Introduction and Perspective." In: *Journal of Knowledge Management* 1.1, pp. 6–14.

Wistrand, K. and Karlsson, F. (2004). "Method Components – Rationale Revealed." In: *Proceedings of the International Conference on Advanced Information Systems Engineering*, pp. 189–201.

Xu, L. and Brinkkemper, S. (2007). "Concepts of Product Software." In: *European Journal of Information Systems*. Vol. 16. 5, pp. 531–541.

Yin, R. K. (2003). *Case Study Research - Design and Methods*. SAGE Publications.

PUBLISHED WORK

We list here all published work by the author, split into refereed (with subcategories for journal, conference, workshops, and others) and un-refereed (book chapters) work.

REFEREED JOURNAL PAPERS

Vlaanderen, K. et al. (2014). "Online Method Engine: a Toolset for Method Assessment, Improvement, and Enactment. Manuscript accepted for publication." In: *International Journal of Information System Modeling and Design (IJISMD)* 5.3.

Vlaanderen, K., Weerd, I. van de, and Brinkkemper, S. (2013). "Improving Software Product Management: a Knowledge Management Approach." In: *International Journal of Business Information Systems (IJBIS)* 12.1, pp. 3–22.

Vlaanderen, K. et al. (2011). "The Agile Requirements Refinery: Applying SCRUM Principles to Software Product Management." In: *Information and Software Technology (IST)* 53.1, pp. 58–70.

REFEREED CONFERENCE PAPERS

Vlaanderen, K., Dalpiaz, F., and Brinkkemper, S. (2014). "Finding Optimal Plans for Incremental Method Engineering. Manuscript accepted for publication." In: *International Conference on Advanced Information Systems Engineering*.

Leenen, W. et al. (2012). "Transforming to Product Software: A Case Study on the Evolution of Software Product Management Processes during the Stages of Productization." In: *Proceedings of the International Conference on Software Business*. Ed. by Cusumano, M., Iyer, B., and Venkatraman, V. Boston, MA, USA: Springer, pp. 40–54.

Stijn, P. van et al. (2012). "Documenting Evolutionary Process Improvements with Method Increment Case Descriptions." In: *Proceedings of the European Conference on System & Software Process Improvement and Innovation (EuroSPI)*. Ed.

by Winkler, D., O'Connor, R. V., and Messnarz, R. Vienna, Austria: Springer, pp. 193–204.

Vlaanderen, K., Weerd, I. van de, and Brinkkemper, S. (2012). "On the Design of a Knowledge Management System for Incremental Process Improvement for Software Product Management." In: *International Journal of Information System Modeling and Design (IJISMD)* 3.4, pp. 46–66.

Vlaanderen, K., Valverde, F., and Pastor, O. (2008). "Improvement of a Web Engineering Method Applying Situational Method Engineering." In: *Proceedings of the International Conference on Enterprise Information Systems (ICEIS)*. Ed. by Cordeiro, J. and Filipe, J. 1. Barcelona, Spain, pp. 1–14.

REFEREED WORKSHOP PAPERS

Vlaanderen, K. et al. (2014). "Demonstration of the Online Method Engine." In: *Proceedings of the International Conference on Advanced Information Systems Engineering Forum (accepted for publication)*. Ed. by Nurcan, S. and Pimenidis, E. Thessaloniki, Greece.

Vlaanderen, K. and Tuijl, G. V. (2013). "Incremental Method Enactment for Computer Aided Software Engineering Tools." In: *Workshop for Exploring Modelling Methods for Systems Analysis and Design (EMMSAD)*. Ed. by Nurcan, S. et al. Barcelona: Springer, pp. 370–384.

Vlaanderen, K., Brinkkemper, S., and Weerd, I. van de (2011). "Evolutionary Process Improvement Using Method Increments." In: *Proceedings of the International Working Conference on Requirements Engineering: Foundation for Software Quality (RefsQ) - Doctoral Symposium*. Ed. by Paech, B. and Gervasi, V. Essen, Germany: Springer, pp. 234–241.

Vlaanderen, K., Weerd, I. van de, and Brinkkemper, S. (2011). "The Online Method Engine: From Process Assessment to Method Execution." In: *Proceedings of the IFIP WG8.1 Working Conference on Method Engineering*. Ed. by Ralyté, J., Mirbel, I., and Deneckère, R. Paris, France: Springer, pp. 108–122.

Vlaanderen, K. (2010b). "Model-Driven Assessment in Software Product Management." In: *International Workshop on Software Product Management (IWSPM)*. Ed. by Fricker, S. and Aurum, A. Sydney, Australia: Springer, pp. 17–25.

REFEREED BOOK CHAPTERS

Theuns, M., Vlaanderen, K., and Brinkkemper, S. (2012). "Exploring the Relationship Between Scrum and Release Planning Activities." In: *Software for People: Fundamentals, Trends and Best Practices*. Ed. by Maedche, A., Botzenhardt, A., and Neer, L. Berlin, Germany: Springer, pp. 239–256.

Vlaanderen, K., Valverde, F., and Pastor, O. (2009). "Model-Driven Web Engineering in the CMS Domain: A Preliminary Research Applying SME." In: *Enterprise Information Systems (Selected papers of the 10th International Conference on Enterprise Information Systems)*. Ed. by Cordeiro, J. and Filipe, J. Volume 19. Berlin, Germany: Springer, pp. 226–237.

BOOKS

Vähäniitty, J. et al. (2011). *Towards Agile Product and Portfolio Management*. Ed. by Heikkilä, V., Rautiainen, K., and Vähäniitty, J. Espoo, Finland, p. 227.

REPORTS

Vlaanderen, K. (2010a). "Improving Software Product Management Processes: a detailed view of the Product Software Knowledge Infrastructure." Master Thesis. Utrecht, the Netherlands: Utrecht University.

Vlaanderen, K. et al. (2009). *Agile Product Management at Planon*. Tech. rep. Utrecht University.

SUMMARY

With the research described in this dissertation, we aim to shed light on the characteristics of process improvement efforts by looking at their *evolution* (*how* to change?) rather than their *content* (*what* to change?). This research is triggered by three main propositions, derived from earlier work: (i) methods can be dissected into method fragments; (ii) the process of method evolution can be dissected into method increments; and (iii) method increments can be used to guide method construction.

These assumptions have not been thoroughly tested and explored yet. Especially the method increment concept is not well understood. The research reported in this dissertation tries to fill this gap by answering the following question: *How can method increments be used to support process improvement efforts?* The dissertation provides an answer to this question through three parts.

The first part describes two studies that investigate the characteristics of process improvement efforts, by looking at such efforts in the domain of agile software development. One study describes a case study, resulting in a method description for agile software product management. While this relates more to the *what* than the *how* of process improvement, it provides preliminary insights on the structure of process improvement efforts. We expand upon this insight in the second chapter, in which we analyze and compare the introduction of Scrum at four case companies. The results show a variety of approaches, with heterogeneous effects on effectiveness.

In **the second part** of the dissertation, we develop a better understanding of the notion of method increment. The first chapter expands on the topic of knowledge gathering related to method increments by providing a detailed description and evaluation of a method increment template. In the second chapter of this part, we propose and evaluate a formalization of the process-deliverable diagram, specifically aimed at describing method changes, along with a planning framework and process for generating plans that comply with different types of constraints.

In **the third part**, we describe the design, realization, and evaluation of a knowledge management system that supports incremental method engineering. This system, the Online Method Engine (OME), encompasses the dissemination

of method knowledge, linked to process assessment, improvement, and enactment. The OME is based firmly upon the results from the first two parts, and demonstrates a potential solution that helps practitioners during process improvement efforts.

SAMENVATTING (DUTCH SUMMARY)

Met het onderzoek dat wordt beschreven in deze dissertatie proberen we inzicht te bieden in de eigenschappen van procesverbeterinitiatieven door te kijken naar de evolutie (*hoe* verandert het?) in plaats van de specifieke veranderingen (*wat* verandert er?). Dit onderzoek is gebaseerd op drie proposities die zijn afgeleid van eerder werk: (i) methodes kunnen worden opgesplitst in methodefragmenten; (ii) het proces van methode-evolutie kan worden opgesplitst in methode-incrementen; en (iii) methode-incrementen kunnen worden gebruikt om methodeconstructie te geleiden.

Deze aannames zijn nog niet grondig getest en verkend. Met name onze kennis van het methode-incrementconcept schiet tekort. Het onderzoek dat wordt beschreven in deze dissertatie probeert dit gat op te vullen door de volgende vraag te beantwoorden: *Hoe kunnen methode-incrementen gebruikt worden om procesverbeterinitiatieven te ondersteunen?*. De dissertatie geeft antwoord op deze vraag middels drie delen.

Deel 1 beschrijft twee studies waarin de karakteristieken van procesverbeterprojecten worden onderzocht, wat resulteert in een methodebeschrijving voor agile softwareproductmanagement. Hoewel dit voornamelijk gerelateerd is aan de specifieke veranderingen van een procesverbetertraject en minder aan het verbeterproces, geeft dit een initieel inzicht in de structuur van procesverbeterinitiatieven. We borduren voort op dit inzicht in het tweede hoofdstuk, waarin we de introductie van Scrum binnen vier bedrijven bestuderen. De resultaten tonen verscheidene aanpakken, met sterk uiteenlopende effecten m.b.t. effectiviteit.

In **deel 2** van de dissertatie ontwikkelen we een beter begrip van het methode-incrementconcept. Het eerste hoofdstuk gaat in op het verzamelen van kennis met betrekking tot methode-incrementen, door middel van een gedetailleerde beschrijving en evaluatie van een template voor methode-incrementen. In het tweede hoofdstuk van dit deel introduceren en evalueren we een formalisatie van het*process-deliverable diagram* (PDD), specifiek gericht op het beschrijven van methodeveranderingen, in combinatie met een planningsraamwerk en een proces voor het genereren van plannen die voldoen aan verschillende typen voorwaarden.

In **deel 3** beschrijven we het ontwerp, de realisatie en de evaluatie van een kennismanagementsysteem dat ondersteuning biedt voor incrementele methodeconstructie. Dit systeem, de *online method engine* (OME), omvat de verspreiding van kennis gekoppeld aan het beoordelen, verbeteren en in werking stellen van processen. De OME is sterk gebaseerd op de resultaten van de eerste twee delen, en demonstreert een mogelijke oplossing die praktijkbeoefenaars helpt tijdens procesverbetertrajecten.

BIOGRAPHY

Kevin Vlaanderen was born on August 12th, 1986, in Hilversum, the Netherlands. From 2004 until 2008, he studied Information Science at Utrecht University, where he received his Bachelor of Science degree. During largely the same period, from 2005 until 2009, he also studied Spanish Language & Culture, which resulted in a Bachelor of Arts degree from the same university. After this, he continued with a master in Business Informatics, which resulted in a Master of Science degree in 2010.

His research interests focus on the domains of software product management, software process improvement, method engineering, and agile software development. He is an active reviewer and program committee member for several scientific conferences and journals, and he is a personal member of the International Software Product Management Board (ISPMA). In addition, he was the co-organizer of several editions of the ProductCamp Amsterdam unconference. During his PhD programme, he was the coordinator and lead teacher of the semi-yearly course on Software Product Management for professionals, taught at Utrecht University.

In his spare time, he is an avid reader of prose, and a volunteer at several poetry and literature events. His travels led him to all six inhabited continents.

SIKS DISSERTATION SERIES

1998

1998-01	Johan van den Akker (CWI) DEGAS - An Active, Temporal Database of Autonomous Objects.
1998-02	Floris Wiesman (UM) Information Retrieval by Graphically Browsing Meta-Information.
1998-03	Ans Steuten (TUD) A Contribution to the Linguistic Analysis of Business Conversations within the Language/Action Perspective.
1998-04	Dennis Breuker (UM) Memory versus Search in Games.
1998-05	E.W.Oskamp (RUL) Computerondersteuning bij Straftoemeting.

1999

1999-01	Mark Sloof (VU) Physiology of Quality Change Modelling; Automated modelling of Quality Change of Agricultural Products.
1999-02	Rob Potharst (EUR) Classification using decision trees and neural nets.
1999-03	Don Beal (UM) The Nature of Minimax Search.
1999-04	Jacques Penders (UM) The practical Art of Moving Physical Objects.
1999-05	Aldo de Moor (KUB) Empowering Communities: A Method for the Legitimate User-Driven Specification of Network Information Systems.
1999-06	Niek J.E. Wijngaards (VU) Re-design of compositional systems.
1999-07	David Spelt (UT) Verification support for object database design.
1999-08	Jacques H.J. Lenting (UM) Informed Gambling: Conception and Analysis of a Multi-Agent Mechanism for Discrete Reallocation.

2000

2000-01	Frank Niessink (VU) Perspectives on Improving Software Maintenance.
2000-02	Koen Holtman (TUE) Prototyping of CMS Storage Management.
2000-03	Carolien M.T. Metselaar (UVA) Sociaal-organisatorische gevolgen van kennistechnologie; een procesbenadering en actorperspectief.
2000-04	Geert de Haan (VU) ETAG, A Formal Model of Competence Knowledge for User Interface Design.
2000-05	Ruud van der Pol (UM) Knowledge-based Query Formulation in Information Retrieval.
2000-06	Rogier van Eijk (UU) Programming Languages for Agent Communication.
2000-07	Niels Peek (UU) Decision-theoretic Planning of Clinical Patient Management.
2000-08	Veerle Coup (EUR) Sensitivity Analyis of Decision-Theoretic Networks.
2000-09	Florian Waas (CWI) Principles of Probabilistic Query Optimization.
2000-10	Niels Nes (CWI) Image Database Management System Design Considerations, Algorithms and Architecture.
2000-11	Jonas Karlsson (CWI) Scalable Distributed Data Structures for Database Management.

2001

2001-01	Silja Renooij (UU) Qualitative Approaches to Quantifying Probabilistic Networks.
2001-02	Koen Hindriks (UU) Agent Programming Languages: Programming with Mental Models.
2001-03	Maarten van Someren (UvA) Learning as problem solving.
2001-04	Evgueni Smirnov (UM) Conjunctive and Disjunctive Version Spaces with. Instance-Based Boundary Sets.
2001-05	Jacco van Ossenbruggen (VU) Processing Structured Hypermedia: A Matter of Style.
2001-06	Martijn van Welie (VU) Task-based User Interface Design.
2001-07	Bastiaan Schonhage (VU)

Diva: Architectural Perspectives on Information Visualization.

2001-08 Pascal van Eck (VU)
A Compositional Semantic Structure for Multi-Agent Systems Dynamics.

2001-09 Pieter Jan 't Hoen (RUL)
Towards Distributed Development of Large Object-Oriented Models,
Views of Packages as Classes.

2001-10 Maarten Sierhuis (UvA)
Modeling and Simulating Work Practice.
BRAHMS: a multiagent modeling and simulation language for work practice analysis and design.

2001-11 Tom M. van Engers (VUA)
Knowledge Management: The Role of Mental Models in Business Systems Design.

2002

2002-01 Nico Lassing (VU)
Architecture-Level Modifiability Analysis.

2002-02 Roelof van Zwol (UT)
Modelling and searching web-based document collections.

2002-03 Henk Ernst Blok (UT)
Database Optimization Aspects for Information Retrieval.

2002-04 Juan Roberto Castelo Valdueza (UU)
The Discrete Acyclic Digraph Markov Model in Data Mining.

2002-05 Radu Serban (VU)
The Private Cyberspace Modeling Electronic Environments inhabited by Privacy-concerned Agents.

2002-06 Laurens Mommers (UL)
Applied legal epistemology; Building a knowledge-based ontology of the legal domain.

2002-07 Peter Boncz (CWI)
Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications.

2002-08 Jaap Gordijn (VU)
Value Based Requirements Engineering: Exploring Innovative E-Commerce Ideas.

2002-09 Willem-Jan van den Heuvel(KUB)
Integrating Modern Business Applications with Objectified Legacy Systems.

2002-10 Brian Sheppard (UM)
Towards Perfect Play of Scrabble.

2002-11 Wouter C.A. Wijngaards (VU)
Agent Based Modelling of Dynamics: Biological and Organisational Applications.

2002-12 Albrecht Schmidt (Uva)
Processing XML in Database Systems.

2002-13 Hongjing Wu (TUE)
A Reference Architecture for Adaptive Hypermedia Applications.

2002-14 Wieke de Vries (UU)

Agent Interaction: Abstract Approaches to Modelling, Programming and Verifying Multi-Agent Systems.

2002-15 Rik Eshuis (UT)
Semantics and Verification of UML Activity Diagrams for Workflow Modelling.

2002-16 Pieter van Langen (VU)
The Anatomy of Design: Foundations, Models and Applications.

2002-17 Stefan Manegold (UVA)
Understanding, Modeling, and Improving Main-Memory Database Performance.

2003

2003-01 Heiner Stuckenschmidt (VU)
Ontology-Based Information Sharing in Weakly Structured Environments.

2003-02 Jan Broersen (VU)
Modal Action Logics for Reasoning About Reactive Systems.

2003-03 Martijn Schuemie (TUD)
Human-Computer Interaction and Presence in Virtual Reality Exposure Therapy.

2003-04 Milan Petkovic (UT)
Content-Based Video Retrieval Supported by Database Technology.

2003-05 Jos Lehmann (UVA)
Causation in Artificial Intelligence and Law - A modelling approach.

2003-06 Boris van Schooten (UT)
Development and specification of virtual environments.

2003-07 Machiel Jansen (UvA)
Formal Explorations of Knowledge Intensive Tasks.

2003-08 Yongping Ran (UM)
Repair Based Scheduling.

2003-09 Rens Kortmann (UM)
The resolution of visually guided behaviour.

2003-10 Andreas Lincke (UvT)
Electronic Business Negotiation: Some experimental studies on the interaction between medium, innovation context and culture.

2003-11 Simon Keizer (UT)
Reasoning under Uncertainty in Natural Language Dialogue using Bayesian Networks.

2003-12 Roeland Ordelman (UT)
Dutch speech recognition in multimedia information retrieval.

2003-13 Jeroen Donkers (UM)
Nosce Hostem - Searching with Opponent Models.

2003-14 Stijn Hoppenbrouwers (KUN)
Freezing Language: Conceptualisation Processes across ICT-Supported Organisations.

2003-15 Mathijs de Weerdt (TUD)
Plan Merging in Multi-Agent Systems.

2003-16 Menzo Windhouwer (CWI)
Feature Grammar Systems - Incremental Maintenance of Indexes to Digital Media Warehouses.

2003-17 David Jansen (UT)
Extensions of Statecharts with Probability, Time, and Stochastic Timing.

2003-18 Levente Kocsis (UM)
Learning Search Decisions.

2004

2004-01 Virginia Dignum (UU)
A Model for Organizational Interaction: Based on Agents, Founded in Logic.

2004-02 Lai Xu (UvT)
Monitoring Multi-party Contracts for E-business.

2004-03 Perry Groot (VU)
A Theoretical and Empirical Analysis of Approximation in Symbolic Problem Solving.

2004-04 Chris van Aart (UVA)
Organizational Principles for Multi-Agent Architectures.

2004-05 Viara Popova (EUR)
Knowledge discovery and monotonicity.

2004-06 Bart-Jan Hommes (TUD)
The Evaluation of Business Process Modeling Techniques.

2004-07 Elise Boltjes (UM)
Voorbeeldig onderwijs; voorbeeldgestuurd onderwijs, een opstap naar.
abstract denken, vooral voor meisjes.

2004-08 Joop Verbeek(UM)
Politie en de Nieuwe Internationale Informatiemarkt, Grensregionale politiële gegevensuitwisseling en digitale expertise.

2004-09 Martin Caminada (VU)
For the Sake of the Argument; explorations into argument-based reasoning.

2004-10 Suzanne Kabel (UVA)
Knowledge-rich indexing of learning-objects.

2004-11 Michel Klein (VU)
Change Management for Distributed Ontologies.

2004-12 The Duy Bui (UT)
Creating emotions and facial expressions for embodied agents.

2004-13 Wojciech Jamroga (UT)
Using Multiple Models of Reality: On Agents who Know how to Play.

2004-14 Paul Harrenstein (UU)
Logic in Conflict. Logical Explorations in Strategic Equilibrium.

2004-15 Arno Knobbe (UU)
Multi-Relational Data Mining.

2004-16 Federico Divina (VU)
Hybrid Genetic Relational Search for Inductive Learning.

2004-17 Mark Winands (UM)
Informed Search in Complex Games.

2004-18 Vania Bessa Machado (UvA)

Supporting the Construction of Qualitative Knowledge Models.

2004-19 Thijs Westerveld (UT)
Using generative probabilistic models for multimedia retrieval.

2004-20 Madelon Evers (Nyenrode)
Learning from Design: facilitating multidisciplinary design teams.

2005

2005-01 Floor Verdenius (UVA)
Methodological Aspects of Designing Induction-Based Applications.

2005-02 Erik van der Werf (UM))
AI techniques for the game of Go.

2005-03 Franc Grootjen (RUN)
A Pragmatic Approach to the Conceptualisation of Language.

2005-04 Nirvana Meratnia (UT)
Towards Database Support for Moving Object data.

2005-05 Gabriel Infante-Lopez (UVA)
Two-Level Probabilistic Grammars for Natural Language Parsing.

2005-06 Pieter Spronck (UM)
Adaptive Game AI.

2005-07 Flavius Frasincar (TUE)
Hypermedia Presentation Generation for Semantic Web Information Systems.

2005-08 Richard Vdovjak (TUE)
A Model-driven Approach for Building Distributed Ontology-based Web Applications.

2005-09 Jeen Broekstra (VU)
Storage, Querying and Inferencing for Semantic Web Languages.

2005-10 Anders Bouwer (UVA)
Explaining Behaviour: Using Qualitative Simulation in Interactive Learning Environments.

2005-11 Elth Ogston (VU)
Agent Based Matchmaking and Clustering - A Decentralized Approach to Search.

2005-12 Csaba Boer (EUR)
Distributed Simulation in Industry.

2005-13 Fred Hamburg (UL)
Een Computermodel voor het Ondersteunen van Euthanasiebeslissingen.

2005-14 Borys Omelayenko (VU)
Web-Service configuration on the Semantic Web; Exploring how semantics meets pragmatics.

2005-15 Tibor Bosse (VU)
Analysis of the Dynamics of Cognitive Processes.

2005-16 Joris Graaumans (UU)
Usability of XML Query Languages.

2005-17 Boris Shishkov (TUD)
Software Specification Based on Re-usable Business Components.

2005-18 Danielle Sent (UU)
Test-selection strategies for probabilistic networks.

2005-19 Michel van Dartel (UM)
Situated Representation.

2005-20 Cristina Coteanu (UL)
Cyber Consumer Law, State of the Art and Perspectives.

2005-21 Wijnand Derks (UT)
Improving Concurrency and Recovery in Database Systems by Exploiting Application Semantics.

2006

2006-01 Samuil Angelov (TUE)
Foundations of B2B Electronic Contracting.

2006-02 Cristina Chisalita (VU)
Contextual issues in the design and use of information technology in organizations.

2006-03 Noor Christoph (UVA)
The role of metacognitive skills in learning to solve problems.

2006-04 Marta Sabou (VU)
Building Web Service Ontologies.

2006-05 Cees Pierik (UU)
Validation Techniques for Object-Oriented Proof Outlines.

2006-06 Ziv Baida (VU)
Software-aided Service Bundling - Intelligent Methods & Tools for Graphical Service Modeling.

2006-07 Marko Smiljanic (UT)
XML schema matching – balancing efficiency and effectiveness by means of clustering.

2006-08 Eelco Herder (UT)
Forward, Back and Home Again - Analyzing User Behavior on the Web.

2006-09 Mohamed Wahdan (UM)
Automatic Formulation of the Auditor's Opinion.

2006-10 Ronny Siebes (VU)
Semantic Routing in Peer-to-Peer Systems.

2006-11 Joeri van Ruth (UT)
Flattening Queries over Nested Data Types.

2006-12 Bert Bongers (VU)
Interactivation - Towards an e-cology of people, our technological environment, and the arts.

2006-13 Henk-Jan Lebbink (UU)
Dialogue and Decision Games for Information Exchanging Agents.

2006-14 Johan Hoorn (VU)
Software Requirements: Update, Upgrade, Redesign - towards a Theory of Requirements Change.

2006-15 Rainer Malik (UU)
CONAN: Text Mining in the Biomedical Domain.

2006-16 Carsten Riggelsen (UU)
Approximation Methods for Efficient Learning of Bayesian Networks.

2006-17 Stacey Nagata (UU)
User Assistance for Multitasking with Interruptions on a Mobile Device.

2006-18 Valentin Zhizhkun (UVA)
Graph transformation for Natural Language Processing.

2006-19 Birna van Riemsdijk (UU)
Cognitive Agent Programming: A Semantic Approach.

2006-20 Marina Velikova (UvT)
Monotone models for prediction in data mining.

2006-21 Bas van Gils (RUN)
Aptness on the Web.

2006-22 Paul de Vrieze (RUN)
Fundaments of Adaptive Personalisation.

2006-23 Ion Juvina (UU)
Development of Cognitive Model for Navigating on the Web.

2006-24 Laura Hollink (VU)
Semantic Annotation for Retrieval of Visual Resources.

2006-25 Madalina Drugan (UU)
Conditional log-likelihood MDL and Evolutionary MCMC.

2006-26 Vojkan Mihajloviç (UT)
Score Region Algebra: A Flexible Framework for Structured Information Retrieval.

2006-27 Stefano Bocconi (CWI)
Vox Populi: generating video documentaries from semantically annotated media repositories.

2006-28 Borkur Sigurbjornsson (UVA)
Focused Information Access using XML Element Retrieval.

2007

2007-01 Kees Leune (UvT)
Access Control and Service-Oriented Architectures.

2007-02 Wouter Teepe (RUG)
Reconciling Information Exchange and Confidentiality: A Formal Approach.

2007-03 Peter Mika (VU)
Social Networks and the Semantic Web.

2007-04 Jurriaan van Diggelen (UU)
Achieving Semantic Interoperability in Multi-agent Systems: a dialogue-based approach.

2007-05 Bart Schermer (UL)
Software Agents, Surveillance, and the Right to Privacy: a & Legislative Framework for Agent-enabled Surveillance.

2007-06 Gilad Mishne (UVA)
Applied Text Analytics for Blogs.

2007-07 Natasa Jovanovic (UT)
To Whom It May Concern - Addressee Identification in Face-to-Face Meetings.

2007-08 Mark Hoogendoorn (VU)
Modeling of Change in Multi-Agent Organizations.

2007-09 David Mobach (VU)
Agent-Based Mediated Service Negotiation.

2007-10 Huib Aldewereld (UU)
Autonomy vs. Conformity: an Institutional Perspective on Norms and Protocols.

2007-11 Natalia Stash (TUE)
Incorporating Cognitive/Learning Styles in a General-Purpose Adaptive Hypermedia System.

2007-12 Marcel van Gerven (RUN)
Bayesian Networks for Clinical Decision Support: A Rational Approach to Dynamic Decision-Making under Uncertainty.

2007-13 Rutger Rienks (UT)
Meetings in Smart Environments; Implications of Progressing Technology.

2007-14 Niek Bergboer (UM)
Context-Based Image Analysis.

2007-15 Joyca Lacroix (UM)
NIM: a Situated Computational Memory Model.

2007-16 Davide Grossi (UU)
Designing Invisible Handcuffs. Formal investigations in Institutions and Organizations for Multi-agent Systems.

2007-17 Theodore Charitos (UU)
Reasoning with Dynamic Networks in Practice.

2007-18 Bart Orriens (UvT)
On the development an management of adaptive business & collaborations.

2007-19 David Levy (UM)
Intimate relationships with artificial partners.

2007-20 Slinger Jansen (UU)
Customer Configuration Updating in a Software Supply Network.

2007-21 Karianne Vermaas (UU)
Fast diffusion and broadening use: A research on residential adoption and usage of broadband internet in the Netherlands between 2001 and 2005.

2007-22 Zlatko Zlatev (UT)
Goal-oriented design of value and process models from patterns.

2007-23 Peter Barna (TUE)
Specification of Application Logic in Web Information Systems.

2007-24 Georgina Ramírez Camps (CWI)
Structural Features in XML Retrieval.

2007-25 Joost Schalken (VU)
Empirical Investigations in Software Process Improvement.

2008

2008-01 Katalin Boer-Sorbán (EUR)
Agent-Based Simulation of Financial Markets: A modular,continuous-time approach.

2008-02 Alexei Sharpanskykh (VU)
On Computer-Aided Methods for Modeling and Analysis of Organizations.

2008-03 Vera Hollink (UVA)
Optimizing hierarchical menus: a usage-based approach.

2008-04 Ander de Keijzer (UT)
Management of Uncertain Data - towards unattended integration.

2008-05 Bela Mutschler (UT)
Modeling and simulating causal dependencies on process-aware information systems from a cost perspective.

2008-06 Arjen Hommersom (RUN)
On the Application of Formal Methods to Clinical Guidelines, an Artificial Intelligence Perspective.

2008-07 Peter van Rosmalen (OU)
Supporting the tutor in the design and support of adaptive e-learning.

2008-08 Janneke Bolt (UU)
Bayesian Networks: Aspects of Approximate Inference.

2008-09 Christof van Nimwegen (UU)
The paradox of the guided user: assistance can be counter-effective.

2008-10 Wauter Bosma (UT)
Discourse oriented summarization.

2008-11 Vera Kartseva (VU)
Designing Controls for Network Organizations: A Value-Based Approach.

2008-12 Jozsef Farkas (RUN)
A Semiotically Oriented Cognitive Model of Knowledge Representation.

2008-13 Caterina Carraciolo (UVA)
Topic Driven Access to Scientific Handbooks.

2008-14 Arthur van Bunningen (UT)
Context-Aware Querying; Better Answers with Less Effort.

2008-15 Martijn van Otterlo (UT)
The Logic of Adaptive Behavior: Knowledge Representation and Algorithms for the Markov Decision Process Framework in First-Order Domains.

2008-16 Henriette van Vugt (VU)
Embodied agents from a user's perspective.

2008-17 Martin Op 't Land (TUD)
Applying Architecture and Ontology to the Splitting and Allying of Enterprises.

2008-18 Guido de Croon (UM)
Adaptive Active Vision.

2008-19 Henning Rode (UT)
From Document to Entity Retrieval: Improving Precision and Performance of Focused Text Search.

2008-20 Rex Arendsen (UVA)
Geen bericht, goed bericht. Een onderzoek naar de effecten van de introductie van elektronisch berichtenverkeer met de overheid op de administratieve lasten van bedrijven.

2008-21 Krisztian Balog (UVA)
People Search in the Enterprise.

2008-22 Henk Koning (UU)
Communication of IT-Architecture.

2008-23 Stefan Visscher (UU)
Bayesian network models for the management of ventilator-associated pneumonia.

2008-24 Zharko Aleksovski (VU)
Using background knowledge in ontology matching.

2008-25 Geert Jonker (UU)
Efficient and Equitable Exchange in Air Traffic Management Plan Repair using Spender-signed Currency.

2008-26 Marijn Huijbregts (UT)
Segmentation, Diarization and Speech Transcription: Surprise Data Unraveled.

2008-27 Hubert Vogten (OU)
Design and Implementation Strategies for IMS Learning Design.

2008-28 Ildiko Flesch (RUN)
On the Use of Independence Relations in Bayesian Networks.

2008-29 Dennis Reidsma (UT)
Annotations and Subjective Machines - Of Annotators, Embodied Agents, Users, and Other Humans.

2008-30 Wouter van Atteveldt (VU)
Semantic Network Analysis: Techniques for Extracting, Representing and Querying Media Content.

2008-31 Loes Braun (UM)
Pro-Active Medical Information Retrieval.

2008-32 Trung H. Bui (UT)
Toward Affective Dialogue Management using Partially Observable Markov Decision Processes.

2008-33 Frank Terpstra (UVA)
Scientific Workflow Design; theoretical and practical issues.

2008-34 Jeroen de Knijf (UU)
Studies in Frequent Tree Mining.

2008-35 Ben Torben Nielsen (UvT)
Dendritic morphologies: function shapes structure.

2009

2009-01 Rasa Jurgelenaite (RUN)
Symmetric Causal Independence Models.

2009-02 Willem Robert van Hage (VU)
Evaluating Ontology-Alignment Techniques.

2009-03 Hans Stol (UvT)
A Framework for Evidence-based Policy Making Using IT

2009-04 Josephine Nabukenya (RUN)
Improving the Quality of Organisational Policy Making using Collaboration Engineering.

2009-05 Sietse Overbeek (RUN)
Bridging Supply and Demand for Knowledge Intensive Tasks - Based on Knowledge, Cognition, and Quality.

2009-06 Muhammad Subianto (UU)
Understanding Classification.
2009-07 Ronald Poppe (UT)
Discriminative Vision-Based Recovery and Recognition of Human Motion.
2009-08 Volker Nannen (VU)
Evolutionary Agent-Based Policy Analysis in Dynamic Environments.
2009-09 Benjamin Kanagwa (RUN)
Design, Discovery and Construction of Service-oriented Systems.
2009-10 Jan Wielemaker (UVA)
Logic programming for knowledge-intensive interactive applications.
2009-11 Alexander Boer (UVA)
Legal Theory, Sources of Law & the Semantic Web.
2009-12 Peter Massuthe (TUE, Humboldt-Universitaet zu Berlin)
perating Guidelines for Services.
2009-13 Steven de Jong (UM)
Fairness in Multi-Agent Systems.
2009-14 Maksym Korotkiy (VU)
From ontology-enabled services to service-enabled ontologies (making ontologies work in e-science with ONTO-SOA)
2009-15 Rinke Hoekstra (UVA)
Ontology Representation - Design Patterns and Ontologies that Make Sense.
2009-16 Fritz Reul (UvT)
New Architectures in Computer Chess.
2009-17 Laurens van der Maaten (UvT)
Feature Extraction from Visual Data.
2009-18 Fabian Groffen (CWI)
Armada, An Evolving Database System.
2009-19 Valentin Robu (CWI)
Modeling Preferences, Strategic Reasoning and Collaboration in Agent-Mediated Electronic Markets.
2009-20 Bob van der Vecht (UU)
Adjustable Autonomy: Controling Influences on Decision Making.
2009-21 Stijn Vanderlooy (UM)
Ranking and Reliable Classification.
2009-22 Pavel Serdyukov (UT)
Search For Expertise: Going beyond direct evidence.
2009-23 Peter Hofgesang (VU)
Modelling Web Usage in a Changing Environment.
2009-24 Annerieke Heuvelink (VUA)
Cognitive Models for Training Simulations.
2009-25 Alex van Ballegooij (CWI)
RAM: Array Database Management through Relational Mapping.
2009-26 Fernando Koch (UU)
An Agent-Based Model for the Development of Intelligent Mobile Services.
2009-27 Christian Glahn (OU)

Contextual Support of social Engagement and Reflection on the Web.

2009-28 Sander Evers (UT)
Sensor Data Management with Probabilistic Models.

2009-29 Stanislav Pokraev (UT)
Model-Driven Semantic Integration of Service-Oriented Applications.

2009-30 Marcin Zukowski (CWI)
Balancing vectorized query execution with bandwidth-optimized storage.

2009-31 Sofiya Katrenko (UVA)
A Closer Look at Learning Relations from Text.

2009-32 Rik Farenhorst (VU) and Remco de Boer (VU)
Architectural Knowledge Management: Supporting Architects and Auditors.

2009-33 Khiet Truong (UT)
How Does Real Affect Affect Affect Recognition In Speech?

2009-34 Inge van de Weerd (UU)
Advancing in Software Product Management: An Incremental Method Engineering Approach.

2009-35 Wouter Koelewijn (UL)
Privacy en Politiegegevens; Over geautomatiseerde normatieve informatie-uitwisseling.

2009-36 Marco Kalz (OUN)
Placement Support for Learners in Learning Networks.

2009-37 Hendrik Drachsler (OUN)
Navigation Support for Learners in Informal Learning Networks.

2009-38 Riina Vuorikari (OU)
Tags and self-organisation: a metadata ecology for learning resources in a multilingual context.

2009-39 Christian Stahl (TUE, Humboldt-Universitaet zu Berlin)
Service Substitution – A Behavioral Approach Based on Petri Nets.

2009-40 Stephan Raaijmakers (UvT)
Multinomial Language Learning: Investigations into the Geometry of Language.

2009-41 Igor Berezhnyy (UvT)
Digital Analysis of Paintings.

2009-42 Toine Bogers
Recommender Systems for Social Bookmarking.

2009-43 Virginia Nunes Leal Franqueira (UT)
Finding Multi-step Attacks in Computer Networks using Heuristic Search and Mobile Ambients.

2009-44 Roberto Santana Tapia (UT)
Assessing Business-IT Alignment in Networked Organizations.

2009-45 Jilles Vreeken (UU)
Making Pattern Mining Useful.

2009-46 Loredana Afanasiev (UvA)
Querying XML: Benchmarks and Recursion.

2010

2010-01 Matthijs van Leeuwen (UU)

Patterns that Matter.

2010-02 Ingo Wassink (UT)
Work flows in Life Science.

2010-03 Joost Geurts (CWI)
A Document Engineering Model and Processing Framework for Multimedia documents.

2010-04 Olga Kulyk (UT)
Do You Know What I Know? Situational Awareness of Co-located Teams in Multidisplay Environments.

2010-05 Claudia Hauff (UT)
Predicting the Effectiveness of Queries and Retrieval Systems.

2010-06 Sander Bakkes (UvT)
Rapid Adaptation of Video Game AI

2010-07 Wim Fikkert (UT)
Gesture interaction at a Distance.

2010-08 Krzysztof Siewicz (UL)
Towards an Improved Regulatory Framework of Free Software. Protecting user freedoms in a world of software communities and eGovernments.

2010-09 Hugo Kielman (UL)
A Politiele gegevensverwerking en Privacy, Naar een effectieve waarborging.

2010-10 Rebecca Ong (UL)
Mobile Communication and Protection of Children.

2010-11 Adriaan Ter Mors (TUD)
The world according to MARP: Multi-Agent Route Planning.

2010-12 Susan van den Braak (UU)
Sensemaking software for crime analysis.

2010-13 Gianluigi Folino (RUN)
High Performance Data Mining using Bio-inspired techniques.

2010-14 Sander van Splunter (VU)
Automated Web Service Reconfiguration.

2010-15 Lianne Bodenstaff (UT)
Managing Dependency Relations in Inter-Organizational Models.

2010-16 Sicco Verwer (TUD)
Efficient Identification of Timed Automata, theory and practice.

2010-17 Spyros Kotoulas (VU)
Scalable Discovery of Networked Resources: Algorithms, Infrastructure, Applications.

2010-18 Charlotte Gerritsen (VU)
Caught in the Act: Investigating Crime by Agent-Based Simulation.

2010-19 Henriette Cramer (UvA)
People's Responses to Autonomous and Adaptive Systems.

2010-20 Ivo Swartjes (UT)
Whose Story Is It Anyway? How Improv Informs Agency and Authorship of Emergent Narrative.

2010-21 Harold van Heerde (UT)
Privacy-aware data management by means of data degradation.

2010-22 Michiel Hildebrand (CWI)

End-user Support for Access to. Heterogeneous Linked Data.

2010-23 Bas Steunebrink (UU)
The Logical Structure of Emotions.

2010-24 Dmytro Tykhonov
Designing Generic and Efficient Negotiation Strategies.

2010-25 Zulfiqar Ali Memon (VU)
Modelling Human-Awareness for Ambient Agents: A Human Mindreading Perspective.

2010-26 Ying Zhang (CWI)
XRPC: Efficient Distributed Query Processing on Heterogeneous XQuery Engines.

2010-27 Marten Voulon (UL)
Automatisch contracteren.

2010-28 Arne Koopman (UU)
Characteristic Relational Patterns.

2010-29 Stratos Idreos(CWI)
Database Cracking: Towards Auto-tuning Database Kernels.

2010-30 Marieke van Erp (UvT)
Accessing Natural History - Discoveries in data cleaning, structuring, and retrieval.

2010-31 Victor de Boer (UVA)
Ontology Enrichment from Heterogeneous Sources on the Web.

2010-32 Marcel Hiel (UvT)
An Adaptive Service Oriented Architecture: Automatically solving Interoperability Problems.

2010-33 Robin Aly (UT)
Modeling Representation Uncertainty in Concept-Based Multimedia Retrieval.

2010-34 Teduh Dirgahayu (UT)
Interaction Design in Service Compositions.

2010-35 Dolf Trieschnigg (UT)
Proof of Concept: Concept-based Biomedical Information Retrieval.

2010-36 Jose Janssen (OU)
Paving the Way for Lifelong Learning; Facilitating competence development through a learning path specification.

2010-37 Niels Lohmann (TUE)
Correctness of services and their composition.

2010-38 Dirk Fahland (TUE)
From Scenarios to components.

2010-39 Ghazanfar Farooq Siddiqui (VU)
Integrative modeling of emotions in virtual agents.

2010-40 Mark van Assem (VU)
Converting and Integrating Vocabularies for the Semantic Web.

2010-41 Guillaume Chaslot (UM)
Monte-Carlo Tree Search.

2010-42 Sybren de Kinderen (VU)
Needs-driven service bundling in a multi-supplier setting - the computational e3-service approach.

2010-43 Peter van Kranenburg (UU)
A Computational Approach to Content-Based Retrieval of Folk Song Melodies.

2010-44 Pieter Bellekens (TUE)
An Approach towards Context-sensitive and User-adapted Access to Heterogeneous Data Sources, Illustrated in the Television Domain.

2010-45 Vasilios Andrikopoulos (UvT)
A theory and model for the evolution of software services.

2010-46 Vincent Pijpers (VU)
e3alignment: Exploring Inter-Organizational Business-ICT Alignment.

2010-47 Chen Li (UT)
Mining Process Model Variants: Challenges, Techniques, Examples.

2010-48 Milan Lovric (EUR)
Behavioral Finance and Agent-Based Artificial Markets.

2010-49 Jahn-Takeshi Saito (UM)
Solving difficult game positions.

2010-50 Bouke Huurnink (UVA)
Search in Audiovisual Broadcast Archives.

2010-51 Alia Khairia Amin (CWI)
Understanding and supporting information seeking tasks in multiple sources.

2010-52 Peter-Paul van Maanen (VU)
Adaptive Support for Human-Computer Teams: Exploring the Use of Cognitive Models of Trust and Attention.

2010-53 Edgar Meij (UVA)
Combining Concepts and Language Models for Information Access.

2011

2011-01 Botond Cseke (RUN)
Variational Algorithms for Bayesian Inference in Latent Gaussian Models.

2011-02 Nick Tinnemeier(UU)
Organizing Agent Organizations. Syntax and Operational Semantics of an Organization-Oriented Programming Language.

2011-03 Jan Martijn van der Werf (TUE)
Compositional Design and Verification of Component-Based Information Systems.

2011-04 Hado van Hasselt (UU)
Insights in Reinforcement Learning; Formal analysis and empirical evaluation of temporal-difference learning algorithms.

2011-05 Base van der Raadt (VU)
Enterprise Architecture Coming of Age - Increasing the Performance of an Emerging Discipline.

2011-06 Yiwen Wang (TUE)
Semantically-Enhanced Recommendations in Cultural Heritage.

2011-07 Yujia Cao (UT)
Multimodal Information Presentation for High Load Human Computer Interaction.

2011-08 Nieske Vergunst (UU)
BDI-based Generation of Robust Task-Oriented Dialogues.

2011-09 Tim de Jong (OU)
Contextualised Mobile Media for Learning.

2011-10 Bart Bogaert (UvT)
Cloud Content Contention.

2011-11 Dhaval Vyas (UT)
Designing for Awareness: An Experience-focused HCI Perspective.

2011-12 Carmen Bratosin (TUE)
Grid Architecture for Distributed Process Mining.

2011-13 Xiaoyu Mao (UvT)
Airport under Control. Multiagent Scheduling for Airport Ground Handling.

2011-14 Milan Lovric (EUR)
Behavioral Finance and Agent-Based Artificial Markets.

2011-15 Marijn Koolen (UvA)
The Meaning of Structure: the Value of Link Evidence for Information Retrieval.

2011-16 Maarten Schadd (UM)
Selective Search in Games of Different Complexity.

2011-17 Jiyin He (UVA)
Exploring Topic Structure: Coherence, Diversity and Relatedness.

2011-18 Mark Ponsen (UM)
Strategic Decision-Making in complex games.

2011-19 Ellen Rusman (OU)
The Mind ' s Eye on Personal Profiles.

2011-20 Qing Gu (VU)
Guiding service-oriented software engineering - A view-based approach.

2011-21 Linda Terlouw (TUD)
Modularization and Specification of Service-Oriented Systems.

2011-22 Junte Zhang (UVA)
System Evaluation of Archival Description and Access.

2011-23 Wouter Weerkamp (UVA)
Finding People and their Utterances in Social Media.

2011-24 Herwin van Welbergen (UT)
Behavior Generation for Interpersonal Coordination with Virtual Humans On Specifying, Scheduling and Realizing Multimodal Virtual Human Behavior.

2011-25 Syed Waqar ul Qounain Jaffry (VU)
Analysis and Validation of Models for Trust Dynamics.

2011-26 Matthijs Aart Pontier (VU)
Virtual Agents for Human Communication - Emotion Regulation and Involvement-Distance Trade-Offs in Embodied Conversational Agents and Robots.

2011-27 Aniel Bhulai (VU)
Dynamic website optimization through autonomous management of design patterns.

2011-28 Rianne Kaptein(UVA)
Effective Focused Retrieval by Exploiting Query Context and Document Structure.

2011-29 Faisal Kamiran (TUE)
Discrimination-aware Classification.

2011-30 Egon van den Broek (UT)

Affective Signal Processing (ASP): Unraveling the mystery of emotions.

2011-31 Ludo Waltman (EUR)
Computational and Game-Theoretic Approaches for Modeling Bounded Rationality.

2011-32 Nees-Jan van Eck (EUR)
Methodological Advances in Bibliometric Mapping of Science.

2011-33 Tom van der Weide (UU)
Arguing to Motivate Decisions.

2011-34 Paolo Turrini (UU)
Strategic Reasoning in Interdependence: Logical and Game-theoretical Investigations.

2011-35 Maaike Harbers (UU)
Explaining Agent Behavior in Virtual Training.

2011-36 Erik van der Spek (UU)
Experiments in serious game design: a cognitive approach.

2011-37 Adriana Burlutiu (RUN)
Machine Learning for Pairwise Data, Applications for Preference Learning and Supervised Network Inference.

2011-38 Nyree Lemmens (UM)
Bee-inspired Distributed Optimization.

2011-39 Joost Westra (UU)
Organizing Adaptation using Agents in Serious Games.

2011-40 Viktor Clerc (VU)
Architectural Knowledge Management in Global Software Development.

2011-41 Luan Ibraimi (UT)
Cryptographically Enforced Distributed Data Access Control.

2011-42 Michal Sindlar (UU)
Explaining Behavior through Mental State Attribution.

2011-43 Henk van der Schuur (UU)
Process Improvement through Software Operation Knowledge.

2011-44 Boris Reuderink (UT)
Robust Brain-Computer Interfaces.

2011-45 Herman Stehouwer (UvT)
Statistical Language Models for Alternative Sequence Selection.

2011-46 Beibei Hu (TUD)
Towards Contextualized Information Delivery: A Rule-based Architecture for the Domain of Mobile Police Work.

2011-47 Azizi Bin Ab Aziz(VU)
Exploring Computational Models for Intelligent Support of Persons with Depression.

2011-48 Mark Ter Maat (UT)
Response Selection and Turn-taking for a Sensitive Artificial Listening Agent.

2011-49 Andreea Niculescu (UT)
Conversational interfaces for task-oriented spoken dialogues: design aspects influencing interaction quality.

2012

2012-01 Terry Kakeeto (UvT)
Relationship Marketing for SMEs in Uganda.

2012-02 Muhammad Umair(VU)
Adaptivity, emotion, and Rationality in Human and Ambient Agent Models.

2012-03 Adam Vanya (VU)
Supporting Architecture Evolution by Mining Software Repositories.

2012-04 Jurriaan Souer (UU)
Development of Content Management System-based Web Applications.

2012-05 Marijn Plomp (UU)
Maturing Interorganisational Information Systems.

2012-06 Wolfgang Reinhardt (OU)
Awareness Support for Knowledge Workers in Research Networks.

2012-07 Rianne van Lambalgen (VU)
When the Going Gets Tough: Exploring Agent-based Models of Human Performance under Demanding Conditions.

2012-08 Gerben de Vries (UVA)
Kernel Methods for Vessel Trajectories.

2012-09 Ricardo Neisse (UT)
Trust and Privacy Management Support for Context-Aware Service Platforms.

2012-10 David Smits (TUE)
Towards a Generic Distributed Adaptive Hypermedia Environment.

2012-11 J.C.B. Rantham Prabhakara (TUE)
Process Mining in the Large: Preprocessing, Discovery, and Diagnostics.

2012-12 Kees van der Sluijs (TUE)
Model Driven Design and Data Integration in Semantic Web Information Systems.

2012-13 Suleman Shahid (UvT)
Fun and Face: Exploring non-verbal expressions of emotion during playful interactions.

2012-14 Evgeny Knutov(TUE)
Generic Adaptation Framework for Unifying Adaptive Web-based Systems.

2012-15 Natalie van der Wal (VU)
Social Agents. Agent-Based Modelling of Integrated Internal and Social Dynamics of Cognitive and Affective Processes.

2012-16 Fiemke Both (VU)
Helping people by understanding them - Ambient Agents supporting task execution and depression treatment.

2012-17 Amal Elgammal (UvT)
Towards a Comprehensive Framework for Business Process Compliance.

2012-18 Eltjo Poort (VU)
Improving Solution Architecting Practices.

2012-19 Helen Schonenberg (TUE)
What's Next? Operational Support for Business Process Execution.

2012-20 Ali Bahramisharif (RUN)
Covert Visual Spatial Attention, a Robust Paradigm for Brain-Computer Interfacing.

2012-21 Roberto Cornacchia (TUD)
Querying Sparse Matrices for Information Retrieval.

2012-22 Thijs Vis (UvT)
Intelligence, politie en veiligheidsdienst: verenigbare grootheden?

2012-23 Christian Muehl (UT)
Toward Affective Brain-Computer Interfaces: Exploring the Neurophysiology of Affect during Human Media Interaction.

2012-24 Laurens van der Werff (UT)
Evaluation of Noisy Transcripts for Spoken Document Retrieval.

2012-25 Silja Eckartz (UT)
Managing the Business Case Development in Inter-Organizational IT Projects: A Methodology and its Application.

2012-26 Emile de Maat (UVA)
Making Sense of Legal Text.

2012-27 Hayrettin Gurkok (UT)
Mind the Sheep! User Experience Evaluation & Brain-Computer Interface Games.

2012-28 Nancy Pascall (UvT)
Engendering Technology Empowering Women.

2012-29 Almer Tigelaar (UT)
Peer-to-Peer Information Retrieval.

2012-30 Alina Pommeranz (TUD)
Designing Human-Centered Systems for Reflective Decision Making.

2012-31 Emily Bagarukayo (RUN)
A Learning by Construction Approach for Higher Order Cognitive Skills Improvement, Building Capacity and Infrastructure.

2012-32 Wietske Visser (TUD)
Qualitative multi-criteria preference representation and reasoning.

2012-33 Rory Sie (OUN)
Coalitions in Cooperation Networks (COCOON)

2012-34 Pavol Jancura (RUN)
Evolutionary analysis in PPI networks and applications.

2012-35 Evert Haasdijk (VU)
Never Too Old To Learn – On-line Evolution of Controllers in Swarm- and Modular Robotics.

2012-36 Denis Ssebugwawo (RUN)
Analysis and Evaluation of Collaborative Modeling Processes.

2012-37 Agnes Nakakawa (RUN)
A Collaboration Process for Enterprise Architecture Creation.

2012-38 Selmar Smit (VU)
Parameter Tuning and Scientific Testing in Evolutionary Algorithms.

2012-39 Hassan Fatemi (UT)
Risk-aware design of value and coordination networks.

2012-40 Agus Gunawan (UvT)
Information Access for SMEs in Indonesia.

2012-41 Sebastian Kelle (OU)
Game Design Patterns for Learning.

2012-42 Dominique Verpoorten (OU)
Reflection Amplifiers in self-regulated Learning.

2012-43 Withdrawn.

2012-44 Anna Tordai (VU)
On Combining Alignment Techniques.

2012-45 Benedikt Kratz (UvT)
A Model and Language for Business-aware Transactions.

2012-46 Simon Carter (UVA)
Exploration and Exploitation of Multilingual Data for Statistical Machine Translation.

2012-47 Manos Tsagkias (UVA)
Mining Social Media: Tracking Content and Predicting Behavior.

2012-48 Jorn Bakker (TUE)
Handling Abrupt Changes in Evolving Time-series Data.

2012-49 Michael Kaisers (UM)
Learning against Learning - Evolutionary dynamics of reinforcement learning algorithms in strategic interactions.

2012-50 Steven van Kervel (TUD)
Ontologogy driven Enterprise Information Systems Engineering.

2012-51 Jeroen de Jong (TUD)
Heuristics in Dynamic Sceduling; a practical framework with a case study in elevator dispatching.

2013

2013-01 Viorel Milea (EUR)
News Analytics for Financial Decision Support.

2013-02 Erietta Liarou (CWI)
MonetDB/DataCell: Leveraging the Column-store Database Technology for Efficient and Scalable Stream Processing.

2013-03 Szymon Klarman (VU)
Reasoning with Contexts in Description Logics.

2013-04 Chetan Yadati(TUD)
Coordinating autonomous planning and scheduling.

2013-05 Dulce Pumareja (UT)
Groupware Requirements Evolutions Patterns.

2013-06 Romulo Goncalves(CWI)
The Data Cyclotron: Juggling Data and Queries for a Data Warehouse Audience.

2013-07 Giel van Lankveld (UvT)
Quantifying Individual Player Differences.

2013-08 Robbert-Jan Merk(VU)
Making enemies: cognitive modeling for opponent agents in fighter pilot simulators.

2013-09 Fabio Gori (RUN)
Metagenomic Data Analysis: Computational Methods and Applications.

2013-10 Jeewanie Jayasinghe Arachchige(UvT)
A Unified Modeling Framework for Service Design.

2013-11 Evangelos Pournaras(TUD)

Multi-level Reconfigurable Self-organization in Overlay Services.

2013-12 Marian Razavian(VU)
Knowledge-driven Migration to Services.

2013-13 Mohammad Safiri(UT)
Service Tailoring: User-centric creation of integrated IT-based homecare services to support independent living of elderly.

2013-14 Jafar Tanha (UVA)
Ensemble Approaches to Semi-Supervised Learning Learning.

2013-15 Daniel Hennes (UM)
Multiagent Learning - Dynamic Games and Applications.

2013-16 Eric Kok (UU)
Exploring the practical benefits of argumentation in multi-agent deliberation.

2013-17 Koen Kok (VU)
The PowerMatcher: Smart Coordination for the Smart Electricity Grid.

2013-18 Jeroen Janssens (UvT)
Outlier Selection and One-Class Classification.

2013-19 Renze Steenhuizen (TUD)
Coordinated Multi-Agent Planning and Scheduling.

2013-20 Katja Hofmann (UvA)
Fast and Reliable Online Learning to Rank for Information Retrieval.

2013-21 Sander Wubben (UvT)
Text-to-text generation by monolingual machine translation.

2013-22 Tom Claassen (RUN)
Causal Discovery and Logic.

2013-23 Patricio de Alencar Silva(UvT)
Value Activity Monitoring.

2013-24 Haitham Bou Ammar (UM)
Automated Transfer in Reinforcement Learning.

2013-25 Agnieszka Anna Latoszek-Berendsen (UM)
Intention-based Decision Support. A new way of representing and implementing clinical guidelines in a Decision Support System.

2013-26 Alireza Zarghami (UT)
Architectural Support for Dynamic Homecare Service Provisioning.

2013-27 Mohammad Huq (UT)
Inference-based Framework Managing Data Provenance.

2013-28 Frans van der Sluis (UT)
When Complexity becomes Interesting: An Inquiry into the Information eXperience.

2013-29 Iwan de Kok (UT)
Listening Heads.

2013-30 Joyce Nakatumba (TUE)
Resource-Aware Business Process Management: Analysis and Support.

2013-31 Dinh Khoa Nguyen (UvT)
Blueprint Model and Language for Engineering Cloud Applications.

2013-32 Kamakshi Rajagopal (OUN)

Networking For Learning; The role of Networking in a Lifelong Learner's Professional Development.

2013-33 Qi Gao (TUD)
User Modeling and Personalization in the Microblogging Sphere.

2013-34 Kien Tjin-Kam-Jet (UT)
Distributed Deep Web Search.

2013-35 Abdallah El Ali (UvA)
Minimal Mobile Human Computer Interaction.
Promotor: Prof. dr. L. Hardman (CWI/UVA)

2013-36 Than Lam Hoang (TUe)
Pattern Mining in Data Streams.

2013-37 Dirk Boerner (OUN)
Ambient Learning Displays.

2013-38 Eelco den Heijer (VU)
Autonomous Evolutionary Art.

2013-39 Joop de Jong (TUD)
A Method for Enterprise Ontology based Design of Enterprise Information Systems.

2013-40 Pim Nijssen (UM)
Monte-Carlo Tree Search for Multi-Player Games.

2013-41 Jochem Liem (UVA)
Supporting the Conceptual Modelling of Dynamic Systems: A Knowledge Engineering Perspective on Qualitative Reasoning.

2013-42 Léon Planken (TUD)
Algorithms for Simple Temporal Reasoning.

2013-43 Marc Bron (UVA)
Exploration and Contextualization through Interaction and Concepts.

2014

2014-01 Nicola Barile (UU)
Studies in Learning Monotone Models from Data.

2014-02 Fiona Tuliyano (RUN)
Combining System Dynamics with a Domain Modeling Method.

2014-03 Sergio Raul Duarte Torres (UT)
Information Retrieval for Children: Search Behavior and Solutions.

2014-04 Hanna Jochmann-Mannak (UT)
Websites for children: search strategies and interface design - Three studies on children's search performance and evaluation.

2014-05 Jurriaan van Reijsen (UU)
Knowledge Perspectives on Advancing Dynamic Capability.

2014-06 Damian Tamburri (VU)
Supporting Networked Software Development.

2014-07 Arya Adriansyah (TUE)
Aligning Observed and Modeled Behavior.

2014-08 Samur Araujo (TUD)
Data Integration over Distributed and Heterogeneous Data Endpoints.

2014-09 Philip Jackson (UvT)
Toward Human-Level Artificial Intelligence: Representation and Computation of Meaning in Natural Language.

2014-10 Ivan Salvador Razo Zapata (VU)
Service Value Networks.

2014-11 Janneke van der Zwaan (TUD)
An Empathic Virtual Buddy for Social Support.

2014-12 Willem van Willigen (VU)
Look Ma, No Hands: Aspects of Autonomous Vehicle Control.

2014-13 Arlette van Wissen (VU)
Agent-Based Support for Behavior Change: Models and Applications in Health and Safety Domains.

2014-14 Yangyang Shi (TUD)
Language Models With Meta-information.

2014-15 Natalya Mogles (VU)
Agent-Based Analysis and Support of Human Functioning in Complex Socio-Technical Systems: Applications in Safety and Healthcare.

2014-16 Krystyna Milian (VU)
Supporting trial recruitment and design by automatically interpreting eligibility criteria.

2014-17 Kathrin Dentler (VU)
Computing healthcare quality indicators automatically: Secondary Use of Patient Data and Semantic Interoperability.

2014-18 Mattijs Ghijsen (VU)
Methods and Models for the Design and Study of Dynamic Agent Organizations.

2014-19 Vincius Ramos (TUE)
Adaptive Hypermedia Courses: Qualitative and Quantitative Evaluation and Tool Support.

2014-20 Mena Habib (UT)
Named Entity Extraction and Disambiguation for Informal Text: The Missing Link.

2014-21 Kassidy Clark (TUD)
Negotiation and Monitoring in Open Environments.

2014-22 Marieke Peeters (UU)
Personalized Educational Games - Developing agent-supported scenario-based training.

2014-23 Eleftherios Sidirourgos (UvA/CWI)
Space Efficient Indexes for the Big Data Era.

2014-24 Davide Ceolin (VU)
Trusting Semi-structured Web Data.

2014-25 Martijn Lappenschaar (RUN)
New network models for the analysis of disease interaction.

2014-26 Tim Baarslag (TUD)
What to Bid and When to Stop.

2014-27 Rui Jorge Almeida (EUR)
Conditional Density Models Integrating Fuzzy and Probabilistic Representations of Uncertainty.

2014-28 Anna Chmielowiec (VU)
Decentralized k-Clique Matching.

2014-29 Jaap Kabbedijk (UU)
Variability in Multi-Tenant Enterprise Software.

2014-30 Peter de Kock Berenschot (UvT)
Anticipating Criminal Behaviour.

2014-31 Leo van Moergestel (UU)
Agent Technology in Agile Multiparallel Manufacturing and Product Support.

2014-32 Naser Ayat (UVA)
On Entity Resolution in Probabilistic Data.

2014-33 Tesfa Tegegne Asfaw (RUN)
Service Discovery in eHealth.

2014-34 Christina Manteli (VU)
The Effect of Governance in Global Software Development: Analyzing Transactive Memory Systems.

2014-35 Joost van Oijen (UU)
Cognitive Agents in Virtual Worlds: A Middleware Design Approach.

2014-36 Joos Buijs (TUE)
Flexible Evolutionary Algorithms for Mining Structured Process Models.

2014-37 Maral Dadvar (UT)
Experts and Machines United Against Cyberbullying.

2014-38 Danny Plass-Oude Bos (UT)
Making brain-computer interfaces better: improving usability through post-processing.

2014-39 Jasmina Maric (UvT)
Web Communities, Immigration and Social Capital.

2014-40 Walter Oboma (RUN)
A Framework for Knowledge Management Using ICT in Higher Education.

2014-41‘ Frederik Hogenboom (EUR)
Automated Detection of Financial Events in News Text.

2014-42 Carsten Eickhoff (CWI/TUD)
Contextual Multidimensional Relevance Models.

2014-43 Kevin Vlaanderen (UU)
Supporting Process Improvement using Method Increments.

Successful software process improvement initiatives do not only address the actual changes to a process (what changes?), but also the evolution of the process (how does it change?). Method increments are descriptions of the steps in this evolutionary path.

Despite previous research, the method increment concept is not well understood. In particular, we do not yet understand how we can use method increments to support process improvement efforts.

Throughout this dissertation, we describe several characteristics of incremental process improvement paths, elaborate the concepts needed to describe these paths, and develop tools to reuse knowledge related to method increments. This results in the design of a knowledge-centric software system to support incremental process improvement in software production.

Universiteit Utrecht

Department of Information and Computing Sciences