Improving Software Product Management Processes: a detailed view of the Product Software Knowledge Infrastructure

K. Vlaanderen

I. van de Weerd

S. Brinkkemper

Technical Report UU-CS-2010-025
July 2010

Department of Information and Computing Sciences
Utrecht University, Utrecht, The Netherlands
www.cs.uu.nl

ISSN: 0924-3275

Department of Information and Computing Sciences
Utrecht University
P.O. Box 80.089
3508 TB Utrecht
The Netherlands

Universiteit Utrecht

MBI Master Thesis

Improving Software Product Management Processes: a detailed view of the Product Software Knowledge Infrastructure

Author:	Kevin Vlaanderen, BSc
E-mail:	k.vlaanderen@cs.uu.nl
Student #:	0444367
Enrolled in:	2004

Period:	September 2009 - April 2010
Supervision:	Inge van de Weerd & Sjaak Brinkkemper
Thesis number:	INF/SCR-09-66

abstract Nowadays, a huge amount of software is being developed worldwide. Mainly among SMEs (small-to-medium enterprises), there are a lot of companies that focus on the development of product software. In order to manage software products effectively, implementing software product management processes in the organization is essential. Unfortunately, hardly any schooling is available in the area of SPM. In this thesis, a beginning will be made with the elaboration of a product software knowledge infrastructure (PSKI) that should help companies improve their product management process using method increments. The foundation for this elaboration lies in data about the current state of SPM, gathered from six product software companies. Also, two techniques will be elaborated that are a fundamental part of the PSKI. In addition to this, a core aspect of the PSKI is elaborated in a proof-of-concept. Combined, these deliverables form a strong basis for further research.

PUBLIC VERSION

June 17, 2010

Version Information

Version	Date issued	Remarks	Author
0.1	October 5th 2009	Set-up	Kevin Vlaanderen
0.8	May 6th 2010	First review-version	Kevin Vlaanderen
0.8	May 10th 2010	Review by Inge van de Weerd	Inge van de Weerd
0.9	May 21st 2010	Second review-version	Kevin Vlaanderen
0.9	May 26th 2010	Review by Inge van de Weerd	Inge van de Weerd
0.9	June 8th 2010	Review by Sjaak Brinkkemper	Sjaak Brinkkemper
1.0	June 9th 2010	Final version	Kevin Vlaanderen

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 The Product Software industry

Nowadays, a huge amount of software is being developed worldwide. 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 (Xu & Brinkkemper, 2005). Mainly among SMEs (small-to-medium enterprises), there are a lot of companies that focus on the development of product software. The change of 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 software product management processes in the organization is essential. Software product management (SPM) is "the process of managing requirements, defining releases, and defining products in a context where many internal and external stakeholders are involved" (Weerd et al., 2006a; Gorchel, 2000).

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 Weerd et al. (2006a) 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.

The framework for software product management has been well received within both the academic and the corporate world. Since its publication, several activities related to the framework have been organized, such as workgroups, a product management course, and a supporting website.

1.2 Product Software Knowledge Infrastructure

Unfortunately, hardly any schooling is available in the area of SPM. Product managers have often evolved into that function after first performing a development- or project management function. Because of this, a lot of product managers miss an essential piece of knowledge, required for performing their function effectively. In order to support product managers in implementing the correct software product management methods, we want to develop a knowledge infrastructure that serves this need.

The current software product management website[1] contains several kinds of information. First of all, it has an interactive version of the reference framework. By clicking on SPM activities, existing methods can be viewed. Examples of these are the prioritization of requirements according to the technique by Wiegers (2009), or the definition of a product roadmap according to the FastStart method Phaal et al. (2000). The website also contains several white papers, scientific articles and weblinks.

[1] http://softwareproductmanagement.org

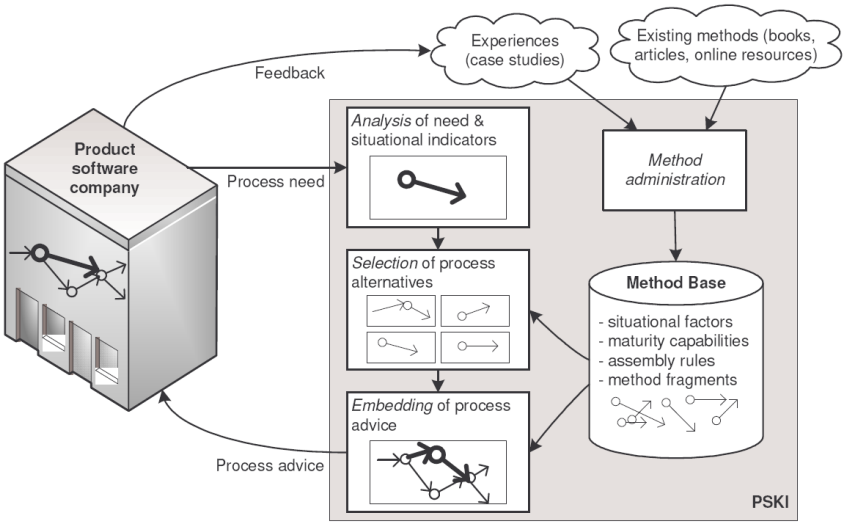


Figure 1.1: Product Software Knowledge Infrastructure (van De Weerd, Versendaal, & Brinkkemper, 2006)

The current website is a good start for spreading knowledge regarding software product management and for getting software product management at a higher level. However, in the current solution, several key items are missing. In 2006, van De Weerd, Versendaal, & Brinkkemper (2006) recognized that product software companies currently experience many performance failures in respect to their SPM process. To support product software companies in optimizing these processes, they developed a conceptual model of a product software knowledge infrastructure (PSKI). With this infrastructure, product software companies can obtain a custom-made advice that helps them to improve their processes. A schematic overview of the PSKI is illustrated in figure 1.1. The current activities are 'analysis of need and situational indicators', 'selection of process alternatives', 'embedding of process advice', and 'method administration'. All data is stored in a 'method base'.

Because of the complexity of the PSKI, its concepts and activities require further elaboration before the PSKI can be fully developed. The parts that make up the PSKI need to be described in more detail, exploring the scientific and industrial problems that need to be solved in the near future. What are the problems and issues that product software companies are actually experiencing? And how could a knowledge infrastructure solve such issues?

1.3 Contribution

In this thesis, a beginning will be made with the elaboration of the product software knowledge infrastructure mentioned above. The foundation for this elaboration lies in data about the current state of SPM, gathered from six product software companies. The processes and problems found in these case enhance our understanding of the field, enabling the design of an effective solution.

Also, two techniques will be elaborated that are a fundamental part of the PSKI. First, an alternative technique for performing assessments of the SPM process is proposed, combining SPM capabilities with PDD's. The resulting technique is then implemented in MetaEdit+. In addition to this, the generation of document templates based on method increments is elaborated in a proof-of-concept. Combined, these deliverables form a strong basis for further research.

Chapter 2

Research Method

The following sections will introduce the questions that drive this research, along with the underlying research approach. Section 2.1 describes the research questions that will be answered during the remainder of this thesis. Section 2.2 gives an overview of the steps that will be taken in order to answer these questions.

2.1 Research Questions

In order to evolve the PSKI from its current state to the envisioned state, many issues will need to be investigated. This research will be performed during a four-year PhD-track. However, before elaboration of the PSKI can begin, we need a better understanding of the possible contents of such a system. This thesis will lay the foundation for further research on the elaboration of a knowledge infrastructure that is capable of effectively helping product software companies improve their product management processes.

In essence, the research starts as a continuation of the research by van De Weerd, Versendaal, & Brinkkemper (2006). They tried to solve the problem of improving the product management process at software companies by answering the following question:

> How can product software companies improve the maturity of their product management processes using concepts of method engineering and situational capability maturation?

This resulted in the PSKI shown in figure 1.1. In this thesis, our understanding of such a system will be further enhanced by elaborating on the contents of the envisioned PSKI. Therefore, the main research question that is answered here is the following:

> **How can a knowledge infrastructure support and improve current SPM processes?**

Before this question can be answered, we need to look at the way in which product software companies currently are performing product management. As a tool for this assessment, we first propose a solution for combining the process-deliverable diagramming technique with the earlier mentioned capability matrix. This step answers sub-question one:

1. How can process-deliverable diagrams be used for assessing the maturity of a software product management process?

The resulting approach is then applied in six cases, in order to retrieve the answer to sub-questions two and three:

2. How is SPM currently performed in industry?
3. What are the major problems within software product management processes?

Based on the answer to the second and third sub-question, we can zoom in on the PSKI and further define its contents. The problems that are identified are used for elaborating each aspect of the current PSKI model. The fourth sub-question is therefore:

4. What should a knowledge infrastructure in the domain of Software Product Management look like?

Although the answer to sub-question four is already a more detailed view than shown in figure 1.1, it is still defined on a rather high level. For the actual implementation of the PSKI, each aspect of the proposed solution will need to be elaborated. In this thesis, we will start doing this by elaborating one of the core aspects of the system, method fragment storage. Thus, the fifth sub-question is:

5. How can method fragments be modeled and stored effectively?

We conclude by demonstrating one of the keystone aspects of the envisioned system, method increments. More specifically, we will demonstrate how method increments can be translated into updated company documents, based on the new process. The research question that guides this is the following:

6. How can templates be updated according to method increments?

The next chapter describes how these questions will be answered.

2.2 Research Approach

2.2.1 Design Science

As stated above, the aim of this scientific thesis is to establish a baseline for further research. The research questions posed in the previous section imply that we are investigating ways in which software product managers can be aided in their everyday practices. We are thus dealing with the creation of something new, or at least the preparation for this creation. To speak in the terms used by March & Smith (1995), we are *designing* an *artifact* that serves *human purposes*.

This classification originates from 1995, when IT started to attract significant scientific attention. It was recognized that the potential influence of IT was enormous, which could already be seen in the position IT had taken in the contemporary organizational setting. The scientific interest for IT was twofold; on the one hand, science tried to describe and explain phenomena related to IT, and on the other hand there was the belief that science could improve IT practices, resulting in prescriptive work.

Simon (1981) conceptualized this distinction with the terms 'natural sciences' and 'design sciences', where the former deals with the explanation of how and why things are as they are, and the latter deals with 'devising artifacts to attain goals' (Simon, 1981, p. 133). Later, Tsichritzis (1997) and Denning (1997) refined this definition in the context of IT-research by saying that the design-science paradigm seeks to 'create innovations that define the ideas, practices, technical capabilities, and products through which the analysis, design, implementation, and use of information systems can be effectively and efficiently accomplished.

When applied to IT-research, it is important to recognize certain interactions between these two types of research activity. Firstly, the phenomena caused by artifacts that are created through design science can serve as targets for investigation using a natural science approach. Secondly, the process of design can be aided by knowledge produced earlier through the application of natural science research. Thirdly, a synergy exists between the need for explanation of design research output on the one hand, and the verification or justification of natural science output (i.e. theories) on the other. Both of these requirements can be fulfilled by the application of research activities belonging to the alternative research type.

When viewed on a lower abstraction level, the two types of science consist of four basic activities. Firstly, the art of natural sciences consists of theorization and justification. Theorization means the explication of 'the characteristics of the artefact and its interactions with the environment that

result in the observed performance' (March & Smith, 1995). Justification is then the gathering of evidence to test the resulting theories. Secondly, the two main activities related to design science are the building and evaluation of artifacts. Artifacts are built in order to perform a specific task. By evaluation it is then determined whether the goals of the artifact have in fact been accomplished (March & Smith, 1995).

In table 2.1, these four main activities are depicted on the horizontal axis. On the vertical axis, you can see the four types of artifacts that design science produces: constructs, models, methods and instantiations. According to (March & Smith, 1995), constructs or concepts form the vocabulary of a domain, and a model is a set of propositions or statements expressing relationships among constructs. Methods can be defined as '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). Finally, an instantiation is the realization of an artifact in its environment (March & Smith, 1995).

		Research Activities			
		Build	Evaluate	Theorize	Justify
Research Outputs	Constructs				
	Model				
	Method				
	Instantiation				

Table 2.1: Information Technology Research Framework (redrawn from (March & Smith, 1995))

Based on this classification, the type of research that is performed in this work is design science. The main activity is construction or building, with the main result of this activity being an updated model for performing computer-aided method engineering in the domain of Software Product Management. Furthermore, a method for performing assessments is created, and a proof-of-concept (instantiation) is built.

To ensure a rigorous and relevant research approach, the work is performed and evaluated according to the conceptual framework proposed by Hevner et al. (2004). The framework is shown in figure 2.1.

The environment in which this research is positioned, is formed by product software *organizations* that try to improve their software product management *processes*, based on the *capabilities* of the *people* that perform the *role* of product manager, and on the *characteristics* of the organization as a whole.

When analyzing the knowledge base that is input for this research, the work in this thesis is based on existing *theories* on SPM and method engineering. The work is strongly founded on the software product management reference *framework* (Weerd et al., 2006a,b), and *constructs* created in the domains of method-engineering and software product management. Also, the Process-Deliverable *Diagram* (Weerd, 2005) is repeatedly used to convey research findings, and forms the basis for one of the major artifacts of this work.

The environment and the knowledge base are the two origins of input for the research that is performed in this work. During this work, several *artifacts* are created. Firstly, an **assessment method** is created that allows the combination of modeling and analyzing a company's SPM process in one task. Secondly, the software product management processes of several companies are modeled in the form of a set of **PDDs**. The results are *assessed* by experts from these companies. Based on the results, a proposal is made for a **computer-aided method engineering (CAME) method** in the domain of software product management. Finally, the thesis will describe **a solution for the storage of Process-Deliverable Diagrams** and **a proof-of-concept for method increments**.

Along with the framework in figure 2.1, Hevner et al. (2004) propose the following set of guidelines:

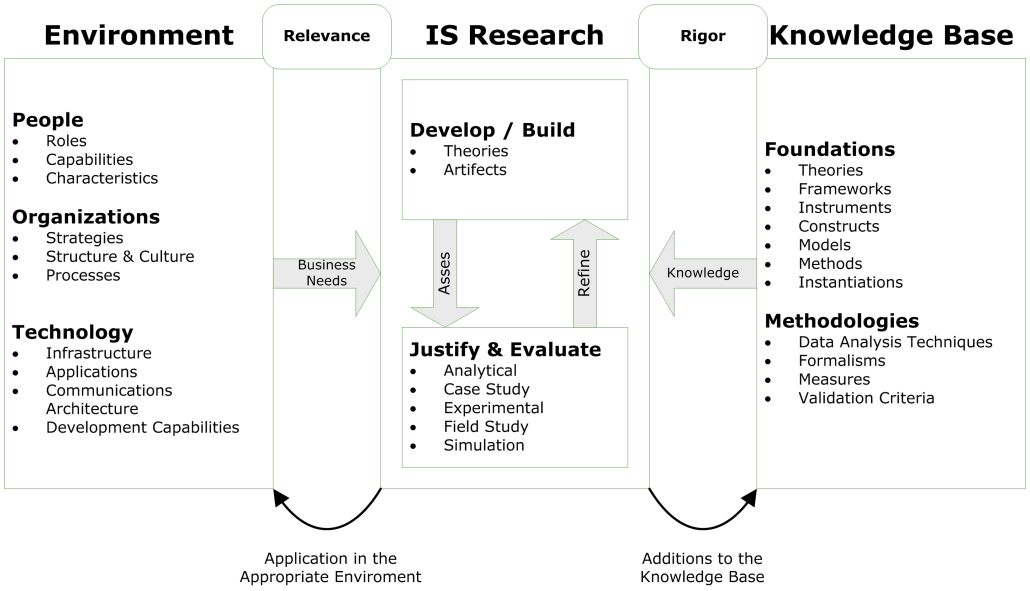


Figure 2.1: Information Systems Research Framework (redrawn from (Hevner et al., 2004))

1. Design as an Artifact: Design-science research must product a viable artifact in the form of a construct, a model, a method, or an instantiation.
2. Problem Relevance: The objective of design-science research is to develop technology-based solutions to important and relevant business problems.
3. Design Evaluation: The utility, quality , and efficacy of a design artifact must be rigorously demonstrated via well-executed evaluation methods.
4. Research Contributions: Effective design-science research must provide clear and verifiable contributions in the areas of the design artifact, design foundations, and/or design methodologies.
5. Research Rigor: Design-science research relies upon the application of rigorous methods in both the construction and evaluation of the design artifact.
6. Design as a Search Process: The search for an effective artifact requires utilizing available means to reach desired ends while satisfying laws in the problem environment.
7. Communication of Research: Design-science research must be presented effectively both to technology-oriented as well as management-oriented audiences.

As several artifacts will be produced during this research, guideline 1 is met. Although the main product, the proposal for a computer-aided method-engineering method, is not a full-grown information system, it shows the ideas, technical capabilities and products by which the system can be effectively and efficiently accomplished (Tsichritzis, 1997; Denning, 1997). This also ensures the implementation of guideline 4.

Guideline 2, problem relevance, is met by the modeling of several software product management processes, as performed by a set of product software companies. Through these models, problematic aspects in the process can be identified. The solution proposed in this work will be relevant, as it is based on these problems.

Since the solution proposed in this work will be of a conceptual nature, no extensive evaluation can yet be performed (guideline 3). However, the design will be evaluated according to the problems found during the case-studies, to determine whether it solves all the problems that have been raised.

Guideline 4 is met by increasing our level of understanding regarding software product management methods, the problems that are being experienced, and how these can be solved using method increments.

Research rigor (guideline 5) is obtained by appropriate usage of existing research work in the field of software product management and method engineering. The solution will rely heavily on existing, validated constructs. Furthermore, rigor is accomplished by performing the empirical part of the work (case-studies) according to validated and well-known research methods (Yin, 2003; Jansen & Brinkkemper, 2008).

Guideline 6 is met implicitly, as this work elaborates on an earlier proposal (Brinkkemper, 1996; van De Weerd, Versendaal, & Brinkkemper, 2006).

To conclude, guideline 7 is met by the extraction of knowledge from this work, for publication at both academic as well as professional forums.

2.2.2 Research Steps

Figure 2.2 shows an elaborate plan for the work performed in this thesis. As with any project, the process started with the writing of a proposal, which included the planning and the research approach described here. In the next step, related literature in the fields of software product management, method engineering, domain-driven software development and component-based development was analyzed, resulting in the part called 'Theoretical Background' (chapter 3).

After these two sections, the actual research could be performed, as described by the four following activities. First of all, an approach was developed to assess product management processes through employing process-deliverable diagrams (chapter 4). This work was accompanied by the elaboration of an approach for storing method fragments, described in chapter 7. Case studies were then performed using this approach, in order to gather valuable data from industry (chapter 5). Based on the results from the literature study and the case studies, a more detailed vision of the Product Software Knowledge Infrastructure was then proposed (chapter 6). Finally, part of this model has been elaborated into a proof-of-concept prototype, focussing on method increments. This is described in chapter 8. The thesis ends with a summary of the main results, and a proposal for further research (chapter 9).

2.2.3 Deliverables

The research approach described above results in a set of deliverables:

Case study database. All data related to the case studies is stored in the case study database. This database is the repository of all information input to and produced by the case study. It contains the following deliverables per case:

- **Four process-deliverable diagrams.** For each of the focus areas in the reference framework for SPM, a PDD was drawn based on the findings at the case company. Also, activity and concept tables were created based on the PDD's.
- **Capability matrix.** Based on the PDD's, a capability matrix was generated for each case company. This capability matrix formed the basis for the advice report.
- **Advice report.** For each case, an advice report was written that describes the research methods used, the major bottlenecks that were found, and a set of recommendations.

Assessment-technique for SPM processes. The process-deliverable diagramming technique has been updated to incorporate the capabilities from the capability-matrix. This allows for automatic assessment of a company's SPM process based on process descriptions.

Detailed vision of the PSKI. Based on the findings of the case studies, a more updated view of the PSKI has been created. It is described in the form of a set of PDDs, along with a narrative description.

Method fragment storage technique. An implementation of PDDs in MetaEdit+ has been developed. This technique can be used to create PDDs in a more user-friendly and efficient manner. Also, it allows exporting to XML and latex.

Method increment proof-of-concept. To demonstrate the feasibility of combining method increments and template generation using web-techniques, a proof-of-concept has been created.

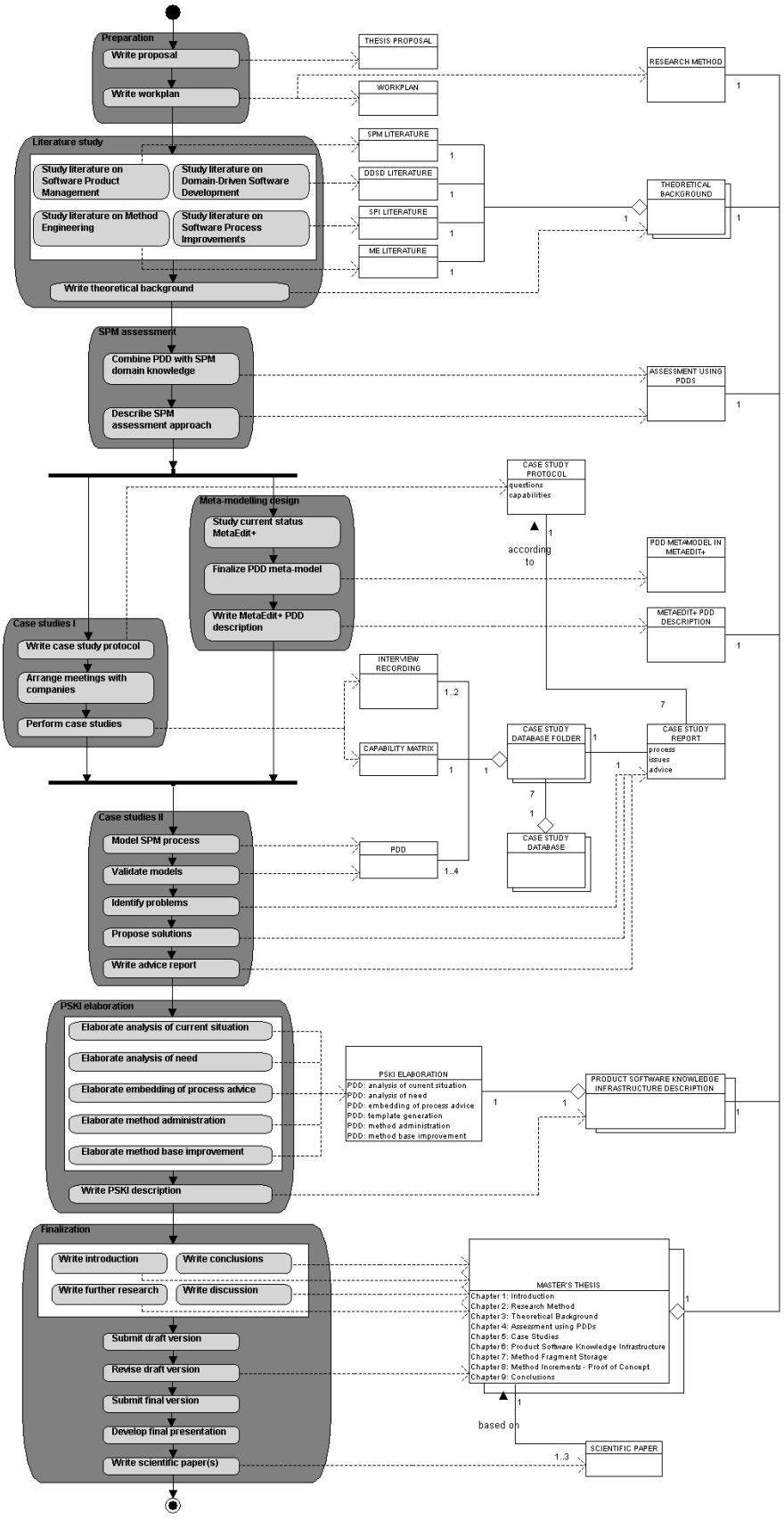


Figure 2.2: Planning of the steps related to this thesis

Chapter 3

Theoretical Background

The following sections will summarize relevant academic work related to software product management and method engineering. Section 3.1 describes the approach that was taken during the literature analysis. The remainder of the sections contain the results of this analysis per research area.

3.1 Analysis Approach

A firm knowledge of the current situation of the various related research fields is required for proper elaboration of the research questions. In this section, an overview is given of the state-of-art of all related fields, insofar they are applicable to the scope of this thesis. To guide the literature analysis, a classification scheme has been used that groups all related literature according to a two-dimensional matrix. This matrix is shown in table 3.1. On the vertical axis, all related fields are identified. On the horizontal axis, the literature is classified according to three categories; theory, practice and overview.

The classification scheme has been used during the initial phase only, to keep the long list of literature manageable. The numbers next to each code indicates the number of papers/books that have been assigned that code. Nearly all of the mentioned works (excluding those marked as 'other') have been read and used during the literature analysis. The remaining works have been used in some other way in these thesis, i.e. as a general reference or for illustration purposes.

The category named 'theory' contains theoretical discussions on the current state of affairs of a specific topic. This type presents conceptual solutions for problems, that are novel or significantly improve existing solutions.

The 'practice'-category consists of practical solutions to current research problems. This includes methods and techniques, validated through empirical means. This category also includes industrial practice and experience papers.

The third category, 'review', deals with abstractions from the current state-of-art and provides insightful observations or fruitful analogies. This type of literature is considered to provide an overview of the current state of art. It discusses a collection of academic material on a specific topic.

For each of the combinations <category/type>, a shorthand code was assigned.

During the process of the literature review, a structured approach was followed by repeatedly performing a set of steps. The first step for each possibly related paper was to read the abstract (and in some cases the introduction) and to classify the paper based on this. The classification consisted of one or more of the shorthand codes described above.

In the case of a theoretical paper, the most important statements within the text were highlighted. These points were transcribed and used during the writing of the section on related literature. In the case of a practical paper, notes were made of the main positive and negative aspects. Review papers were mainly used in the search for additional research material.

As a final step, the reference list of each related paper was scanned for additional articles and papers. When found, these were noted and processed in the same way.

In the next sections, each of the mentioned research areas, except 'other', will be elaborated.

	Theory	Practice	Review
Software Product Management			
— General	SPM-T (10)	SPM-P (2)	SPM-R (0)
— Portfolio Management	PM-T (7)	PM-P (4)	PM-R (1)
— Product Roadmapping	PR-T (3)	PR-P (4)	PR-R (1)
— Requirements Engineering	RE-T (22)	RE-P (13)	RE-R (0)
— Release Planning	RP-T (12)	RP-P (6)	RP-R (0)
— Assessment	A-T (4)	A-P (2)	A-R (0)
Method Engineering			
— General	ME-T (21)	ME-P (15)	ME-R (1)
— Meta-Modeling	MM-T (6)	MM-P (4)	MM-R (0)
— Method Fragment Selection	MFS-T (0)	MFS-P (1)	MFS-R (0)
— Method Fragment Assembly	MFA-T (2)	MFA-P (2)	MFA-R (0)
— Method Assessment	MA-T (2)	MA-P (2)	M-R (0)
Other			
— General	O-T (25)	O-P (29)	O-R (3)

Table 3.1: Literature Framework

3.2 Software Product Management

3.2.1 Product Software

Ever since the late 1960's, many software development companies have made the shift from creating customer-specific software to market-oriented standard software. During these years, many names have been proposed for this type of software. Among these definitions are shrink-wrapped software, (complex) Common-Of-The-Shelf software, packaged software and commercial software. For the sake of clarity, this thesis will make use of the definition posed by Xu & Brinkkemper (2005), by using the term 'software product' in the sense of a "packaged configuration of software components or a software-based service, with auxiliary materials, which is released for and traded in a specific market".

3.2.2 Software Product Management Reference Framework

The reference framework mentioned earlier (and shown in figure 3.1) is subdivided into four focus areas: portfolio management, product roadmapping, release planning and requirements management. Portfolio management is the top-level and comprises decisions regarding the portfolio of products and product families. Product roadmapping is the second process area. Processes that fall within this category are the identification of themes and core assets, with the aim of developing and maintaining a roadmap for a certain product. Requirements management includes the collection, identification and organization of requirements. Finally, release planning includes the processes of prioritizing and selecting requirements for the new release, validating and launching the release, and managing scope changes.

Positioned around the framework are nine stakeholders, of which six are internal and three are external stakeholders. The definition of the stakeholders' responsibilities is as follows (Weerd et al., 2006a):

Company board (internal) is responsible for the definition and communication of strategy, vision and mission to the rest of the company. Also, it has the managerial supervision of the different departments, including product management.

Research&innovation (internal) has two core responsibilities, namely 1) doing research to new opportunities for product innovations, and 2) finding new ways to incorporate improvements

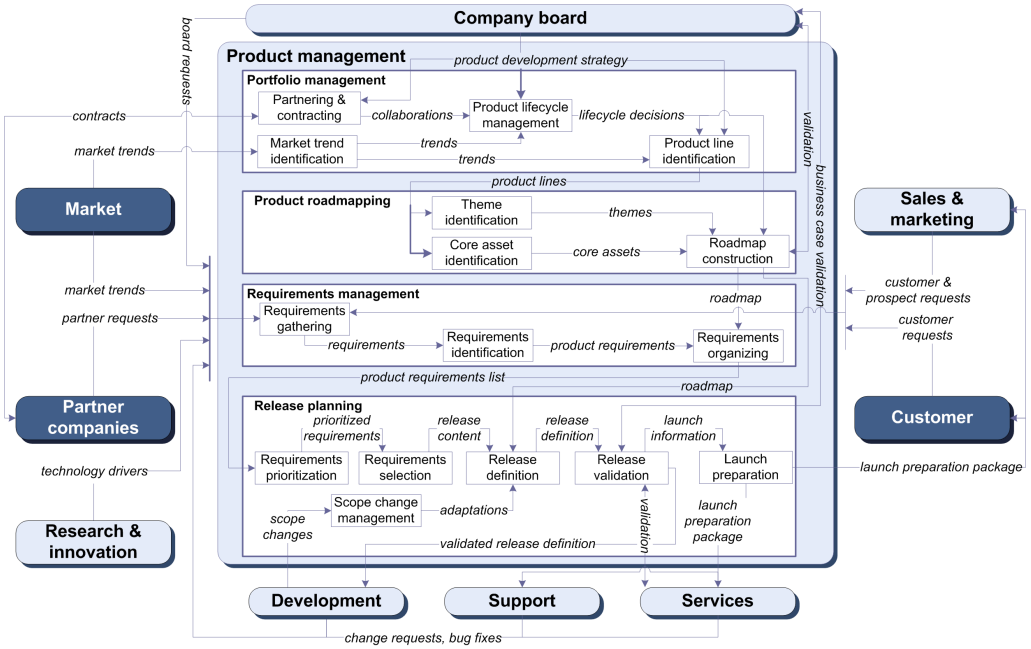


Figure 3.1: Reference Framework for Software Product Management

or new features into the existing products.

Services (internal) includes consultants who are responsible for the implementation of the software product at the customer organization.

Development (internal) has as main responsibility the execution of the release plan.

Support (internal) stands for the helpdesk to answer questions and for small defect repair unit.

Sales&marketing (internal) is the first contact with a potential customer.]

The market (external) is an abstract stakeholder, standing for potential customers, competitors and analysts.

Partners (external) , including implementation partners, who implement the product at a customer, development partners, with whom product components are developed, and distribution partners, selling the product.

Customers (external) often have new feature requests in the process of closing the deal or during the usage of the product.

The next sections will describe the main areas within Software Product Management in more detail.

3.2.3 Portfolio Management

Portfolio management is the top layer within the framework, and consists of activities that are influenced strongly by the company board, the market and partner companies. Activities that are identified in this layer are 'partnering&contracting', 'market trend identification', 'product lifecycle management' and 'product line identification'. Although this focus area has for a long time been neglected in literature related to product software companies, the activities have recently received significant scientific attention.

Product line identification entails the identification of sets of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way (Weerd

et al., 2006a; Clements & Northrop, 2001). The area has gained significant interest during the last decade (Abramovici & Soeg, 2002; Ardis et al., 2000; Ebert & Smouts, 2003; Ebert, 2006). In most cases, the benefits of software product lines are related to financial aspects, leading to much attention to cost-based approaches (Khurum et al., 2008). Several tools and techniques have been proposed to aid in product line identification (Yamazaki, 2009; Fricker & Stoiber, 2008). Many of these methods are fairly similar on a high level, but show significant differences on a lower level. Within product line management, the notions of variability and configuration management hold an important position (Gurp et al., 2001; Jansen, 2007; Ali et al., 2009).

Product lifecycle management, or the management of product-related information and knowledge within an enterprise throughout the entire product lifecycle (Weerd et al., 2006a; Abramovici & Soeg, 2002), has been discussed in a number of books and articles during the last few years. Examples include works on general product lifecycle management, as by Sääksvuori & Immonen (2008) and Sudarsan et al. (2005), and works with a focus on software products, such as Grieves (2005), Stark (2005) and Kiritsis et al. (2003).

The third activity identified as part of portfolio management, market trend identification, deals with the extraction of valuable information about competitors and possibilities from the market. The topic already gained wide attention many years ago, with the introduction of lead user management (Hippel, 1986). More recently, attention has been paid to market cycles and trends (Aguiar & Gopinath, 2007).

The final activity within portfolio management, partnering&contracting, has received a lot of attention during recent years, mainly in the form of discussions of outsourcing. When talking about outsourcing, trust is one of the major themes, and the subject of many research projects (Oza et al., 2006; Babar et al., 2007; Nguyen et al., 2006; Siakas et al., 2006). More practical solutions are described by for instance Carmel & Abbott (2006) and Damian (2007). For an overview of outsourcing related literature, see the work by Gonzalez et al. (2006).

3.2.4 Product Roadmapping

Product roadmapping deals with the expectations, plans, themes and core assets of a software product (Weerd et al., 2006a; Moon & Yeom, 2004). Activities within this area are 'theme identification', 'core asset identification' and 'roadmap construction', of which the last activity results in the main document; the roadmap. That it is considered an important planning tool can be seen from the large amount of scientific material that has appeared on the topic. Recent examples elaborate on collaboration during the roadmapping process (Jantunen & Smolander, 2006), perspectives on roadmapping (Kappel, 2001), and more practical approaches such as by Vähäniitty et al. (2002) and Regnell, Svensson, & Olsson (2008).

The two supporting activities vary in the amount of debate around them. In the case of theme identification, whose output is used for steering the roadmap creation and aiding in the planning of releases and their contents, not much has been written in the software development domain. In the business domain, some publications can be found. One example is the term 'enduring business theme' (Cline & Girou, 2000), although this is more a combination of themes and core-assets.

More work can be found on core asset identification, which refers to the complex components that are shared by multiple products (Weerd et al., 2006a). These components need to be identified and management, as they can play an important role during planning and optimization. In the domain of product line engineering, core assets are a common concept. K. Lee et al. (2002); J. Lee et al. (2004) described the role of core assets in feature-oriented modeling and engineering. The most important use of core-assets, e.g. reuse, was described by Moon & Yeom (2004) and Tomer et al. (2004).

3.2.5 Requirements Management

Without a doubt the area that has received the most attention in scientific and managerial literature during the last few decades is the area of requirements management. Well before the advent of product software or software in general, the gathering and management of customers' demands was recognized as an import factor for the success of a product, in which many risks were to be found (Lawrence et al., 2001). Initial scientific research focused mainly on the management of

requirements in the context of software projects. However, Potts (1995) already recognized that market-driven software companies are in need of a different requirements management approach.

Within the reference framework, the area of requirements management has been divided into three main tasks: requirements gathering, related to the elicitation of requirements from stakeholders, requirements identification, dealing with the rewriting and restructuring of requirements, and requirements organization, which focuses on the grouping of requirements per product or core asset (Weerd et al., 2006a).

Recently, many approaches have been developed to tackle this problem. Many of these took into account the large amount of requirements that product software companies often have to deal with (Regnell, Svensson, & Wnuk, 2008; Wnuk et al., 2009; Khurum et al., 2007). Also in this light, Gorschek et al. (2007) have proposed an abstraction model to aid decision making when dealing with many requirements.

Also, dealing with a steady increase in agile development practices has lead to some re-evaluation of requirements management practices (Cooper, 2006). Two cases by Pichler et al. (2006); Vlaanderen et al. (2009) show how product managers can actually deal with a large amount of (big) requirements in an agile environment.

On a more abstract level, both understandability as well as quality have been discussed in recent literature. Svahnberg et al. (2008); Dzamashvili-Fogelström & Gorschek (2007) have discussed quality and understandability in relation to requirement definitions, while Gorschek & Davis (2008) focused on the assessment of the requirements management process as a whole.

Also on a more strategic level can we find several works relating to requirements management (Fricker, 2005). For instance, some work has been performed related to a goal-oriented approach for requirements communication (Fricker et al., 2008) and negotiation (Fricker & Grunbacher, 2008; Fricker, 2007).

3.2.6 Release Planning

The final focus area within the reference framework is release planning. At the same time, this is the most encompassing area, dealing with a wide variety of tasks, starting at requirements prioritization and ending at launch preparation. In general, the area can be described with the definition of software release management, or the process through which software is made available to, and obtained by, its users (Weerd et al., 2006a; Hoek et al., 1997).

In total, six main activities are identified within the framework. 'Requirements prioritization' deals with assigning priorities to requirements based on stakeholder input. During 'requirements selection', the requirements that will be implemented for the next release are picked, which is then described during 'release definition'. After the 'release validation', the release definition can be handed over to development for implementation. During the process, changes are handled through 'scope change management'. Finally, 'launch preparation' is related to all the tasks that need to be performed before an actual release, such as stakeholder negotiation, training, etc.

Especially related to requirements prioritization, many techniques have been developed during the years. Examples include the incorporation of stakeholder opinions (Ruhe & Saliu, 2005), the analytical hierarchical process (Saaty, 1980) and linear programming (Akker et al., 2005). Other approaches are described by Svensson et al. (2008); Fehlmann (2008); Svahnberg & Karasira (2009), including an approach for re-prioritization by Racheva et al. (2008).

3.2.7 Assessment

As software development process evolved over the last few decades, several approaches have emerged for assessing the competences of organizations in applying these processes. Examples of this include the approach by Appel (2000) for architectural capability assessment. For one of the sub-fields of SPM, requirements engineering, the Capability Maturity Model for software (Paul et al., 1993) and its successor, the Capability Maturity Model Integration (CMMI Product Team, 2002), have been developed. More examples can be found in the article by Bekkers et al. (2010).

Though the role of product management is becoming increasingly important in product software companies (Ebert, 2007). Nonetheless, until recently no such assessment method had been developed for it. In the context of the Product Software Knowledge Infrastructure, with its aim

of product management maturation, Bekkers et al. (2010) have proposed an approach to fill this gap. It combines multiple descriptions of an organization and its product management process by performing an analysis of the organization's situational factors (Bekkers et al., 2008) and its product management capabilities. The results of the method are described as solution oriented and realistic, allowing for incremental growth and requiring little effort to obtain.

An important part of the approach by Bekkers et al. (2010) is the capability matrix. This matrix shows all the activities from the reference framework for SPM on the vertical axis, and all the associated capability levels on the horizontal axis. Based on the current state of a company's process, its maturity level can be derived from the matrix. (Weerd et al., 2009)

3.3 Method Engineering

The term 'methodology engineering' was introduced in 1992 by Kumar & Welke (1992), referring to the engineering of information systems development methods, taking into account the uniqueness of a project situation. In his work, Brinkkemper (1996) introduces the similar 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 defined as "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" and a methodology refers to 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 & Welke, 1992), a special type of method engineering has been introduced; situational method engineering. Harmsen et al. (1994) define a situational method as "an information systems development method tuned to the situation of the project at hand". One of the ways of transforming methods into situational methods is by the use of route-maps, introduced by Slooten & Brinkkemper (1993). According to Slooten & Hodes (1996); Aydin & Harmsen (2002), route maps can be used to tune methods into situational methods by using different routes to represent different situations.

During the last several years, several modularization constructs have been proposed for situational method engineering. Although these constructs have many aspects in common, some essential differences exist. The six main constructs are 'method fragments', 'method chunks', 'method components', 'method services', 'OPF fragments' and 'FIPA fragments'. Extensive comparisons of these constructs were performed by Cossentino et al. (2006), Agerfalk et al. (2007) and Deneckere et al. (2008).

3.3.1 Method Fragments

Method fragments (Harmsen et al., 1994) are defined as "... a description of an IS engineering method, or any coherent part thereof". Each method fragment can be classified on three orthogonal dimensions: perspective, abstraction level, and layer of granularity. The method fragments consist of a process part and a product part, with the addition of a link between these two parts.

For the modeling of method fragments, an approach is used based on the proposal of Saeki (2003). The technique uses a combination of two UML diagrams (Weerd & Brinkkemper, 2008):

- On the left hand side, an adaptation of the UML activity diagram is used for meta-process modeling. This diagram consists of activities and transitions. The activities can be unordered, sequential, concurrent or conditional.

- On the right hand side an adaptation of the UML class diagram, a concept diagram, is used for meta-data modeling. This diagram consists of concepts, generalizations, associations, multiplicity and aggregations.

Within the process-deliverable diagram, the two models are integrated in a straightforward way, by connecting every activity with its associated artifact through a dotted line.

PDD's are used within this thesis to model the envisioned PSKI. Examples can thus be found in chapter 1.2 (figures 6.2, 6.3, etc.).

Method fragments can stored in a method base, using the MEL method engineering language (Brinkkemper et al., 2001). Once retrieved from the method base, they can be combined following the assembly rules described by Brinkkemper et al. (1999).

More recently, work has been performed on allowing incremental method evolution (Weerd et al., 2007). According to this work, method fragments can be used to describe and improve the evolution of software product management methods, by allowing the insertion, modification and deletion of method fragment components.

3.3.2 Method Chunks

The second main construct for method engineering is the method chunk (Ralyté & Rolland, 2001; Ralyté et al., 2003). Method chunks are designed to contain a more comprehensive situational description. Chunks consist of two parts: the method knowledge in the form of the method chunk body and interface, and the meta-method knowledge in the form of the method chunk descriptor. Method chunks are in essence process-driven, as they contain partial descriptions of the method process model in the form of reusable guidelines.

A guideline can be either simple, tactical or strategic. This distinction exists to describe different levels of formality, granularity, etc. Its interface describes the situation in which a chunk can be applied. This is represented in the form of the input and the intention of the method chunk. Related to this is the method chunk body, describing how this intention can be achieved. It consists, similar to method fragments, of a process and a product part. However, in this case they are described in the form of NATURE context trees (Jarke et al., 1999) or MAP graphs (Rolland et al., 1999). MAP graphs describe a method fragment using intentions, and strategies to reach these intentions.

Related to the guideline, but on a higher level, the descriptor provides information about the context of the method chunks. In the descriptor are described such things as the id, the name, the type, and the domain of the method chunk.

To guide the description of existing methods by method chunks, Ralyté & Rolland (2001) have proposed an approach for method re-engineering. This has been extended with have introduced a generic process model for situational method engineering (Ralyté et al., 2003). The proposal contains three approaches:

- the assembly of method chunks, where components of existing methods are extracted and placed in a method base
- the extension of an existing method, by applying extension patterns
- the generation of a method by abstraction/instantiation of a model/meta-model

These approaches should aid method engineers in the construction of new methods.

3.3.3 Method Components

The situationality described by method chunks is also to be found in method components (Wistrand & Karlsson, 2004; Karlsson & Wistrand, 2006). However, even more than with method chunks, method components are supposed to be independent, exchangeable and reusable. Apart from the process and product description also found in the previous two constructs, method components also define a notation. Furthermore, components heavily rely on a rationale, describing why and when the component can and should be used.

This rationale, also described by Rossi et al. (2004); Agerfalk & Fitzgerald (2006), is formed by goals that describe the reasons for which a method element exists. Attached to this are values of the method creator.

3.3.4 Method Services

In 2007, Rolland (2007) suggested a move towards a more service-oriented approach for method engineering. Such an idea was also posed by Guzélian & Cauvet (2007). The idea was elaborated by Deneckere et al. (2008), who suggested method services, using a Service-Oriented Architecture (SOA). Just as method chunks, method services are described by an intention, and a description of how and why they should be used. Also, they contain a product- and process aspect.

The idea of the adapted SOA, renamed into Method Oriented Architecture, is that method chunks are published, through a method provider, in a method registry. From there, they can be used by method clients. Through the use of several standards, such as XML, WSDL and BPEL, such an infrastructure should be created.

3.3.5 OPF Method/Process Components

A slightly more rigid approach is adopted with the OPEN Process Framework (Henderson-Sellers, 2002). Based on the OPEN process meta-model (Graham et al., 1997; Henderson-Sellers et al., 1998), the process components used in the OPEN process framework contain a structure that is fairly similar to the solutions we have seen earlier. Here, components consist of producers who produce work products through work units. Components are organized by stages and documented by languages. Also, guidelines are used to steer the execution/implementation of components.

3.3.6 Method Configuration

Another type of method engineering is method configuration, introduced and used by Karlsson (2002) and Karlsson & Å gerfalk (2004). With method configuration, one method is chosen, which is then adapted to the situation at hand.

More recently, research has been done to create a method engineering process for the construction of an implementation method for web-based CMS applications. van De Weerd, Brinkkemper, et al. (2006) introduce an assembly-based situational method engineering approach for this purpose. The approach consists of four steps: (a) identification of implementation situations, (b) selection of candidate methods, (c) analysis and storage of relevant fragments in the method base, and (d) assembly of the new method using route maps to obtain situationality. There steps have in turn formed the basis for the conceptualization of the knowledge infrastructure for incremental process improvement in SPM, as described in chapter 1.2.

Chapter 4

Assessment using Process-Deliverable Diagrams

The process-deliverable diagram created by Weerd & Brinkkemper (2008), based on the work by Saeki (2003), is a strong instrument for the description and communication of software development processes. The two combined views, process and deliverable, provide an easy-to-use and clear view of all the main parts of a process. We have seen very successful applications of it in several projects. This includes the visual description of method increments by Weerd et al. (2007).

The positive experiences with applying PDD's for describing product management processes suggests that its use might be extended. Given the nature of the modeling technique, describing both the process and its deliverables, it potentially matches very closely the *capabilities* as described by Bekkers et al. (2010). By combining PDD's with the capability matrix, one can create a modeling technique that both captures a company's SPM process and its maturity at the same time.

Such a solution would offer various advantages. For one, combining the two tasks saves a considerable amount of time. Furthermore, it potentially enhances reliability, as the process description and assessment are both based on the same data. Also, it forgoes the problem experienced with performing assessments using the capability matrix, where product managers are prone to answer 'yes' to questions that, according to the accompanying rules, should be registered as 'no'.

In the next chapter, the approach for combining PDD's and the capability matrix is described in further detail.

4.1 Combining PDD's with the Capability Matrix and the Reference Framework for SPM

In order to combine the reference framework for SPM and the capability matrix with the PDD modeling technique, we need a way to attach semantic information to the activities and deliverables in the diagrams. In order to do so, we need a clear understanding of what each construct in the PDD means. For deliverables, this is very clear, as differences in abstraction level are clearly indicated using generalizations and compositions. However, with activities, this distinction is not so clear.

In most cases, the differences between simple and complex activities are used to indicate differences in granularity level. Simple activities indicate an activity at the lowest level of granularity within the scope of the diagram. Complex activities are of a higher level, with their sub-activities again indicating low-level activities. Complex activities can in most cases be interpreted as 'phases'.

In this set-up, the use of complex activities is basically a grouping mechanism for somewhat related tasks, as there are no rules governing the naming or application of complex activities. A good motivator for this way of working has always been the enhanced readability of the diagrams. However, as we are now aiming towards more complex and comprehensive applications, readability might no longer be the chief motivation.

In fact, a more structured approach is more in its place here. The basis for finding such

a structured approach was the proposition that the PDD should be context- or domain-aware. This means that existing domain knowledge should be used during the construction of process-deliverable diagrams. In the domain of software product management, this domain-knowledge is readily available in the form of the reference framework for SPM (section 3.2.2) and the capability matrix (section 3.2.7). These artifacts provide structure in the form of focus areas, processes and capabilities.

There are of course several ways of applying this structure, ranging from the use of guidelines based on the knowledge contained in the reference framework and capability matrix, to a restrictive, structured approach. The three most important alternatives are the following (all three differ mainly in their approach to process modeling):

A The modeling of activities is not bound to strong rules. Small (simple) activities can be bundled in complex activities, to enhance the readability of the diagram. This grouping has no semantic consequences.

Advantages :

- The user has a lot of freedom.
- The solution is straightforward and thus easy to implement.
- The solution does not differ from the current approach.

Disadvantages :

- Activities do not contain semantic information regarding the processes and capabilities.
- The approach is prone to errors and inconsistencies.
- The solution is difficult to employ for computational purposes.

B Main activities are used to model processes from the reference framework. Smaller activities can be freely placed within those processes. Every main activity has an attribute 'process'. Activities that do not implement an SPM process are modeled top-level as simple activities.

Advantages :

- Semantic information regarding implemented processes is available.
- Increased structure makes it easier to read the diagrams.
- The solution is more suitable for computational purposes.

Disadvantages :

- No semantic information regarding implemented capabilities is available.
- The user is somewhat restricted in his/her freedom.
- The notation is somewhat complicated if the process is indicated in the diagram.

C Activities are used to model implemented capabilities. In some cases, an activity can contain sub-activities, but this should often not be necessary. Most activities have in principle a set of 'process/capability'-combinations that they implement.

Advantages :

- Semantic information regarding both the implemented processes as well as their respective capabilities is available.
- The solution is highly suitable for computational purposes.

Disadvantages :

- The user is even more restricted during modeling than with solution B; furthermore, he/she has to decide for every activity which process/capabilities it implements.
- The notation is somewhat complicated when both the implemented processes as well as the capabilities are shown in the diagram.

All three options produce almost identical diagrams. However, there are some significant visual implications. In the second case, we obtain diagrams fairly similar to the old situation (situation A), but with a more structured approach towards the use of complex activities. Therefore, readability is maintained and even enhanced by the structure introduced.

On the other hand, the notation is somewhat complicated by the display of the process attribute. Also, freedom is somewhat restricted by the added structure. In addition, the biggest disadvantage is the fact that this approach does not fully utilize the structure provided by the reference framework and capability matrix.

This issue is solved by the third approach. Information regarding maturity levels is added to the model, making it semantically much more informative. A second implication is that the use of complex activities is diminished significantly. Only capabilities that are achieved by performing multiple other activities should be displayed as a complex activity. In practice, this does not happen often.

Reducing the use of complex activities makes the diagram less complex but also less structured and therefore perhaps less readable. This issue is enhanced by the fact that both processes as well as capabilities now need to be displayed.

Given above arguments and counter-arguments, option C is more suited due to the goal that we are currently pursuing. The increased semantics make this the best choice. Also, the diagrams in the advice reports (see also chapter 5) show that readability is not impaired significantly.

To indicate the processes and capabilities that an activity implements, a set of two arguments is added to each activity; a process and the associated capability that is implemented by the activity. Although displaying this information in the diagram is not entirely necessary since we are dealing with meta-meta-information, the communicative power of the PDD is enhanced when append with an indicator. To maintain readability, the indicator is not shown in its full form. Instead, it is shown in the form [process-acronym]:[capability][- ...].

4.2 Performing Assessments

Currently, the assessment of a product management process using the situational assessment method (Bekkers et al., 2010) takes a very direct approach. During one or more interviews, experts are asked wich of the capabilities are being performed in their company. This is done by going through the list of capabilities, describing each of them and asking if the expert thinks that it fits the current SPM process.

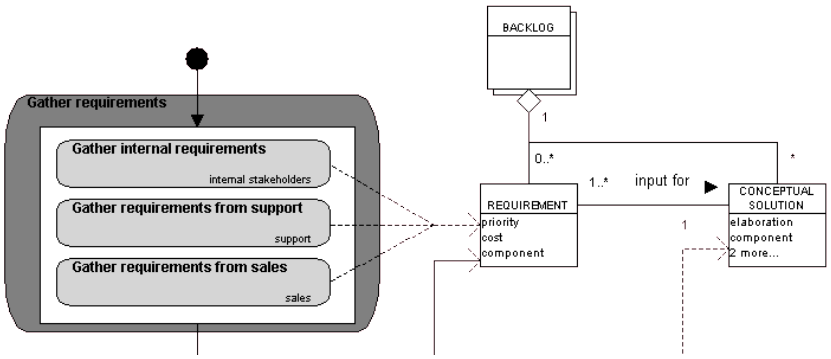


Figure 4.1: Modeling the SPM process

This approach is very to the point. It follows a simple process that makes it very efficient. However, with this approach, one can not always be sure whether the expert gives a view of the process that is too optimistic. By already providing the capabilities in advance, experts might be prone to answer 'yes' too early, when the process does in fact not yet satisfy all the requirements of a capability.

With the addition of information regarding the implemented capabilities to the PDD, we can perform assessments using a different approach. By modeling a company's SPM process first, then deciding which activities implement which capabilities and then deriving the capability matrix from it, we circumvent for a large part the possibility that experts give a too optimistic reply. The expert should only be aware of the processes that are described in the reference framework, so that he can adapt his story to them, but not of the capabilities. The reference framework should also be used by the interviewer to actively steer the interview.

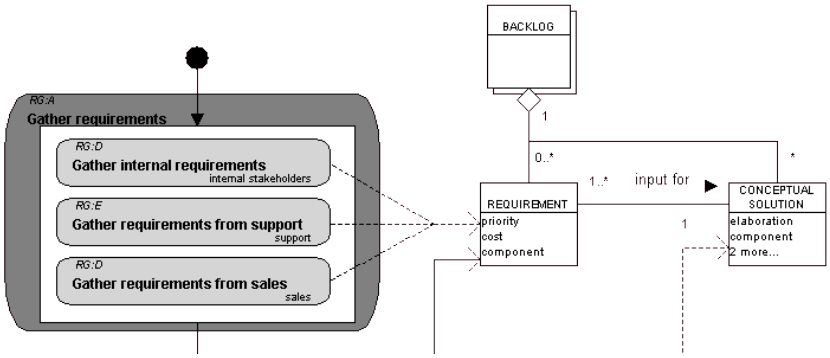


Figure 4.2: Adding capability-information to the PDD

The assessment process starts with the description of the company's SPM process. The capturing of the required data can be performed in various manners, but a semi-structured interview has worked well during the elaboration of this thesis. Other possibilities include document-analysis if process-descriptions are available at the company or a structured interview. However, with the last option it might be difficult to capture workflows, exceptions and ad-hoc processes. The first stage of the assessment process should result in one or more PDD's (see figure 4.1 for an example), describing all the relevant aspects of the SPM process. No capability-information is yet modeled.

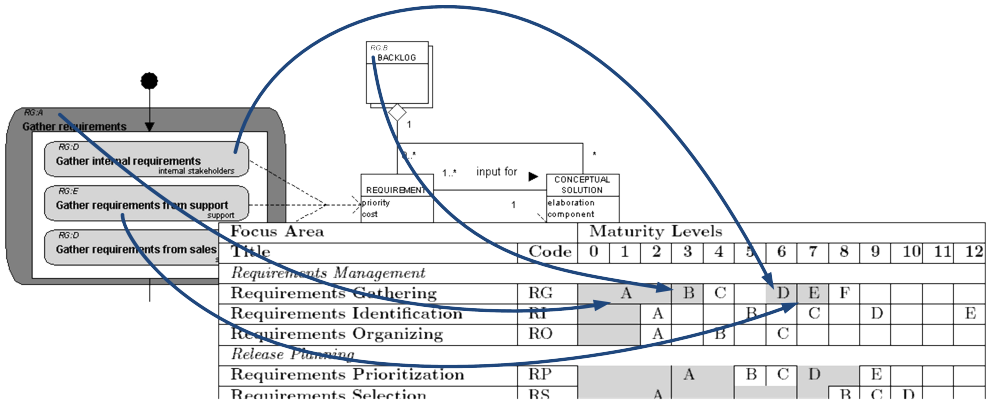


Figure 4.3: Translating the PDD into a capability matrix

During the second step, the interviewer or analyst compares the process description with the list of capabilities. Based on the descriptions of the capabilities, he/she determines which activities implement which capabilities. These capabilities are then modeled into the PDD's as described above. Figure 4.2 shows the addition of capabilities to the example above.

Once all relevant capabilities have been modeled, the company's maturity matrix can be derived from the PDD(s). This is done by marking all capabilities that are implemented by one or more activities or deliverables in the PDD(s). See figure 4.3 for an example of this process, based on the example above.

4.3 Evaluation

The approach for assessment of SPM processes described above is very similar to the original approach, proposed by Bekkers et al. (2010). Both approaches rely heavily on the reference framework for SPM and the capability matrix that was derived from it. Also, the resulting areas-of-improvement matrix is the same.

However, there are a few significant differences between the two approaches. The approach taken by Bekkers et al. (2010) can be defined as a one-tier approach. By going through the list of capabilities, product managers identify all the activities that have been implemented in their company's SPM process.

The approach described above is a two-tier approach. Firstly, the company's process is modelled without any reference to the capabilities. An expert then analyzes the process description, assigning capabilities to certain method fragments. The company's current capability profile can then be **derived** from the models.

Both approaches have their own advantages and disadvantages. In the original approach, there is no need for modelling the entire SPM process. The required information can be obtained directly by asking the right questions or by filling in a questionnaire. As descriptions are available for each capability, the expert (either the interviewer or the product manager) can easily decide whether capabilities have been implemented or not.

On the other hand, there is a risk of obtaining false data when using this approach. We have seen in the past that companies have a tendency to be rather liberal when assessing their own process or product (take as an example the requirements management tools survey by INCOSE[1]). This results in unreliable data.

When using the approach proposed in this thesis, product managers are initially not made familiar with the list of capabilities. The process descriptions that are obtained are based entirely on the description given by experts, and documents. This allows an unbiased expert to analyze the process and identify the capabilities that have been implemented, resulting in a more objective overview.

However, although modeling and assessment can be performed at the same time, this approach does require some extra work, as the entire process needs to be modeled. Also, when modeling a company's process, it is often hard to get a correct picture of the entire process. Many activities are often performed concurrently, making it hard to identify the building blocks of the process.

Ultimately, the two approaches are only a variety of the same idea. Due to their own advantages and disadvantages, they are suitable for different situations. The original approach is mainly suitable for situations in which companies want a quick solution for improving their current process. The approach described in this thesis is more suitable for situations in which complex method engineering needs to be applied, as a complete process description is needed then. In those cases, combining the modeling activity and the assessment activity can save time.

[1] http://www.incose.org/ProductsPubs/Products/rmsurvey.aspx

Chapter 5

Case Studies

This chapter describes the case studies that were performed during this project. These case-studies were performed in order to answer the second and third research question that were posed in chapter 2.1:

- How is SPM currently performed in industry?
- What are the major problems within software product management processes?

Section 5.1 describes the case study protocol that was employed. The results from the case studies are summarized in section 5.2. The individual cases are not included in this thesis.

5.1 Case Study Approach

The approach followed during the case studies is heavily based on the work by Yin (2003) and Jansen & Brinkkemper (2008). Since the state of art regarding method engineering in the domain of software product management is still not very advanced, case studies form a good way of exploring the current way of working, and the issues pertaining to it. We assume that we can identify and investigate pre-existing regularities (and irregularities) using constructs provided among others by Xu & Brinkkemper (2005) and Weerd et al. (2006a). By not restricting us to a single case, we hope to find more interesting results. Most of all, it allows us to find similarities and differences between the cases.

The approach followed for this multiple-case study follows the method described by Jansen & Brinkkemper (2008) to a large extent. According to them, a valid and reliable multiple-case study consists of four phases: case study invitation, case study initialization, case study research execution and finalization. Each phase consists of several activities.

During the case study invitation phase, a case study proposal was written and sent to several companies within the Dutch network for product software companies (ICT-Office[1]). After a positive response from a company, the initiation phase would commence. The first activity during this phase was the determination of a Project Champion. In most cases this was one of the product managers, as this was explicitly asked for in the proposal. It was made clear to the Project Champions what would be the scope of the case study, and what kinds of people and resources would be required. In most cases, the main interview would be performed with the Project Champion.

The execution phase consisted of two interviews and a document study. Due to time constraints, the scope of the latter was restricted to important templates and deliverables within the product management process, thereby excluding a detailed scrutiny of lower-level documents. The results from the interviews and document study were translated into case study reports, which will be described below. The report was then reviewed by the interviewee, to make sure that all the data were correctly interpreted. Each case study was ended by finalizing the case study report, adding it to the case study database, and sending it to the company.

[1] http://www.ictoffice.nl/

5.1.1 Why case studies?

Case studies were chosen because the research deals with questions regarding a contemporary event, over which the researcher has little or no control and in which the borders between the phenomenon of interest and its context are not clear. Also no strong theoretical base exists for the phenomenon of interest (Yin, 2003; Eisenhardt, 1989; Benbasat et al., 1987).

5.1.2 Data Gathering

The selection of cases has been based on availability. All case companies (except the Swiss company) participate in a joint innovation project, performed at Utrecht University under the supervision of dr. Inge van de Weerd and prof. dr. Sjaak Brinkkemper. Data collection was done per case by interviewing and studying documentation.

Introduction The interview session began with a short introduction of the interviewer(s). Some details were shared regarding background and current position. The interviewee was then given the opportunity to introduce him/herself.

Case Study Objective After the introduction, the case study objective was outlined. The interviewers explained the context and the goals of the interview.

Context This was followed by an introduction of the related research framework. The four layers of the reference framework (figure 3.1) were briefly discussed. Also, the processes and the stakeholders were briefly introduced.

Context The next step was the explanation of the capability attributes. Some details were given on the meaning of the attributes: name, weight, goal, activity, dependencies and references.

Context After this, the capability matrix was explained. The interviewer(s) explained what the focus areas/processes are, and what the relation is between the levels and the capabilities.

Data Gathering Once all these items were explained, the second part of the interview could start. Together with the interviewee, the interviewer(s) selected one or more of the layers from the reference framework for further discussion. The interviewer(s) then performed the following process for each of these layers:

1. For each of the activities in the layer, the following tasks were performed:
   (a) For each of the capabilities, the interviewee checked if they are correct and if they are in the right order. [RQX]
   (b) Check whether the interviewee misses any capabilities (and if so, where they should be positioned). [RQX]
   (c) Ask the following questions (when applicable; elaborate when needed): [RQ2]
      - Is the activity performed?
      - If so, is the activity supported by tools/templates?
      - If so, is the activity documented?
      - If so, are you satisfied with the current solution?
2. Ask if there are any activities missing in the capability matrix, that are performed in the company. For each of these activities, perform step 6.iii. [RQ2]

Listing 5.1: Interview structure during the first round

Per case, one or two product management experts were interviewed. These experts were required to have extensive knowledge of the product management process within their company. The interviewees were asked about the activities and deliverables within the product management

process. Furthermore, they had to be able to express their ideas on positive and negative aspects of the current product management process.

Introduction If the interviewee was a different person than during the first interview, the interview session began with a short introduction of the interviewer. Some details were shared regarding background and current position. The interviewee was then given the opportunity to introduce him/herself.

Case Study Objective After the introduction, the case study objective was repeated. The interviewer briefly explained the context and the goals of the interview.

Context This was followed by a brief recapitulation of the related research framework. Those aspects that were not clear to the interviewee anymore were repeated. This also applies to the capabilty matrix.

Data Gathering: Process&Deliverables Once all the items were explained, the second part of the interview could start. Guided by the capability matrix/reference framework, the interviewee and interviewer started drawing the activities and deliverables in each layer. Activities and deliverables were appended to the diagram until the interviewee deemed it correct. When the interviewee elaborated beyond the current topic, notes were made on another sheet. These were then used during the elaboration of that topic. [RQ2]

Data Gathering: Process If, during the drawing process, it became clear that an anomaly existed in the company's process (such as the absence of a formalized process or a small amount of attention paid to it), the interviewee was asked to give his/her view on the subject. The goal was to obtain information regarding difficulties as experienced by the interviewee. [RQ3]

Data Gathering: Deliverables After all drawings were completed, the interviewee was asked specifically about the deliverables that are used within the product management process. Notes were made of these deliverables, and the interviewee was asked for examples of these templates. These will be used during the creation of advice reports, and during the elaboration of [RQ5].

Listing 5.2: Interview structure during the second round

Each interview took approximately two hours. During the first interview, the structure that is denoted in table 5.1 was followed. For each step, an indication is provided stating which research question it is related to. The two sub-questions for this research will be denoted as [RQ2] and [RQ3]. The first interview also handled an additional research question, aimed at validating the capability matrix. As this question is not part of this thesis, this question will be denoted as [RQX].

Due to the relatively unstructured approach of the first round, these interviews did not generate sufficient information for the rest of this study. Therefore, a second round was held. During these interviews, a more structured approach was followed. Guided by the capabilities in the capability matrix, the interviewer and interviewee drew the product management process in a combined effort. The result was a set of draft PDD's, appended with notes, that allowed a more detailed description of the company's process. The structure followed during the second round is described in table 5.2. The indications of the related research questions remain the same as above.

The documents that were studied varied per case, but included in most cases requirements templates, release definitions, roadmaps and tool screenshots.

All data collected was included in a case study database. The case study database consists of case study notes, interview recordings, case study reports/advice reports, filled-in capability matrices and situational factors.

Whom to interview and what documents to study was decided upon in mutual agreement between the researcher and the contact person of the organization being studied.

5.1.3 Deliverables

All data related to the case studies is stored in the case study database. This database is the repository of all information input to and produced by the case study. It contains the following categories: digital organization material (digital folder 1), digital research material (digital folder 2), paper organization material (binder 1) and paper research material (binder 2). The deliverables of each case study are the following:

Four process-deliverable diagrams. For each of the focus areas in the reference framework for SPM, a PDD was drawn based on the findings at the case company.

Capability matrix. Based on the process-deliverable diagrams, a capability matrix was generated for each case company. This capability matrix formed the basis for the advice report.

Advice report. For each case explored, an advice report has been written that describes the methods used, the major bottlenecks that were found, and a set of recommendations.

Results will only be published after explicit approval by the organization.

The case study report is expected to be of value to the organization as it provides insight in what works and what does not work within the context of Software Product Management.

5.2 Results

5.2.1 Case Study Companies

A total of six companies participated in this study. All of these are located in the Netherlands. Due to confidentiality issues, the names of the participating companies are not shown in this document.

5.2.2 Current Situation

Based on the results of the case studies, we are able to create an overview of the current state of SPM in small to medium Dutch software companies. In order to do so, we combine the six maturity matrices of the Dutch cases into one overview matrix.

Only the Dutch cases have been taken into account for this overview, for two reasons. The first reason is that combining Dutch cases with a Swiss case would would skew the overview, as the Dutch and Swiss markets might not be the same. Secondly, the Swiss case was not performed entirely. Only a subset of the focus areas have been elaborated. Taking this partial data into consideration would thus also give a distorted image.

Table 5.3 shows the overview of the current state of SPM. The gradient gets darker as more companies have implemented each capability. A white shade means that none of the companies has reached that maturity level. Dark gray means that all six companies have reached that maturity level.

Several observations can be made regarding the overview. First of all, overall maturity is fairly low. While most companies have implemented the activities related to the lowest maturity levels, this gets drastically lower when we move on towards the higher levels. After the basic implementation, most companies seize to improve the process by putting more advanced processes in place.

There is however a minor difference between the average maturity level regarding release planning, compared to the other focus areas. Especially in the case of requirements prioritization, release definition and launch preparation, higher maturity levels are obtained. In the other areas, requirements gathering & identification and roadmap construction also reach considerably higher scores. This is not a surprise, as these are the most essential activities in the field of product management.

Automation is still a fairly neglected point in most companies. Activities related to this, such as automation of requirements gathering (RG:C), automatic connection of similar requirements (RI:E), automation of release definition (RD:E) and automated notification of scope changes (SCM:B) all have a very low score. Automation is in most cases a step that is only taken once the process is sufficiently mature. However, in the case of product management, big advantages could

Focus Area		Maturity Levels												
Title	**Code**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Requirements Management														
Requirements Gathering	RG		A		B	C		D	E	F				
Requirements Identification	RG			A			B		C		D			E
Requirements Organizing	RG			A		B		C						
Release Planning														
Requirements Prioritization	RG				A		B	C	D		E			
Requirements Selection	RG			A						B	C	D		
Release Definition	RG				A	B	C		D		E			
Release Validation	RG				A			B			C		D	
Launch Preparation	RG		A				B		C			D	E	F
Scope Change Management	RG				A		B				C		D	
Product Roadmapping														
Theme Identification	RG					A		B						
Core Asset Identification	RG						A			B				C
Roadmap Construction	RG			A			B	C		D	E	F		
Portfolio Management														
Market Trend Identification	RG			A		B		C		D				
Product Lifecycle Management	RG			A			B			C				
Product Line Identification	RG							A			B			

Table 5.3: Current SPM maturity of product software companies

be obtained by early implementation of advanced information systems for automating tasks such as requirements gathering/organizing, requirements prioritization and release definition. Especially when the rate of incoming requirements is high.

Interesting is the fact that none of the case companies have any formal or academic methods and techniques in place. Especially in the case of requirements identification, requirements prioritization and roadmapping, ample technique is described in literature. Nonetheless, the barrier for using such techniques is often too high. During the interviews, the major reasons that were given for this phenomenon were the lack of access to such literature and the complexity of many methods.

5.2.3 Common Problems

Based on the results of the case studies, an advice report has been written for each of the companies. In this report, the major bottlenecks were identified. The bottlenecks were mainly discovered by searching for gaps in the capability matrix. Based on the bottlenecks, advice was given on how to improve the product management process. Again, this advice was based on the steps that are proposed in the capability matrix.

Both the bottlenecks as well as the advice differed in size. Some related only to a small portion of the process, while others implied the addition of an entirely new process. The bottlenecks that were identified included examples such as:

- an informal approach to portfolio management and/or roadmapping;
- a lack of internal communication;
- decentralized requirements storage; and
- the lack of a prioritization technique.

Most of the problems, or variations of them, were shared among several companies. The problems or bottlenecks that were discovered by looking at the maturity matrix can be generalized into five types of problems that were common for all cases.

The first problem can be described as **informality**, as many of the processes were performed ad-hoc, without any structured approach. In some cases, it was hard to decide whether a certain

capability was implemented or not. However, the rules that describe the capability matrix state that processes must be formalized internally. As most processes were not documented/described, this can be seen as a very common problem.

The second problem is labeled **process absence**. This indicates the lack of an entire process such as portfolio management or product roadmapping. It often means that a company has not yet found it important to pay attention to this aspect.

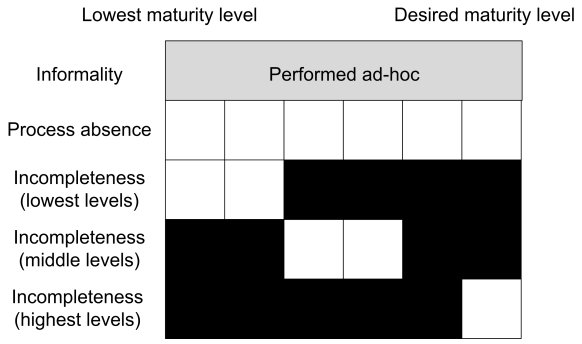


Figure 5.1: Generalized common problems in the SPM process

The third problem that was identified is **incompleteness**, which can be witnessed in three variations. Essentially, this problem describes processes in which a company does perform some of the capabilities, but activities are missing. These missing activities can be found at three positions in the maturity matrix.

Firstly, a company can perform activities related to some capabilities without performing those at the lowest level. In some cases, this might not be an issue. However, in most cases, such a situation does not make sense.

The gap can also exist in the middle of a process, i.e. activities related to the lowest and highest capabilities are performed, but some steps are skipped. Especially in the case of processes such as requirements prioritization or release definition, it might be better to complete the processes by also performing the missing activities. For example, involving the partners while not involving the internal stakeholders will probably lead to discontent among the employees.

Finally, gaps can be found at the higher capability levels. This essentially means that a company does not perform all activities up to the desired maturity level.

5.3 Example Case Study Report

The company that is used in this example (from here on: CaseComp) has a long history with a lot of experience in their market sector. Its network is therefore extensive. Also, a large customer base within a full market results from it. However, the industry is showing change. One of the effects that can be seen is a decrease in the amount of customer companies, but an increase in their size. This happens as more and more companies are combined in larger organizations.

This is a change that has to be reflected in the actions of CaseComp. Currently, this is not yet entirely the case. The long lifetime of the company has resulted in a large portfolio of products, developed in an old-fashioned way, governed by habits and outdated processes. This situation is enforced by a large, senior workforce.

Furthermore, rigorous change is made difficult by the big influence of legislation and industry standards, and a strong company policy. On the other hand, the market does not require huge innovations, shown by the small amount of incoming requests each year. Many customers' main requirement is that the software works as simple as possible, allowing them to facilitate the huge administrative work that they need to deal with.

Table 5.4 shows a list of situational factors for CaseComp. In 2008, Bekkers et al. (2008) determined this list that can be used to describe a company or business unit. These factors are the main factors influencing the selection of software product management method fragments.

Development Philosophy	Waterfall	Size of business unit team (FTE)	+-100
Customer loyalty	High	Customer satisfaction (1-10)	N/A
Customer variability (%)	3%	Number of customers	15.000
Type of customers	Small, medium and large companies		
Number of localizations	1	Market growth	Decreasing, but the size of companies is increasing
Market size	15.000 Customers	Release frequency	Every 35 days
Sector	[Hidden]	Standard dominance	High
Variability of feature requests	Low		
Defects per year	Many	Development platform maturity	Ever changing
New requirements per year	20-80	Number of products	
Product age	Between 1 and 12 years	Product lifetime	Max. 4 years
Product size (KLOC)		Product tolerance	Medium
Company policy	High	Customer involvement	Low
Legislation	Medium	Partner involvement	N/A

Table 5.4: Situational Factors CaseComp

5.3.1 Capability Matrix

For measuring the maturity of CaseComp regarding software product management, we employ a capability matrix developed for this purpose by Bekkers et al. (2010). The matrix shows all the capabilities as defined in the reference framework for SPM (figure 3.1) on the vertical axis. On the horizontal axis, between two and six maturity levels are depicted for each capability. There maturity levels are indicated by a letter ranging from A to F, where A is the lowest level and F the highest. The maturity levels are placed according to their relative importance. By marking the implemented capabilities and their respective maturity level, one can find a company's current state regarding spm. Table 5.5 provides an overview of the maturity of the SPM process at CaseComp.

Focus Area		**Maturity Levels**												
Title	**Code**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Requirements Management														
Requirements Gathering	RG		A		B	C		D	E	F				
Requirements Identification	RI			A			B		C		D			E
Requirements Organizing	RO			A		B		C						
Release Planning														
Requirements Prioritization	RP				A		B	C	D		E			
Requirements Selection	RS			A						B	C	D		
Release Definition	RD				A	B	C		D		E			
Release Validation	RV				A			B			C		D	
Launch Preparation	LP		A				B		C			D	E	F
Scope Change Management	SCM				A		B				C		D	
Product Roadmapping														
Theme Identification	TI					A		B						
Core Asset Identification	CAI						A			B				C
Roadmap Construction	RC			A			B	C		D	E	F		
Portfolio Management														
Market Trend Identification	MTI			A		B		C		D				
Product Lifecycle Management	PLM			A			B			C				
Product Line Identification	PLI							A			B			

Table 5.5: Software Product Management maturity at CaseComp

5.3.2 Identified Issues

Due to its relatively long lifetime, large amount of partners, and wide variety of products, CaseComp has a decent amount of experience with the problems of portfolio management and product roadmapping. However, this experience does not show in the capability matrix shown in the previous section. This lack of maturity results from the relative ad-hoc approach that CaseComp uses in tackling these problems. Figure 5.2 shows a visualization of the portfolio management at CaseComp, in the form of a Process Deliverable Diagram. The diagram shows in one combined view all the activities that CaseComp performs in relation to portfolio management. The diagram maps directly to the capability matrix.

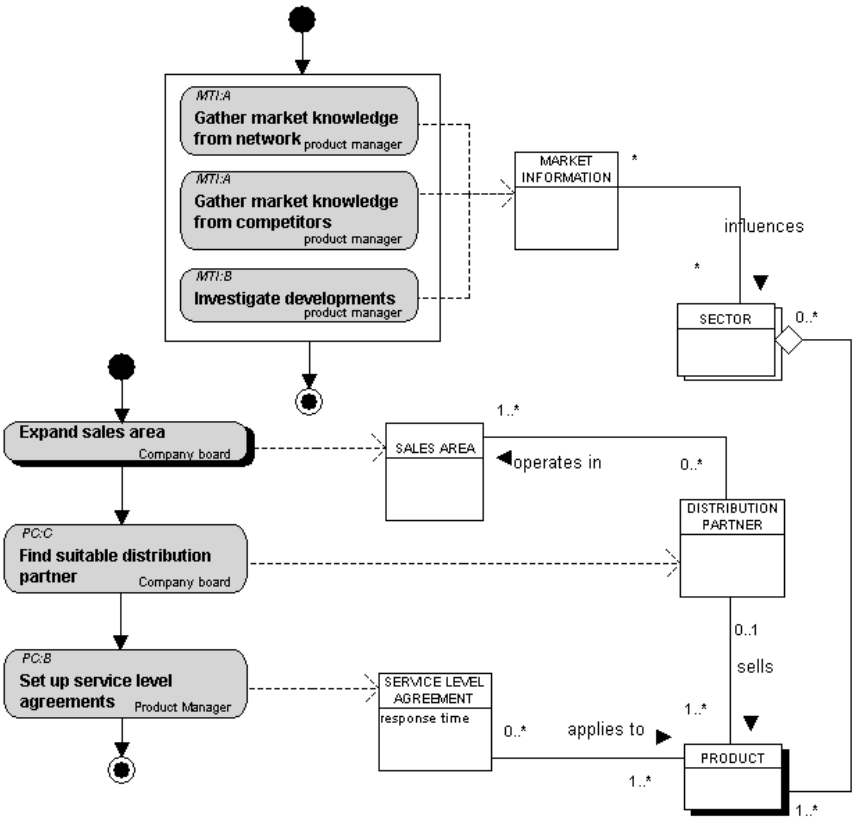


Figure 5.2: Process-Deliverable Diagram of CaseComp's process at the portfolio level

Although the product portfolio consists of a considerable amount of products, there are hardly any structural activities related to the management of it. A lot of ad-hoc discussion takes place at the top of the company, currently resulting in a restructuring of the portfolio. However, recurring decisions based on solid figures seem absent. Most of the portfolio management related activities that take place on a regular basis are related to (distribution-) partner management, which is only loosely related to product management.

A **Informal approach portfolio management**
Therefore, one of the first problems one can identify is the relative simplicity of each of the focus areas. Neither of these areas is elaborated thoroughly, as far as a formal process description is able to show. Although it can be a great advantage to keep the communication lines short and to refrain from forcing too many rules and protocols on employees, a lack of process descriptions brings considerable risk. Especially on a high level such as that of the product portfolio, loosing knowledge when a person is removed from the process can cause problems.

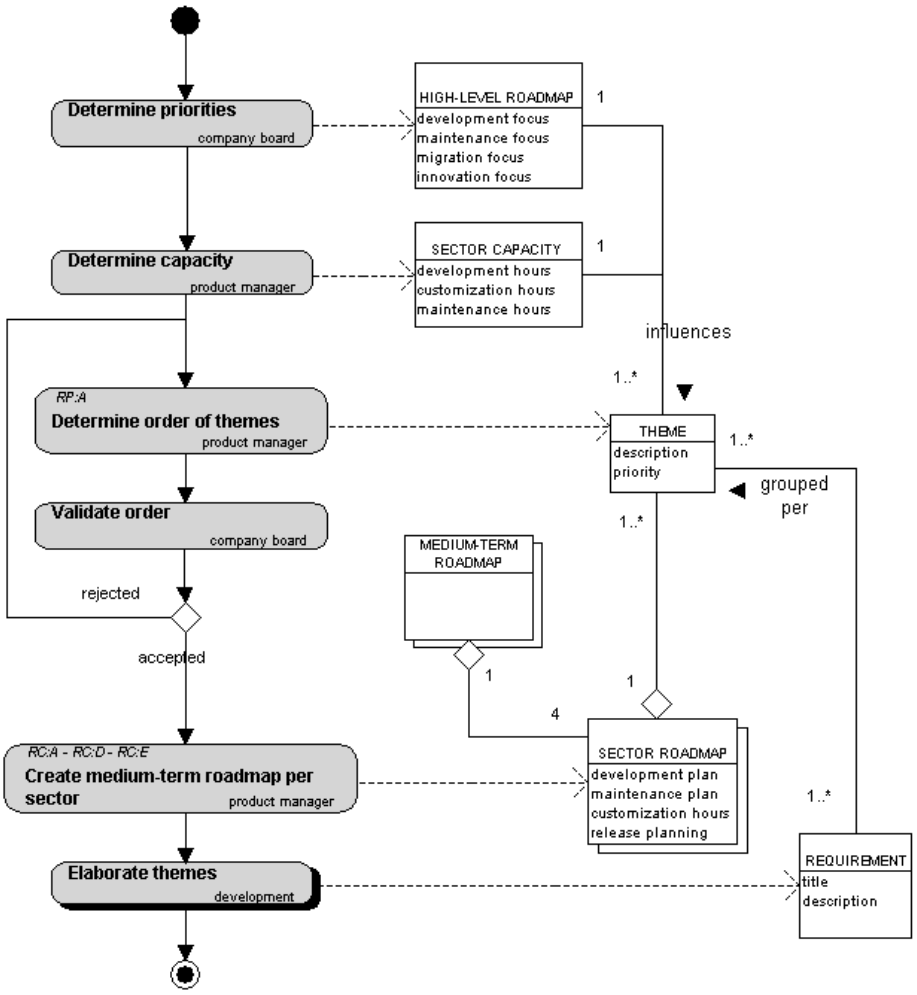


Figure 5.3: Process-Deliverable Diagram of CaseComp's process at the roadmap level

The roadmapping process at CaseComp (shown in figure 5.3) seems fairly advanced, including theme identification and the creation of a long-term roadmap for all of the active sectors. However, some of the same issues as before happen here as well. In this case, it can be seen that the task of *roadmap construction* is performed at a decent maturity level. However, two activities related to the lower maturity levels, *internal consultation* and *customer variant roadmapping*, are not performed.

B **Lack of internal communication about the roadmap**
Although *internal consultation* and *customer variant roadmapping* are not required for the following activities, it displays a gap in the SPM process. In this particular case, these activities are in fact an important step in the creation of a complete, detailed roadmap, instead of being an irrelevant addition. Internal participation is an important means to raise both the quality as well as the acceptance of the roadmap.

Furthermore, *theme identification* is depicted in the PDD, but the CM shows a maturity level of 0. We are dealing here with a gap in the documentation of the process. To prevent knowledge loss, it would be better to define protocols for these activities.

C **Missing core assets**
The third, and probably major, problem is the lack of core-asset identification, which could

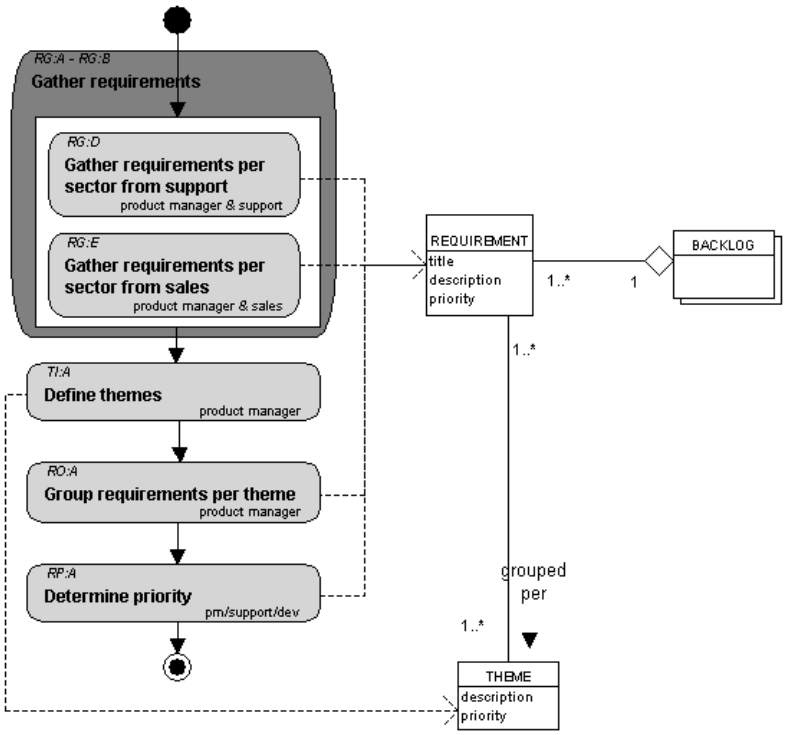


Figure 5.4: Process-Deliverable Diagram of CaseComp's process at the requirements level

prove very useful given the amount of applications in the portfolio. This problem is indicated by a low alignment within this focus area. The use of core assets allows great savings when parts of the applications are shared, such as user management, security and database-drivers. Knowledge about the existence of such components is therefore very important.

Compared to the roadmapping process, the process related to managing requirements seems fairly bleak. First of all, we see a big gap in the maturity for requirements gathering. Once gathered, requirements are hardly analyzed or elaborated, based on the official processes. Only some attention is being paid to organizing the items based on themes, but at a very low, informal level. The same goes for prioritization, where no pre-defined technique is applied for estimating importance.

D **Decentralized requirements storage**
Gathered requirements are stored per project in a document located in a Sharepoint-folder. Although Sharepoint is a mature environment which is very usable for the sharing of documents throughout the company, it is not the ideal solution for storing requirements. The platform is not specialized in storing such data, making it hard to apply structure. Through centralized storage in an environment that has been designed for doing so, this process can be improved significantly.

E **Unstructured requirements analysis**
The structure of the documents on the Sharepoint-server is not pre-defined, and can thus vary per project. This approach is not very structure, which could lead to disruptions in the process.

The final area discussed here is release planning. At CaseComp, all activities on the release-level concern only the post-release period. As can be seen in the matrix, maturity there is very low, with the exception of two activities fulfilling high levels. The fact that some advanced features re in place while the more basic ones are not might suggest a problem.

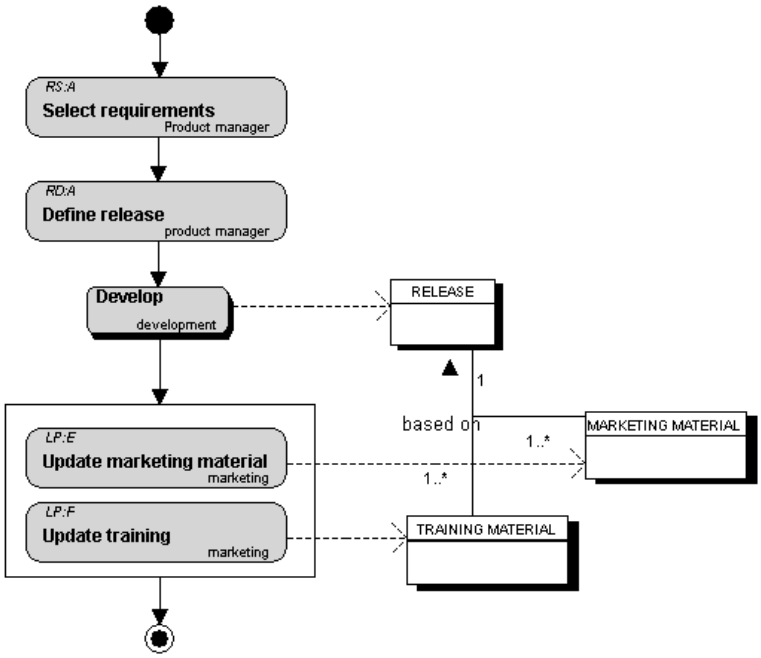


Figure 5.5: Process-Deliverable Diagram of CaseComp's process at the release level

Regarding release planning, a lot can be gained by predefining the contents of the next release(s), based on objective numbers. This also improves clarity both for internal as well as for external stakeholders. Instead of developing according to the prioritized (full) backlog, more detailed plans could be made.

5.3.3 Proposed Improvements

Not all of the problems described in the previous section are of equal importance. All of them were derived in the same way, by analyzing the capability matrix and the process-deliverable diagrams, and they were described without taking into account the impact of the issue. In this section, solutions will be proposed for those problems that both seem important as well as have the potential to be improved with relatively little effort.

A **Formalization portfolio management**
The problem of incomplete of non-existing process description is generally easy to solve. The portfolio management process at CaseComp is not yet very complex. This means that this is the perfect moment to start the formalization of this process. The impact is low, while the advantages for the future are significant. The process description that is given in figure 5.2 can be used as a basis for further elaboration of this process.

B **Improved roadmap communication**
Secondly, the gap in the roadmapping process should be filled in order to lift the roadmap construction process to a higher level. This means that two improvements need to be made. The capability matrix suggests that the first improvement is to more actively involve internal stakeholders during the creation of the roadmap. As this is a generally good thing to do, I suggest to introduce a presentation of the roadmap every 6 months. This presentation should be open to all internal stakeholders, such as support, development and sales. Feedback obtained during this presentation should be actively incorporated in the roadmap. Such an approach increases both the creative input as well as the internal acceptance of the company's direction.

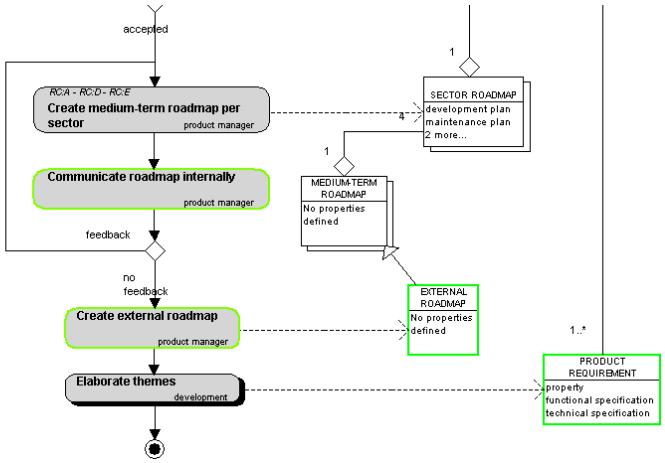


Figure 5.6: Improvements for CaseComp's Roadmapping process

In addition to this, it might be a good idea to open up communication channels towards the customers. As the majority of the customer-base is not technologically savvy, some effort is required to obtain their input. This is already done through visits of CaseComp consultants, and this could be appended by the publication of a brief overview of CaseComp's plans within the near view. This means the creation of a trimmed down customer variant of the roadmap, in which only the most important changes are shown.

Both improvements to the roadmapping process are shown in figure 5.6.

C **Introduction core-asset identification**
A lot can be improved by gaining a better insight into the components that are used by multiple products in your portfolio. Through the implementation of a better requirements management solution and increased traceability, it is easy to gain a better view of the requirements that apply to multiple products.

It is important that a clear overview is created and maintained of all the components that are or can be used by multiple products. This overview needs to be available in a central location, preferable in the same system that is used for requirements management. For every component, a clear description needs to be given. Also, every component needs to be linked to all related requirements. By making the list clear and up-to-date, both product managers as well as developers can safe time by preventing doing the same work twice.

D **Introduction requirements management tool**
Companies that do not store all requirements in a central location run great risks. These risks relate both to the product management as well as to the development process. Examples of negative effects are low traceability, less transparency and an ineffective development process. As requirements management forms the core of the product management and development process, it is important to handle the requirements in the right way. In this light, a lot can be gained by the introduction of a requirements management tool at CaseComp.

	Partnering & contracting	Market trend identification	Product lifecycle management	Product line identification	Theme identification	Core asset identification	Roadmap construction	Requirements gathering	Requirements identification	Requirements organizing	Requirements prioritization	Requirements selection	Release definition	Release validation	Scope change management	Launch preparation	Platform
Accept Ideas (Accept Software)								X	X	X	X	X					Web
Accept Portfolio (Accept Software)		X	X	X													Web
Accept Requirements (Accept Software)						X	X	X		X	X						Web
Primavera Portfolio Management (Oracle)				X													Desktop
Agile Product Lifecycle Management (Oracle)			X														Desktop
DOORS (IBM Rational)								X	X	X					X		Desktop/Web
Focalpoint (IBM Rational)						X	X	X			X	X	X				Web
RequisitePro (IBM Rational)									X	X	X						Web
Featureplan (Ryma Technology Solutions)		X	X	X	X	X	X					X	X				Web
IdeaScope (Ryma Technology Solutions)								X	X	X	X	X		X			Web
Jira (Atlassian)								X	X	X	X						Web
ChangePoint IT Portfolio Management (Compuware)			X	X		X											Desktop
CaliberRM (IBM)								X	X	X	X	X					Desktop
Optimal Trace (Micro Focus)								X	X	X							Desktop
OSRMT (Open Source)								X	X	X	X						Desktop
Portfolio Manager Software Suite (UMT)			X			X											Desktop
RaQuest (Sparx Systems)									X	X							Desktop
VeryBestChoice Light (Expert Decisions)								X		X	X						Web
ReleasePlanner (Expert Decisions)						X	X		X	X			X	X			Web
ReqSimile (n/a)									X								Desktop
Rational RequisitePro (IBM)									X	X	X						Web
Accompa (Accompa)								X	X	X							Web

Table 5.6: Product Management Tools

Currently, many tools are available. Table 5.6 shows the most important and well-known of these tools, along with the SPM-areas that they support. The focus areas of these product vary from idea-development and bug-tracking to portfolio management and product roadmapping. For CaseComp, the biggest advantage can be gained in requirements management. This means that the tools IBM Doors, Accept Requirements, Jira and Accompa are good candidates. Currently, many companies choose for Jira, as it is flexible at a relatively low cost (starting at 1200$ for 25 users). Accompa also offers flexible, web-base software, but the price-tag is a little bit steeper (199$ per month for five users; after that, an additional 29$ per month per user). In principle, the two packages cover the same areas. Accompa wins in the area of user-friendliness, while Jira offers more flexibility and expansion-possibilities. Jira's options for expansion include collaboration-tools, code-reviewing, and traceability at source-level.

E **Distinction product vs. market requirements**

To create more clarity in the requirements engineering process, it is good to maintain a distinction between product- and market requirements. Market requirements in this sense are the wishes and improvements as expressed by the external stakeholders such as customers. They are written down as-is, with some additional information such as issuer, initial priority, date, etc. Product requirements are the result of further analysis of market requirements by a requirements engineer or a product manager. Market requirements are turned into product requirements by combining similar ones, indicating dependencies, rewriting descriptions, assigning priorities, etc.

Whereas market requirements might not always be written down in the same form, product requirements should be, in order to improve efficiency. For each product requirement, the same pieces of information should be available. Also, a link should be maintained between market requirements and the product requirements that are based on them. This link can be used to maintain traceability, and thus to be able to, for instance, notify stakeholders of the status of a requirement.

Introducing the distinction between market- and product requirements implies a few changes to the process, mainly on the deliverable side. First of all, two separate databases should be maintained; one for the market requirements and one for the product requirements. Also, a

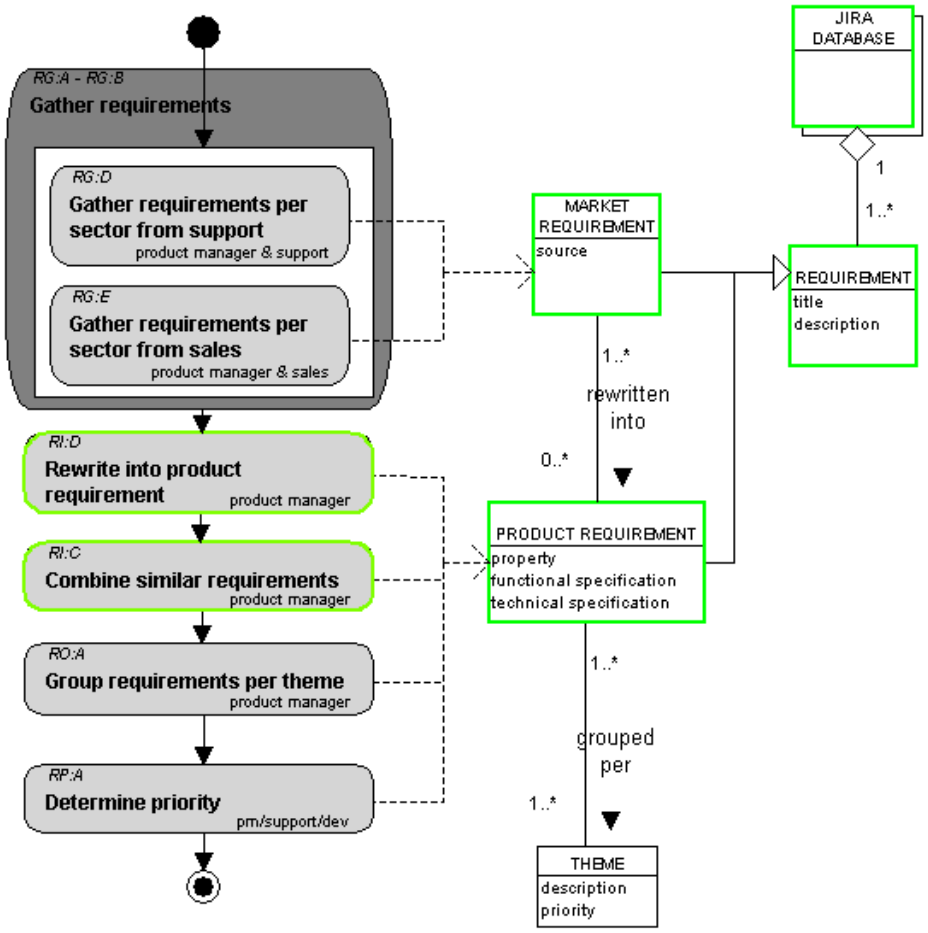


Figure 5.7: Improvements for CaseComp's Requirements Management process

link should be added between the two. Furthermore, some additions to the process-side need to be made. Incoming requirements are no longer initially stored as a 'REQUIREMENT', but as a 'MARKET REQUIREMENT'. Once stored, market requirements need to be processed and rewritten. To start with, I suggest to add two activities: 'combine similar requirements', and 'rewrite into product requirements'. The result of these two activities is a set of product requirements of similar form.

The improvements at the requirements level are shown in figure 5.7.

Chapter 6

Product Software Knowledge Infrastructure

The idea of a knowledge infrastructure for software product management originates from the fact that the function is still not clearly defined. The area is in need of structuring, so that its processes and products can be improved and optimized. A knowledge infrastructure would aid in making available the resources that are now difficult or impossible to access. These resources can be methods, techniques, tools and templates.

By using new web-technologies, a wide range of possibilities becomes available for improving the visibility and transparency of all aspects of software product management. By visualizing products, milestones, requirements and stakeholders in new ways, the efficiency and effectiveness of product managers might be significantly improved. This is especially true when we evolve the PSKI towards Method-as-a-Service. With the addition of maturity levels, templates and visualizations to the original PSKI, and the formalization of method storage, selection and combination, this is a logical next step. Through this, a large portion of the burden that is related to the use of certain Software Product Management methods can be taken away.

The Method-as-a-Service (MaaS) concept comprehends two intertwined parts. The first part consists of the method modification. The main problems in regard to this are linked to process modification, deliverable administration, method fragment storage and method fragment selection. The second part of MaaS is related to the execution of methods. A full implementation of MaaS includes the ability to generate templates, visualize data and 'perform' the method online. This includes executing all steps within the process and storing all resulting data online.

van De Weerd, Versendaal, & Brinkkemper (2006) and Brinkkemper (1996) already identified four main activities that are necessary to create a computer-aided method engineering tool. These activities are 'analysis of need', 'selection of alternatives', 'embedding of process advice', and 'method administration'. The first of these activities has already been elaborated by Bekkers et al. (2010), and will therefore only be discussed briefly, for the sake of completeness. The other activities will be discussed in more detail.

The solution proposed in this thesis is an extension and elaboration of the original proposal. This means that the original activities are given more detail, and that several aspects are added to the model. This should result in a firm basis for further research, and the actual gradual implementation of the knowledge infrastructure. The new model of the PSKI (figure 6.1) is very similar to the original model. However, it contains two extra activities, several added relations and more detail. The parts that have been added or heavily updated are depicted as red.

Originally, the first phase within the PSKI proposal was 'analysis of need & situational indicators', including an assessment of a company both in a general manner employing situational indicators, as well as focused on the SPM process by using capabilities. During the last few years, these concepts have been expanded and elaborated. Bekkers 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 process can be performed by filling in a capability matrix 3.2.7. Based on the results of these two assessments, process advice can then be given to the company.

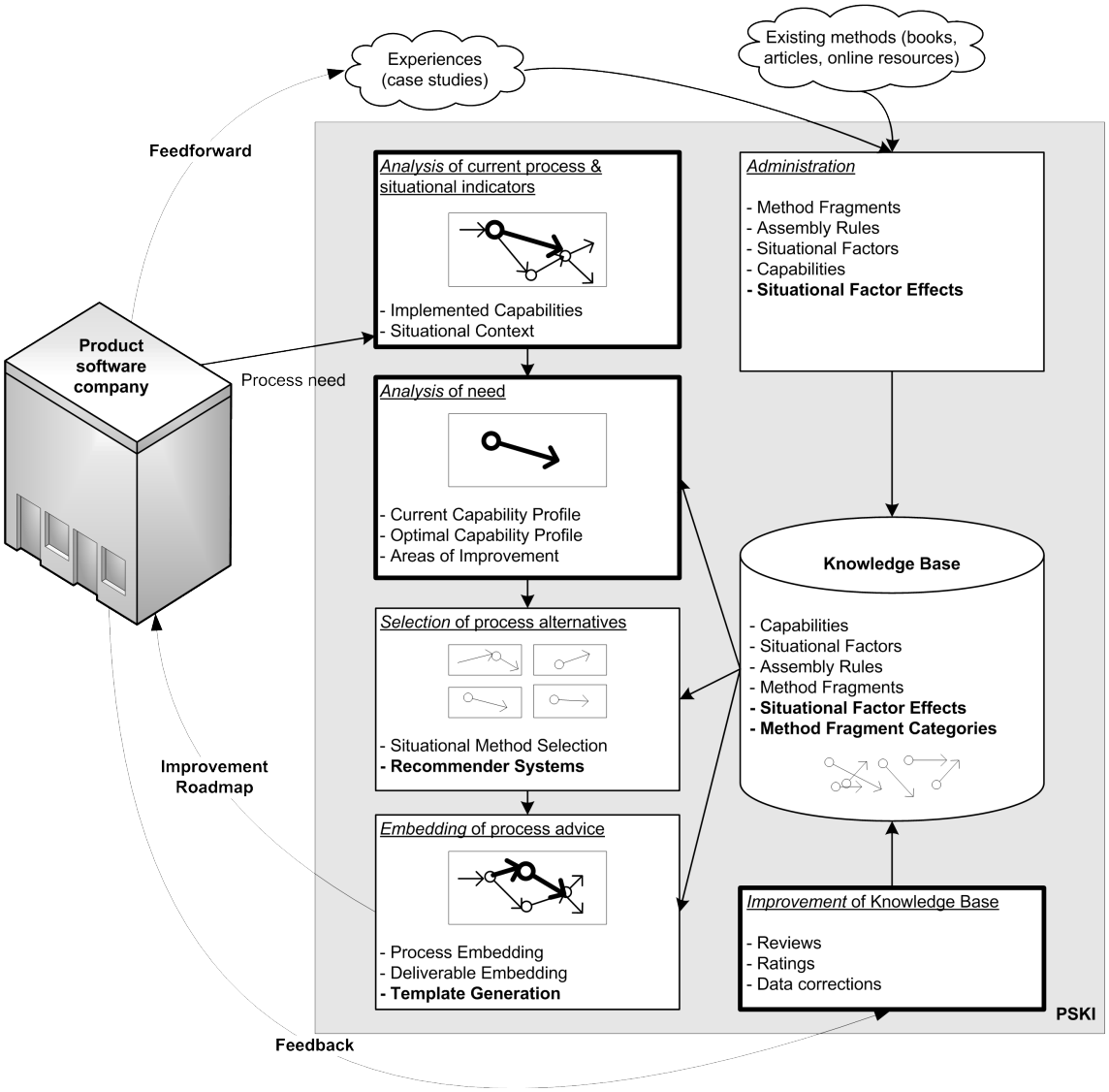


Figure 6.1: Extended version of the PSKI

Due to the advances since the initial publication of the PSKI, it no longer seems appropriate to place both analysis of the current situation as well as analysis of the need in one phase. The complexity of these activities calls for a separation into a phase focusing on the current situation, and a phase focusing on the future situation. Therefore, the name of the first phase should be changed to 'analysis of current situation'.

As several expert have commented during the case studies, the motivation to rely on an automated approach for the generation of a new product management process is low. By taking out the human factor, one also removes all trust that comes with dealing with experts. Although a tool can form a very good basis for process improvement, it should always be appended with a human factor. In the new design of the PSKI, this is done by incorporating a phase for knowledge base improvement. This step ensures valid data and relevant advices based on regular user input.

6.1 MaaS: Method Modification

6.1.1 Analysis of Current Process & Situational Indicators

The first phase is more complex than was initially proposed. As the demand on method engineering can vary per case, we need to incorporate a certain amount of flexibility into the process. First of all, we can identify two types of situations regarding the motivation for employing method engineering:

The need for improvement of a specific area In many cases, method improvements can better be performed in an evolutionary way rather than in an revolutionary way. By doing so, you reduce risk and increase the chance of success. This also means that it is often not required to analyze the entire product management process. Instead, only a subset of the areas is looked at, and only for those areas improvements are provided.

The need for improvement of the entire SPM process For companies that do require a major improvement of their product management process, this should be a possibility in the PSKI. In those cases, the set of areas that is analyzed should be extended to incorporate the entire spectrum of product management related activities, as described in the reference framework for SPM 3.1. This group also contains (new) companies that wish to obtain advice for the product management process 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 during the creation of the PSKI.

Secondly, data from the interviews suggests 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. We can distinguish two manners in which companies are willing to provide information regarding their current product management process:

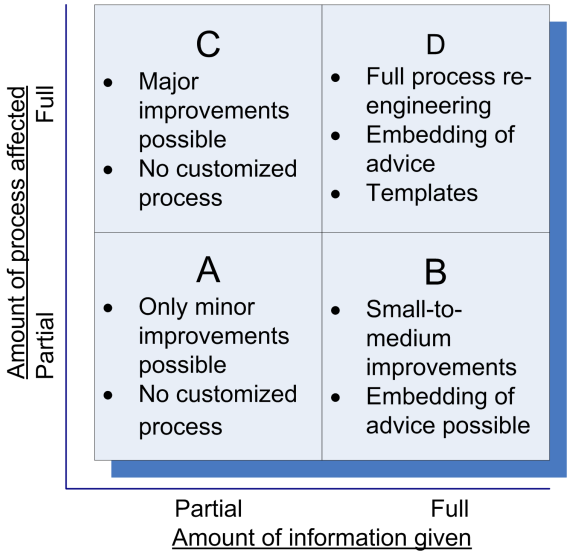


Table 6.1: Variations in the input of the PSKI

Full process information In the optimal case, companies are willing to provide complete information regarding their current process, deliverables, and situational factors. 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 of 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.

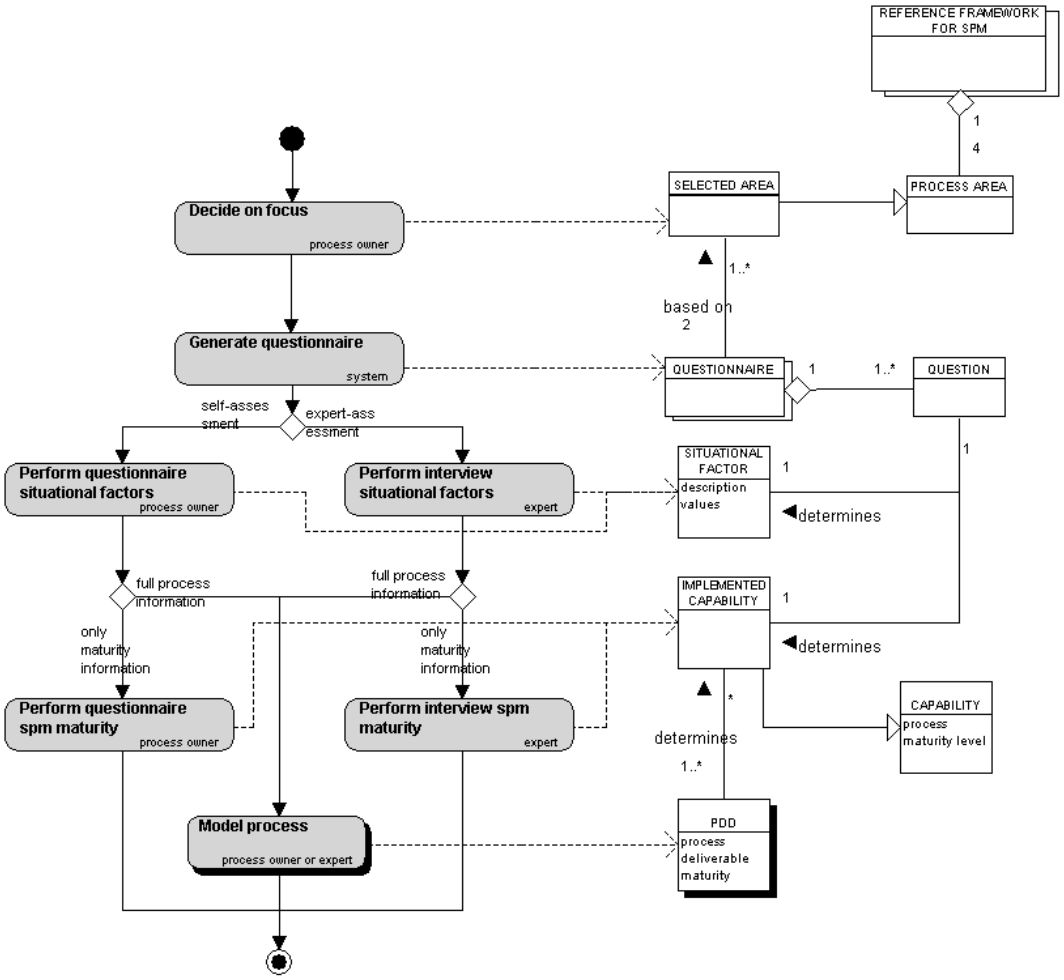


Figure 6.2: Analysis of current situation

Only situational factors and maturity information 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.

These four variations can be depicted in a two dimensional table (table 6.1), with the extend of the affected process on one axis, and the amount of information provided on the other. In any case, information regarding the situational context of the company should be obtained. A very efficient way of doing this is by conducting a questionnaire with a list of all the relevant situational factors, as described by Bekkers et al. (2010). To enhance reliability of the data, the questionnaire could be replaced by an interview.

To gather the remaining information, there are two options, depending on the quadrant of table 6.1. If and when a company is willing to provide full information regarding (part of) their product management 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 are a very suitable technique for this purpose. Section 7 will explain how PDD's can be created in an efficient way. Section 4 shows how the PDD modeling technique can also be used to capture the current maturity level of a company's product management process.

In the case that a company chooses not to provide full information regarding (part of) their product management process, a questionnaire can again be used Bekkers et al. (2010). To enhance reliability of the data, this questionnaire could also be replaced by an interview.

6.1.2 Analysis of Need

The second part of the original first phase in the PSKI is now placed in the new phase 'analysis of need'. This 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. This phase has already been described by Bekkers 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; 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.

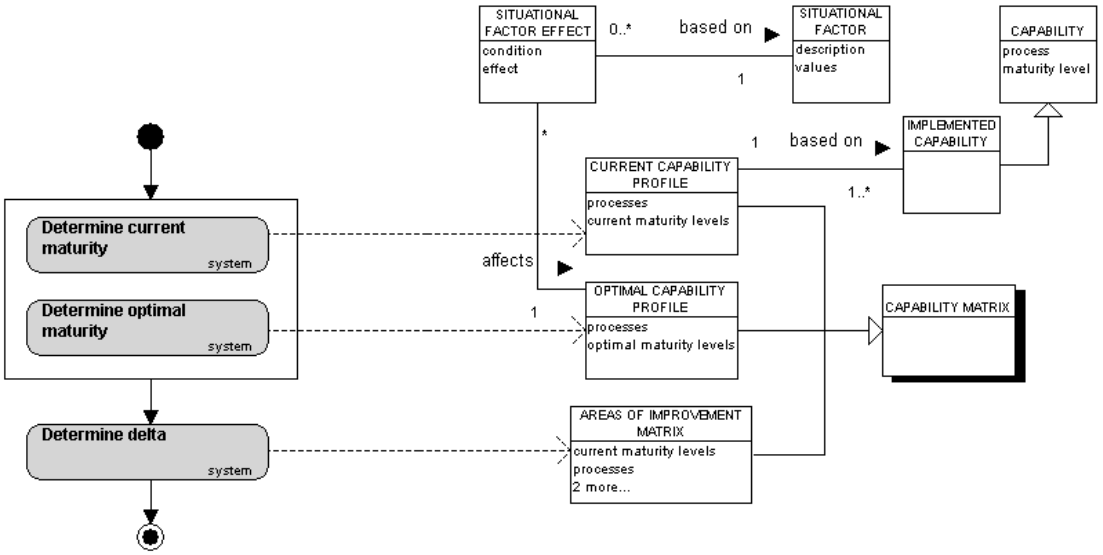


Figure 6.3: Analysis of need

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. For example, having only one product in the product portfolio could have as effect that the process 'product line identification' is not needed. 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 can be called the *delta*. This delta indicates the capabilities that need to be implemented, in order to arrive at the optimal maturity level.

What is important in the context of this phase is the fact that product managers can vary in the rigidity that they demand from the method engineering 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.

The result of this phase is thus an areas of improvement matrix. This matrix describes 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.

6.1.3 Selection of Process Alternatives

For the next phase, each missing capability has to be connected to a method fragment that implements the capability. To facilitate this process, it is best if each method fragment implements only one or a few capabilities. Although this poses a restriction during the creation of method fragments, this is justifiable since it simplifies the remainder of the process drastically. However, method fragments that implement more capabilities, such as complete prioritization techniques, are acceptable.

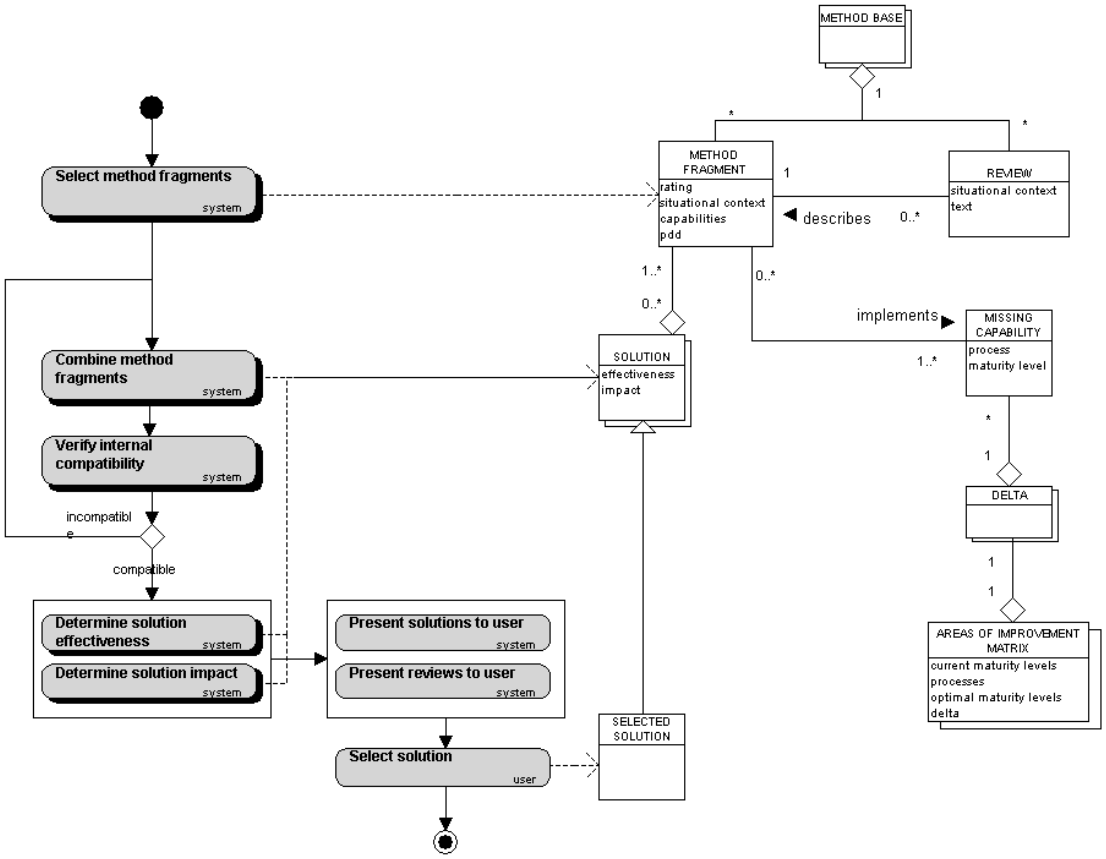


Figure 6.4: Selection of process alternative(s)

The capabilities that a method fragment implements can be used as an attribute during the initial selection of method fragment candidates. As will be described later on, both process fragments as well as deliverable fragments can implement capabilities. For this reason, **capability** is an effective first classifier of a method fragment in the method base.

For further classification, we can reuse the situational factors described by Bekkers et al. (2008). Since many fragments will be applicable in any situation, it does not make sense to describe each fragment by all factors. This would also pose a problem when the list was changed. Therefore, 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 et al. (2010). However, as the rules apply in this case only to one specific fragment, the effect does not have to be specified explicitly, as this will always be 'disallow'. The combined set of rules for a specific fragment forms its second classifier, **situation**.

A third classifier of method fragments is their **rating**. Through the feedback of users, method fragments should be rated on several aspects, such as effectiveness, complexity, etc. Method fragments with a very low rating can in most cases be ignored, while in other cases, method fragments with a high rating are selected over similar method fragments with a low rating.

Although processes, capabilities and situational factors form a very solid ground for method fragment 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 needs to be able to express these preferences during the selection of alternative method fragments. The results from the interviews have indicated that product managers are not always willing to accept suggestions made to them by a machine. Therefore, the process should allow for differences in the amount of freedom that is provided. While it is generally a good idea to suggest one specific method fragment per capability, product managers should be at liberty to select another. This 'freedom-of-choice' has serious consequences for the system to be designed. In order to make the freedom given to product managers useful, we need to provide them with a sufficient amount of information for them to base their decision on.

The first source of information for this is of course the method fragment itself. Since every method fragment can be displayed in the form of a very readable PDD, product managers can use this diagram to form an initial mental image of its implications. This is possible since 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, product managers or process owners should be able to make a valid and well-argued choice regarding the method fragments that should be selected, and thus regarding the changes that should be made to the existing process.

6.1.4 Embedding of Process Advice

After a solution has been selected, the process of embedding or implementing the process advice varies depending on the amount of information that a company has given. The possibilities are limited when only maturity information is known, in contrast with the field of opportunities when full process information is given.

In any case, the initial part of the process can be the same for both situations, as this regards the elaboration of the chosen solution into steps. 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.

The splitting of solutions into steps is subject to several conditions. Solutions cannot be split into steps randomly. The major reason for this is the fact that we need to take dependencies into consideration. If a companies wants to increase the maturity level of its requirements gathering process from A to C, it does not make sense to implement automation before centralized registration. Instead, the first step should be to implement the activity related to level B, followed by an iteration in which level C is implemented.

In most cases, several capabilities can be implemented at the same time. However, to make iterations or steps more successful, it is probably wise to make sure that each step has some sort of goal, or a theme. This ensures a set of changes that is coherent. This way, the change-process seems less chaotic to the employee. This is important, as he or she will be the one performing the new process.

After the roadmap has been presented to the user and has been accepted, the implementation of it can start. In case that only maturity information is available, this process is fairly straightforward, as little support can be given. The changes that have been proposed need to be implemented in the company manually. 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 asset of method engineering, namely the assembly of method fragments. For each step, the method fragments it contains needs to be merged with the existing process. As this is a difficult task, it is probably best to do this fragment by fragment.

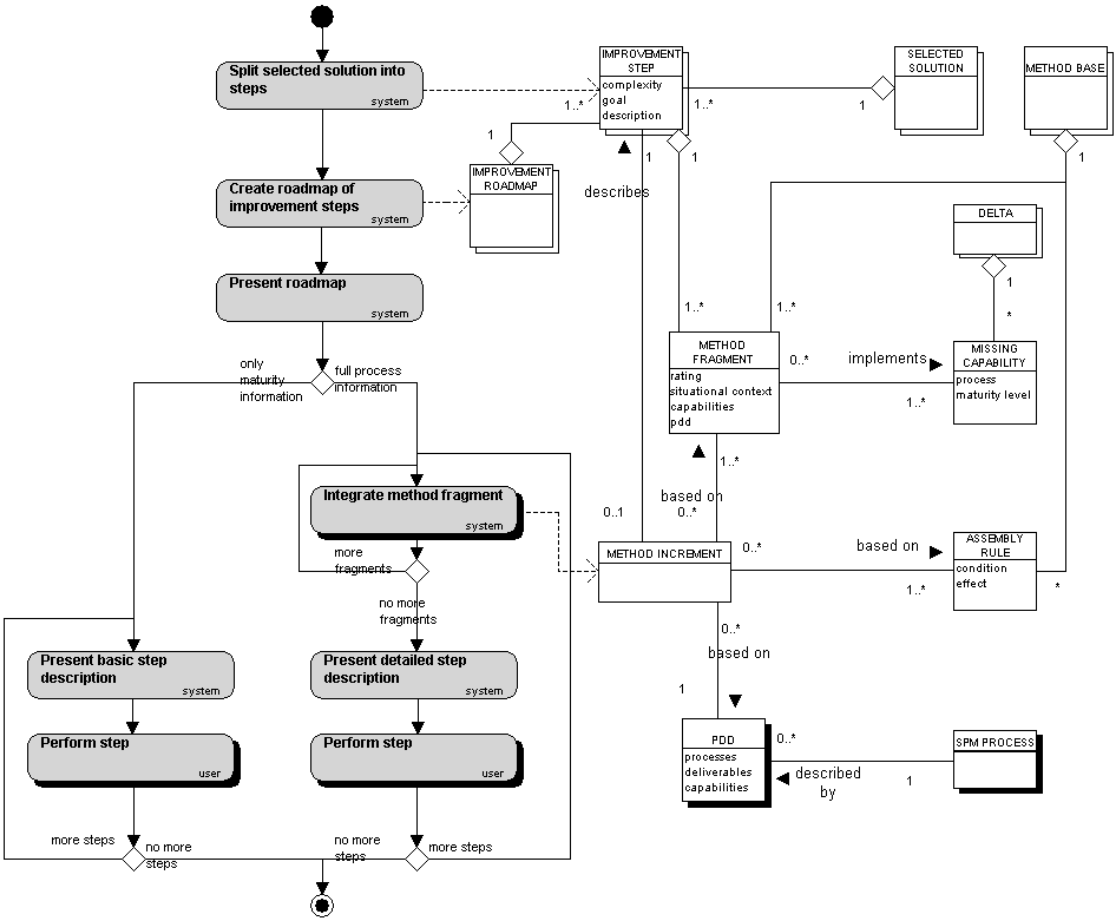


Figure 6.5: Embedding of process advice

A problem with this segmented approach, however, is the risk that some parts of the process get changed multiple times. This is unwanted, as this can lead to confusion among the people that need to perform the process. Therefore, already during the creation of the roadmap, the system should make sure that no such situations occur. This is also another argument for the statement that method fragments should be kept as small as possible. By preventing the usage of complex method fragments, the chance of overlap is made smaller, thereby increasing the chance of success of any algorithm that is charged with creating a coherent roadmap.

After the assembly of all the method fragments within a step into the original process, the changes can actually be performed within the company. To facilitate the change, the system can generate and/or update templates based on the original process and the new process. This activity is partially demonstrated in chapter 8, and consists of two steps (see figure 6.6).

If the companies original product management related documents, such as backlogs and roadmaps, are available to the system, 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. For any deliverables that are not available to the PSKI, templates can be generated base on the generated process description. As the PSKI might not always be aware of the type of document that is referenced by a deliverable, this needs to be specified manually in some cases. The generated templates should be directly usable within the new process.

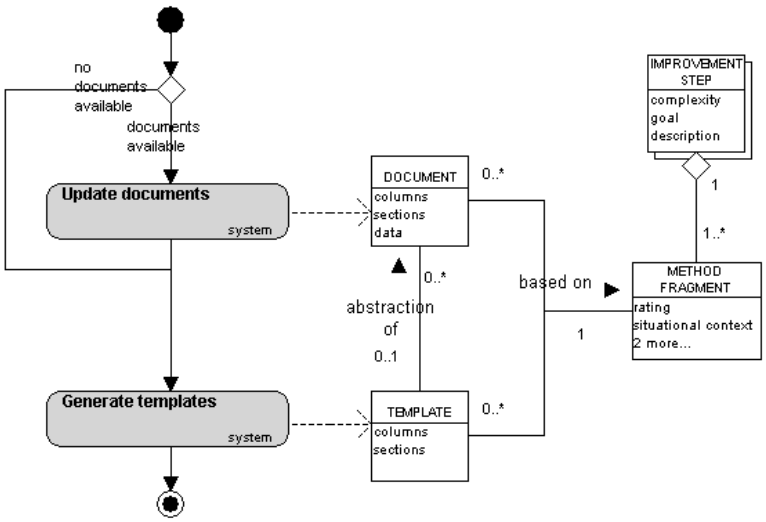


Figure 6.6: Template creation

6.1.5 Administration

The first requirement for a system such as the PSKI is that a repository exists which is filled with representations of these resources. This repository will be called the method base or knowledge base. The concept is derived from the field of method engineering. Several proposals have been done for the way in which the resource-representations can be stored in the database, including method fragments and method chunks. In this study we will adopt the former.

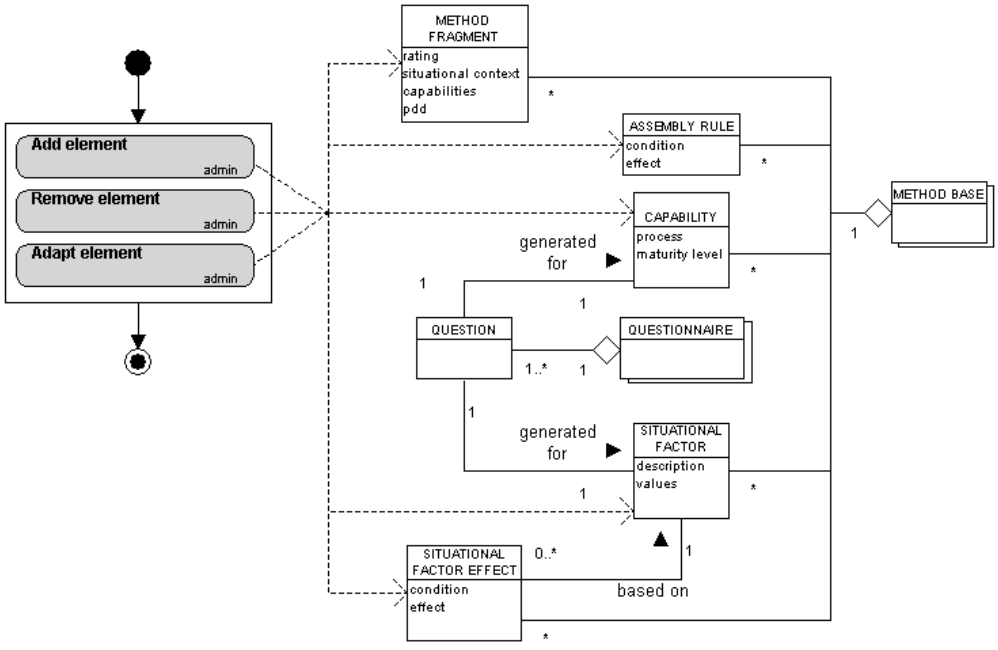


Figure 6.7: Administration

Up until now, method fragments were 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. These diagrams were always oriented towards humans. However, they were not stored in a machine-readable, interchangeable and standardized way. To make these diagrams usable in the PSKI, the manner in which PDD's are captured needs to be adjusted. A good solution for this purpose is MetaEdit+, a meta-modeling tool which can be used for both capturing meta-meta-models as well as the resulting meta-models. See chapter 7 for an elaboration of this technique.

By using MetaEdit+ as a tool for capturing PDD's, method fragment administration is now largely moved out of the PSKI design, except for the importing of the model. Assignment of the classification values can be done within MetaEdit+. On a more abstract level, the method base contains rules that govern the selection and assembly of method fragments, situational factors, capability descriptions and situational factors effects. As these data will change over time, it must be possible to govern them.

6.1.6 Knowledge Base Improvement

The system must support ways for inputting user-generated content such as reviews and advices. This is an essential part of the system, and users must be motivated to do so. This implies that, apart from the method fragments themselves, the method base should be filled with experiences as well. These experiences can be stored in several ways:

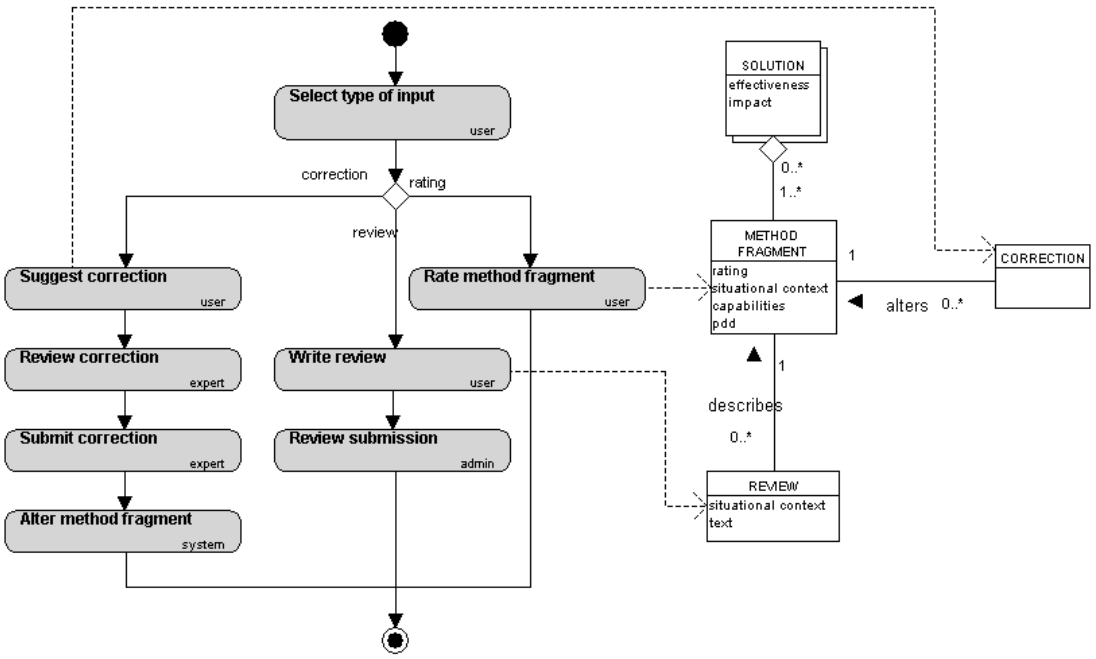


Figure 6.8: Knowledge Base Improvement

Data corrections This applies especially to the situational factors describing the applicability of method fragments. 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 method engineering approach, drastically.

Reviews Data corrections do not solve the problem of freedom required by product managers. For this, a more direct solution is required. One proposed solution is the possibility of writing reviews for certain method fragments. 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.

Recommendations The concept of reviews can be extended further by adopting a recommender-system as seen in many online stores. By taking the situational factors and maturity levels

into account, the system can make suggestions based on choices by similar companies.

To ensure the quality of user-submitted data, and to prevent 'data corrections' from occurring erroneously, rigorous checks need to be implemented. For reviews and recommendations, this can be done by the administrators of the system, or by its users. Nowadays, many websites and communities exist that are self-maintaining, i.e. the users of the system check the content themselves. When content is not appropriate or erroneous, it is flagged, after which it can be checked by the administrators. This reduces the burden of the administrators, and gives users an increased feeling of control. Censorship is not performed by the administrators, but by the users themselves.

For data corrections, an even more automated approach can be taken. Especially for small changes such as spelling errors and incorrect relationships between constructs, users should be able to submit these themselves. When a sufficient amount of users give the same feedback, it can be assumed that the correction is valid. The system can then automatically alter the method fragment according to it. For changes to descriptions or more complex changes, administrators will still need to analyze the feedback before changes are committed to the database.

6.2 MaaS: Method Execution

The PSKI does not adhere to the Method-as-a-Service philosophy unless it allows users to perform the method entirely online. This means that, instead of using various local software tools, all steps are performed online. The method modification aspect is essential for improving the **effectiveness** of SPM processes, but the method execution aspect ultimately allows major improvements in the **efficiency** of these processes, by taking away a large share of the burden of maintaining a complex IT infrastructure.

In the light of applying the Method-as-a-Service concept to the PSKI, we can already see the current website as a first version of this. As figure 6.9 demonstrates, it already contains a basic solution to the problems of method improvement and execution, by providing literature and method fragments on the one hand, and templates on the other.

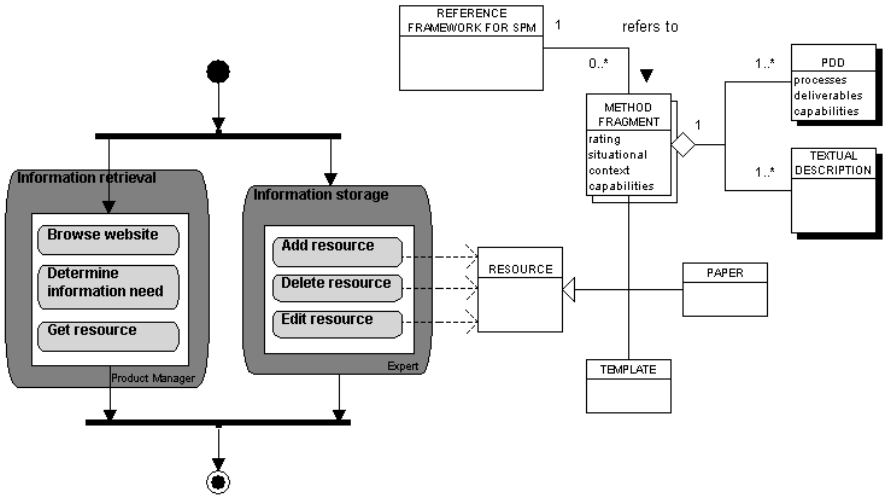


Figure 6.9: Current website

However, the website is only a static knowledge provider. In essence, it is a first approach to the method base that will form the core of the PSKI. On top of this, the method modification layer should be placed, as described in the previous chapter. Once all that functionality exists, the system can be extended with the method execution layer. This implies an integration of the functionality to describe a process, assess the process, adapt the process, and then perform the process (see figure 6.10).

In order to be able to do so, method fragments need to be correctly translated into an interface that offers the right set of tools for product managers and requirements engineers to perform their tasks. The creation and usage of templates is a core aspect of this. As is described in chapter 8, it is feasible to directly adapt existing online documents and templates based on method increments. Such online documents can for instance be placed on the cloud documents solution by Google, Google Docs. As this environment is accessible through an API, it can be integrated into any other system, such as the PSKI.

In addition to the translation of deliverables into documents and the management of these documents, the activities need to be correctly translated. Aspects that need to be taken into consideration here are the correct translation of access rights based on roles, the distinction between automated tasks and user input, the type of interface that is required for a certain set of activities, and the order of the activities, i.e. sequential, simultaneous, or a mix of both.

Currently, these aspects are not all derivable from the PDD's. To solve this, either stricter rules should be applied during the creation of PDD's, or additional models should be created for defining interface, access rights and business process. The former is not a good solution, as this would make the creation of PDD's too complex. The latter is similar compared to model-driven development solutions such as OO-Method (Pastor et al., 1997) and the web-based variant OOWS (Pastor et al., 2003). This would require the addition of several steps to the process for creating the required models, undermining an important aspect of the PSKI, namely the fact that it should be simple.

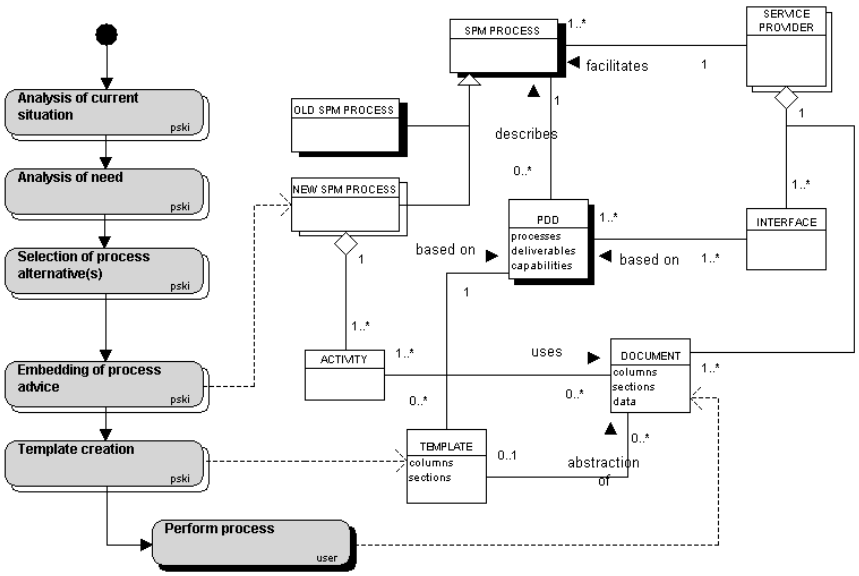


Figure 6.10: PSKI using a Method as a Service approach

To forgo this problem, an alternative solution could be developed, based on pattern recognition. The idea behind this is that certain patterns will exist in the PDD's of SPM processes that can be directly related to correct solutions for the interface. For instance, activities that are performed simultaneously should be connected to a tabbed interface, with a tab for each activity. Activities that are performed linearly can always be displayed as steps, allowing to go back an forward.

Such a solution would require no extra effort of the user. However, the possibilities of recognizing patterns are limited and it is very prone to modeling errors. Therefore, a user should always be able to alter the interface for a given process. Alternative interface elements should be provided for this by the system. The same holds for the generation of documents based on deliverables. As it is not always possible to derive the required file-type for a deliverable, the user should have the option to change this manually.

To capture all the requirements of the translation from method description to interface, a meta-model should be defined describing all possible translations for every construct and pattern. This falls however outside the scope of this thesis.

Chapter 7

Method Fragment Storage

The process-deliverable diagram created by Weerd & Brinkkemper (2008), based on the work by Saeki (2003), is a strong instrument for the description and communication of software development processes. The two combined views, process and deliverable, provide an easy-to-use and clear view of all the main parts of a process. We have seen very successful applications of it in several projects. This includes the visual description of method increments by Weerd et al. (2007).

During the design of the modeling-technique, the focus has been placed mainly on the visual aspect of the diagram, i.e. its effectiveness in communicating the most important aspects of processes. This has resulted in a diagramming-technique that is both powerful in communicating complex processes as well as simple in its use. However, a high usability for visualization purposes does not necessarily mean high usability when using the diagram for other purposes.

When applied to the computational domain, a modeling technique has different requirements. For one, readability is replaced in favor of completeness. Also, complexity is no longer a problem. Instead, structure becomes an essential requirement. This means that we should distinguish between these two purposes when working with process-deliverable diagrams. In the best case, the diagramming-technique will retain all of its usability aspects while becoming effective for computational purposes.

7.1 Process-Deliverable Diagrams in MetaEdit+

One of the biggest advantages of a domain-specific language is that it allows for creating valid diagrams through an easy to use interface. This increases the modeler's efficiency and inhibits errors later in the process.

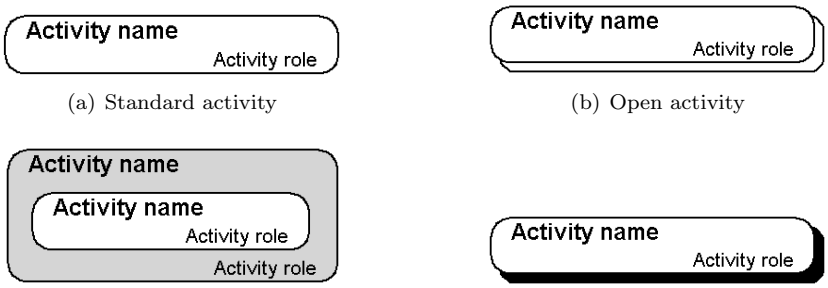
7.1.1 Objects

In order to maintain all of the expressiveness of the original Process-Deliverable Diagrams in Visio, we need eight object types. This is a reduction compared to the original diagram, resulting from MetaEdit+'s context awareness.

One of the most important objects types in the meta-model is the **activity**. Originally, four types of activities existed: standard activities, open activities, sub activities and closed activities. These four types served only the purpose of visual distinction, and did not differ much in semantics. In the MetaEdit+ implementation, this amount of types is therefore reduced to one object type, adjusting its visualization based on the context. This is done by using MetaEdit+'s Reporting Language (MERL).

The visualization has not changed significantly. Figure 7.1 shows the four states of an activity.

The second main object-type in the meta-model is the **concept**. Similar to the activity-type, the original PDD-implementation counted three separate concept types: standard concept, open concept and closed concept. Again, these three types only differed in their visualization, and not in their semantics. Therefore, they have been consolidated into one object-type, changing its visualization based on the context. The three states are shown in figure 7.2.



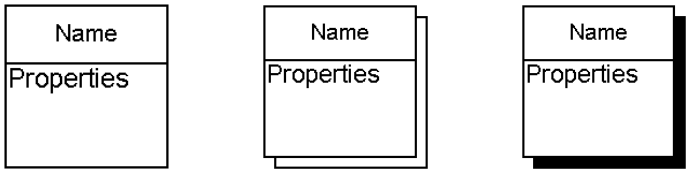
(a) Standard activity

(b) Open activity

(c) Sub activity within open activity

(d) Closed activity

Figure 7.1: 'Activity' object-type visualizations in MetaEdit+



(a) Standard concept

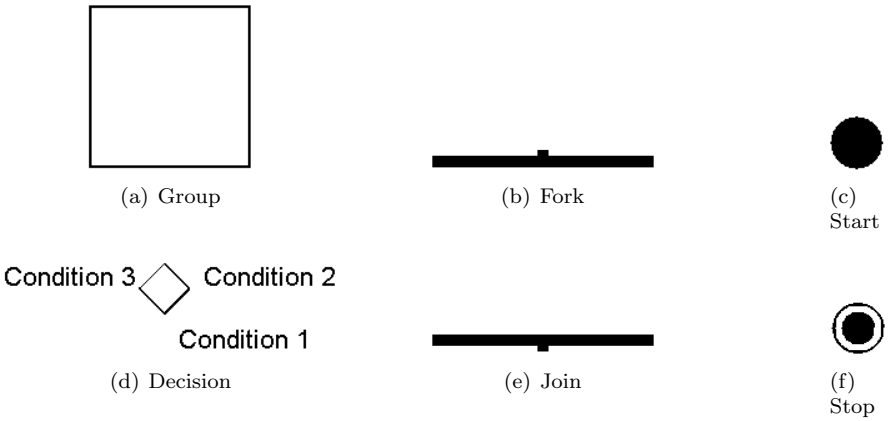
(b) Open activity

(c) Closed activity

Figure 7.2: 'Concept' object-type visualizations in MetaEdit+

Again, the visualizations have not changed significantly. However, the deliverables now show a maximum of three properties. If more properties are added to a concept, it will show only two properties and a row called '[X] more...'.

The divider that was used to indicate a section of unordered activities in the original implementation of the PDD has been replaced by a **group** object-type. This allows unordered activities to be displayed outside an open activity as well. A group is indicated by a white box with black border (figure 7.3(a)). All activities within a group can be executed in random order.



(a) Group

(b) Fork

(c) Start

(d) Decision

(e) Join

(f) Stop

Figure 7.3: Remaining object-types in MetaEdit+

The remaining object-types do not differ from their original implementations. For the sake of completeness, they are shown here as well: **start** (7.3(c)), **stop** (7.3(f)), **decision** (7.3(d)), **fork** (7.3(b)) and **join** (7.3(e)).

7.1.2 Relationships

The relationships defined in the original meta-model of the PDD have not been changed. Their implementation in MetaEdit+ is similar to that of the Visio-implementation. This means that the relationships available within MetaEdit+ are **transition**, for indicating the flow from one activity to the next, **generalization**, for expressing a relationship between a general and a specific concept, **association**, for defining a structural relationship between concepts, **aggregation**, for representing the relation between a complex concept and its parts, and **result**, for connecting the process-side with the deliverable-side. The only addition has been the **choice** relationship, which is a subtype of 'transition' and is used for indicating different conditional paths after a decision.

7.1.3 Rules and constraints

For ensuring the creation of valid diagrams, MetaEdit+ allows the definition of *bindings*. These bindings define what relations are possible within the diagrams. For the PDD-diagram, the following bindings were defined:

Relationship	**Start (role)**	**End (role)**
Aggregation	1 concept (Whole)	1..N concepts (Part)
Association	1 concept	1 concept
Choice	1 decision (From)	1 activity (To) 1 decision (To) 1 fork (To) 1 join (To) 1 group (To) 1 stop (To)
Control flow	1 activity (From) 1 decision (From) 1 group (From) 1 join (From)	1 stop (To)
	1 activity (From) 1 group (From) 1 join (From)	1 activity (To) 1 decision (To) 1 fork (To) 1 join (To) 1 group (To)
	1 fork (From)	1 activity (To) 1 decision (To) 1 fork (To) 1 group (To)
	1 start (From)	1 activity (To) 1 decision (To) 1 fork (To) 1 group (To)
Generalization	1 concept (Superclass)	1..N concepts (Specialization)
Result	1..N activities (Result from)	1..N concepts (Result in)

Table 7.1: Valid relationships within a PDD diagram

In addition to these valid relationships, MetaEdit+ allows the definition of constraints for connectivity, occurrence, ports and uniqueness. To complete the definition of the PDD meta-model in MetaEdit+, the following rules were defined:

- An *activity* may be in at most 1 *from* role
- A *join* may be in at most 1 *from* role

- A *group* may be in at most 1 *from* role
- *Start* may be in at most 1 *Control flow* relationship

Combined with the bindings above, these constraints ensure valid diagrams. However, it must be noted that the rules are not very strict. Given the meta-model of process-deliverable diagrams, constraints should be added that restrict the amount of incoming and/or outgoing flows for the 'activities', 'decisions', 'forks', 'groups' and 'stops' to 1. However, this would make the solution considerably less usable, as more complex solutions would be needed to prevent such situations.

For example, disallowing more than 1 control flow to a 'stop' construct would require adding a 'join' construct to the diagram, even though a 'fork' might not have been used to created multiple paths (this happens when using decisions). Such a 'join' would complicate the diagram and make it less readable.

7.2 Method Increments

The method increments that were mentioned before need to be modeled in MetaEdit+ as well. What is needed is a solution in which new versions of diagrams can be created based on previous versions, and in which changes in version N+1 are visually indicated.

MetaEdit+ does not support version control out of the box. Therefore, manual adjustments must be made to the implementation of PDD's. In order to allow for the modeling of increments in MetaEdit+, a 'version' attribute should be added to each activity, deliverable and property. This attribute is a construct consisting of two sub-attributes. The first one is the version in which the fragment was adapted. The second argument is the action, indicating whether the element was added, removed or changed.

Initially, each element has a default version attribute, with the values 'added' in version '1'. These are the values that are most common, as the majority of constructs are generally created in version 1. After the initial diagram has been created, the evolution of the diagram can be described through increments. To create an increment, the user creates a shallow copy of the current model (version N), and uses this as a basis for all manipulations for version N+1.

In order to add a construct to a diagram, one does so in the usual way, after which the new constructs' 'version'-attribute is set to 'added' in version '[N+1]'. This approach is very similar to the normal approach of adding constructs.

For deletion of a construct, a somewhat different approach is required. In this case, the unwanted construct should not be deleted from the construct altogether, but instead be marked with a 'version'-attribute indicating 'removed' in '[N+1]'.

To avoid cluttering of the diagram, constructs that were set to 'removed' in version [N+1] should be deleted altogether in increment [N+2]. This does not pose a problem, as the construct is no longer of interest. Because the construct is still present in earlier versions of the diagram, it still persists in the database. Therefore, no historical data is lost.

For replacing a construct with another one, one should follow the previous tasks in the order of 'delete' and 'add'. The user has the choice to either place the new construct next to the old one, increasing the amount of information the increment displays, or replace it with another construct by using the appropriate function in MetaEdit+.

Based on the version-attributes, constructs can be given an appropriate color. Constructs that have been marked as 'added' in diagram version '[N+1]' should not be shown in version '[N]' and lower, shown with a green color in version '[N+1]', and shown normally in versions above '[N+1]'.

Constructs that have been marked as 'removed' in diagram version '[N+1]' should be shown normally in versions '[N]' and lower, and shown with a red color in version '[N+1]'. In versions '[N+2]' and higher, the construct should be removed altogether.

As MetaEdit+ does not support version control out-of-the-box, a lot of manual work is needed in order to implement the notion of method increments. Especially in the case of removing or replacing a construct, the steps that are needed can be a bit obscure.

Another serious shortcoming is the fact that it is not quite possible to add the same functionality to relationships, as their appearance can not easily be changed according to parameters. Especially when adding new relationships between existing constructs, this can be a problem.

Unfortunately, these problems are not easy to solve in MetaEdit+. Nonetheless, the solution described here provides for a fully specialized tool that is in general easy to use, and that provides some very important features that are not available in Microsoft Visio.

7.3 XML generation with MERL

The current technique for storing process-deliverable diagrams is a very unsatisfying one, as it is in Microsoft's proprietary Visio-format. This format is closed, making it very hard to share meta-model data between several applications. Although it is possible to export diagrams to XML, the code that is generated is too complex, making it a hardly useful solution.

Also, Visio does not allow for the description of a diagram, i.e. a meta-meta-model. Therefore, the diagrams do not contain any semantic information, making sharing the data useless, even if the file-format were open.

When looking for a storage format that is more open, making it interchangeable and readable at the same time, XML is an obvious choice. "XML offers a medium that is platform, network and technology neutral, independent of underlying programming languages, and readable by machines as well as humans." (Bakker & Jain, 2002)

With the selection of XML as the new storage format for process-deliverable diagrams, the next question relates to the way of creating XML documents from a PDD. When using MetaEdit+ for the creation of the diagrams, there are two options available; either diagrams are exported directly to XML by using MetaEdit+'s built-in XML exporting feature, or diagrams are exported 'manually' by employing the MERL processing language.

Obviously, the first alternative is the easiest approach, requiring no additional work upfront. However, MetaEdit+'s data-format is very restrictive when viewed from the point of view of our PDD implementation. The structure of the documents MetaEdit+ produces follows the GOPRR-model (Graph, Object, Port, Role, Relationship). Although this is an approach very suitable for describing such models, it forgoes one specific feature that is used by our implementation, which is the usage of representational data for determining the relationship between complex activities and their contents, and group indicators and their contents. As this data is not directly represented in the native XML format, it would be very hard to correctly reconstruct the original model in another tool. This seriously impacts the usability of the format, making it the less favorite choice.

Instead, there is the option to use MERL, or the MetaEdit R* Language, for converting diagrams into any format required. Although the language in itself is not very powerful, some tricks will make any conversion to a textual format possible. This gives us the liberty, and requirement, to determine a custom data-model that is more applicable to the domain.

On the other hand, it is not possible to import custom XML-files into MetaEdit+ unless they are in the GOPRR format. To circumvent this problem, an XSLT-stylesheet could be written to convert the custom XML-documents back to the native format. In order to ensure correct representation, this would however require the inclusion of MetaEdit+'s own coordinate system.

To allow XML-exporting, a generator had to be defined using the MERL language. The code for this generator consists of 160 lines of code.

7.4 Latex generation with MERL

The possibility to export PDD's to XML will be very useful for the creation of the PSKI. However, PDD still serve a communicational role as well. For this reason, it is very useful to be able to export PDD's to other formats. Activity tables, concept tables and maturity matrices can be derived directly from the model. Using MERL, they can then be exported to other formats such as HTML or latex.

As this thesis and all case study reports have been written in latex, it was a logical choice to allow exporting to this format. Therefore, generators have been written that allow exportation of the activity & concept tables of a PDD, and the generation of a filled-in capability matrix based on all PDD's of a company's product management process. The code for these generators consist of 235 lines of code.

Chapter 8

Method Increments: Proof of Concept

The solution proposed in this thesis relies heavily on techniques and notions from the method engineering field. Some of these techniques, such as employing process-deliverable diagrams for meta-modeling purposes, have been described and practiced extensively. Also, for method assembly several approaches are available, as described in chapter 3.3. Others, however, are still relatively un-elaborated.

The notion that the solution proposed here relies on the most is that of method increments. Weerd et al. (2007) already described the main concepts that are used when creating method increments, but the technique has not been put into practice yet. Therefore, the final chapter of this thesis will describe the creation of a proof-of-concept for method increments.

The proof-of-concept (PoC) focuses on the use of method increments for template generation. As described earlier, changes made to a process through the PSKI should be facilitated and supported as much as possible, to ensure the success of the evolution. One technique for doing so is providing automated templates based on the deliverables of a process. In the case of minor changes, changes made to a template can be incorporated in the original company documents, preserving any data already existing.

8.1 Platform

For the creation of the PoC, several technical choices needed to be made. Although it is at this point unclear whether the current implementation will form the basis for the further implementation of the PSKI, it is important that choices regarding the platform and the architecture are made carefully, so as to obtain a reliable solution.

The platform on which the PoC is implemented is Google AppEngine[1]. AppEngine provides a complete development stack for building and hosting web applications. The platform can be used free-of-charge, and no additional web-server is needed, since all applications are hosted in the Google-'cloud'. This makes it a very pragmatic solution for this project. Also, through several available API's, applications can get access to documents hosted in the cloud.

Applications hosted on AppEngine can be implemented in one of two languages; Java[2] and Python[3]. Making a choice for one these two is rather trivial, and very much dependent on the preferences of the programmer. Still, both languages have a distinct set of advantages. Some very strong features of Java are that it is object-oriented, it has automated memory allocation and garbage collection, it is platform-independent (not really relevant in this case), and it is robust, due to static typing and strong error-handling.

However, because java-code needs to be compiled into machine-readable byte-code, development can become a rather tedious process. Especially during the development of a proof-of-concept, it

[1] http://code.google.com/appengine/
[2] http://java.sun.com
[3] http://www.python.org

is important to be able to quickly implement and test new features. The compilation step stands in the way of this. This also applies to the robustness of the language. Although much of Java's architecture are very interesting on the long run, they can slow down initial development.

Python, on the other hand, has the advantage that it is a very easy language, making it possible to rapidly create feature-rich applications. Also, there is no need for compilation as it is an interpreted language, much like PHP or JavaScript. The language itself is rather concise, making it possible to express a lot of information using small amounts of code. Combined with dynamic typing, this gives a boost to productivity. However, it is important to keep in mind that the advantages that the language has in the beginning might become problematic once the codebase starts growing.

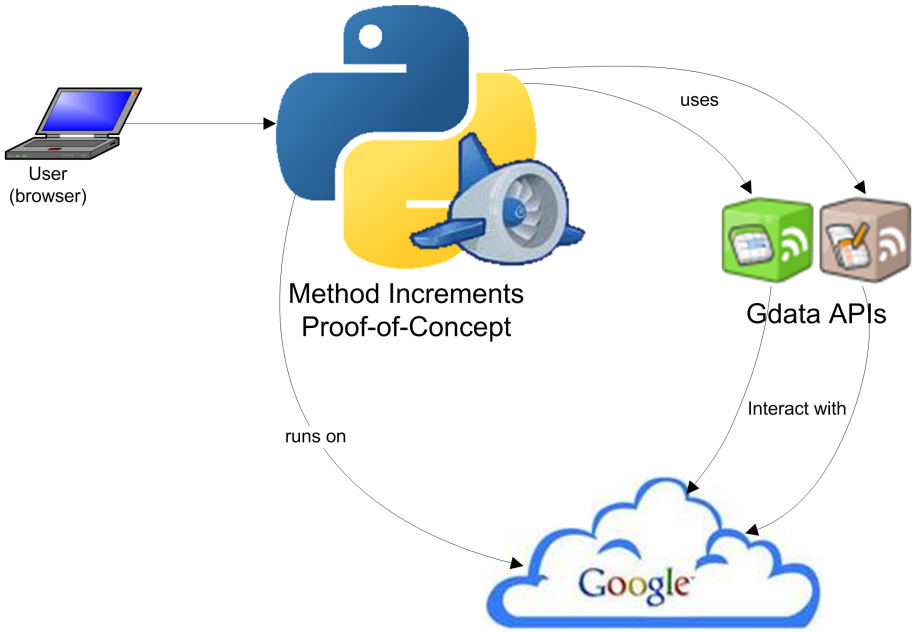


Figure 8.1: Software architecture of the proof-of-concept

Based on these arguments, Python has been selected for the implementation of the PoC. The IDE used during development is Eclipse[4] with the PyDev plug-in[5], enabling code-coloring, syntax-checking and code-completion. The user interface is written in HTML with JavaScript for structuring, and CSS for styling. Figure 8.1 visualizes the software architecture of the PoC.

[4]http://www.eclipse.org
[5]http://www.pydev.org

8.2 Functionality

As said, the functionality of the PoC is limited, and restricted to showing the possibilities of generating templates based on method increments. When a user enters the website, he/she (from now on, 'he' will be used) is shown a landing page with information about the project. In the top-right, the user gets the option to log in to the system. Only users with a valid Google-account can currently log in.

Once the user is logged in, he is presented a menu where he can select 'method increment'. This will guide him to a page showing a list of all the spreadsheets in the database (the 'database' is in this case the set of Google documents in a Google-account created specifically for this application). The items in this list can be expanded to show the worksheets per spreadsheet. Once the user clicks on a worksheet, he is directed to another page.

On this page, he is presented with a deliverable-description, based on the columns in the worksheet. He can then adapt this deliverable to his wishes, by adding, removing and renaming items. Once the user submits his changes, the system calculates the differences between the two deliverables. Based on the delta, the worksheet is adapted, preserving any existing data. The spreadsheet can then be downloaded, after which the process ends.

8.3 Example

To further describe the proof-of-concept, we will walk through an example step-by-step. The example that will be used is the transition from prioritization based on the MoSCoW technique to prioritization using the Wiegers matrix.

Both of these techniques have been around quite a while, and are well-known and widely used within industry. The MoSCoW technique is one of the core techniques of the DSDM software development method (Stapleton, 1999), but it was proposed a few years earlier by Clegg & Barker (1994). It encompasses a very straightforward approach to requirements prioritization, by using a four-term classification, with the four capitals in MoSCoW standing for 'must have', 'should have', 'could have' and 'won't have'. The meta-model for the MoSCoW technique is shown in figure 8.2.

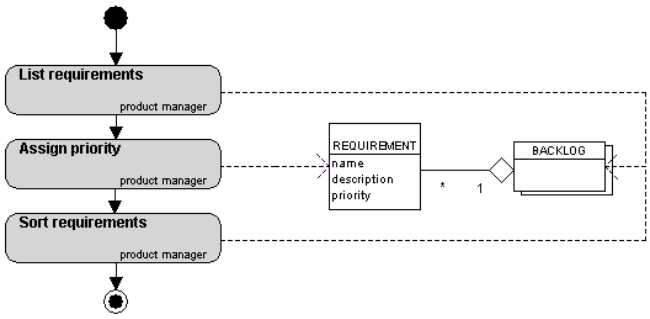


Figure 8.2: Prototype

The Wiegers prioritization technique is a bit more advanced, and takes into consideration multiple perspectives. Although the technique can be made simpler or more complex at will, the most common factors are relative cost, relative benefit, relative risk and relative penalty. Based on these factors, a priority score can be calculated. The meta-model for the Wiegers prioritization technique is shown in figure 8.3. It is depicted as an increment based on figure 8.2, with the green borders indicating added or changed constructs and the red borders indicating removed constructs.

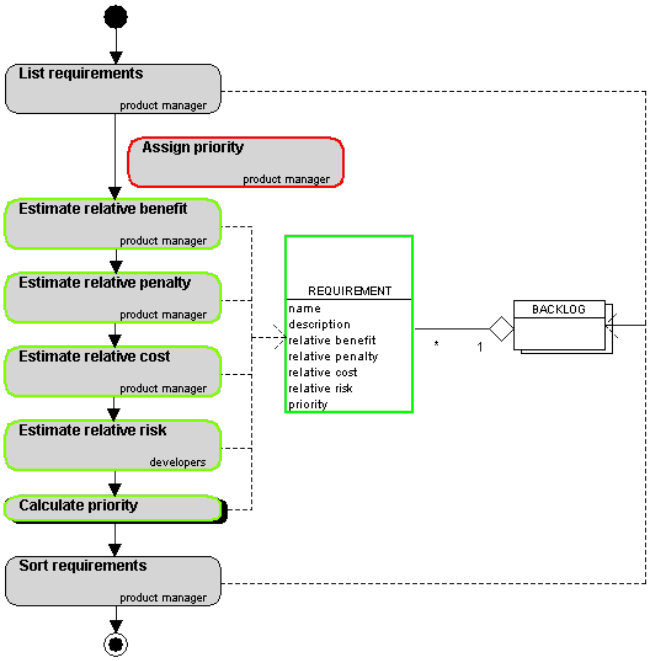


Figure 8.3: Prototype

In this example, we are going to transform a spreadsheet used for MoSCoW-based prioritization into a spreadsheet for Wiegers-based prioritization. For this, we first need to log in to the application. This is done using the standard Google-interface (figure 8.4).

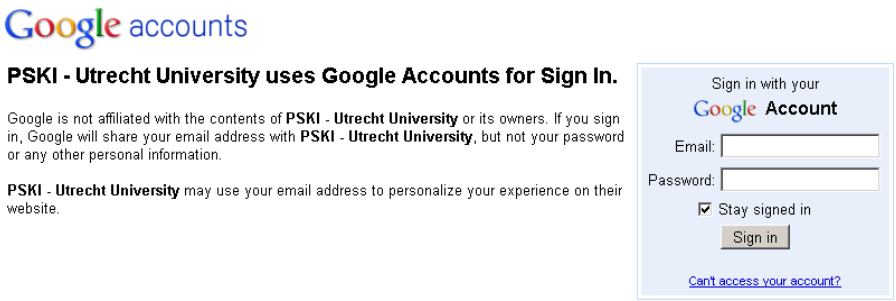


Figure 8.4: Example: logging in to the application

When we have entered the application, we are presented a list of available documents. We select the document that contains the spreadsheet for prioritization using the MoSCoW-technique. Figure 8.5 shows an excerpt of the document.

	A	B	C	D	E
1	**Name**	**Description**	**Priority**		
2	Fix error x	Users have trouble accessing page A	Must have		
3	Add feature y	Users need to be able to create B	Should have		
4	Remove table z	Table z is no longer needed because C	Could have		
5					

Figure 8.5: Example: the original document

Based on the deliverable 'requirement', shown in figure 8.3, we then need to alter the deliverable shown in the application. This can be done using the provided buttons for adding, removing, renaming and moving attributes. Figure 8.6 shows how this looks in the application.

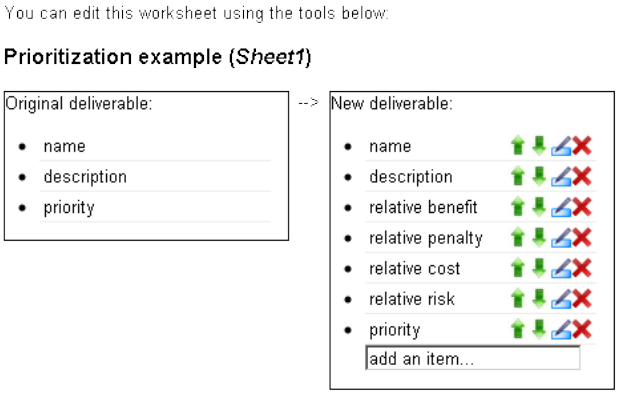


Figure 8.6: Example: updating the meta-model

Based on the changes we have made in the application interface, the original document is updated accordingly. This means that the columns now represent the new set of attributes, maintaining any original data. Figure 8.7 shows an excerpt of the resulting document.

	A	B	C	D	E	F	G	H
1	**Name**	**Description**	**Relative benefit**	**Relative penalty**	**Relative cost**	**Relative risk**	**Priority**	
2	Fix error x	Users have trouble accessing page A						
3	Add feature y	Users need to be able to create B						
4	Remove table z	Table z is no longer needed because C						
5								
6								

Figure 8.7: Example: the updated document

The proof-of-concept shown above is not complete. Its functionality is limited to changing templates based on changes made to the meta-model. It does not allow the manipulation of data, format, or formulas. However, for the sake of this thesis, it suffices in showing the possibilities of employing web-techniques for manipulating templates automatically.

Chapter 9

Conclusions

9.1 Main Results

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 software product management 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 propose the creation of a knowledge infrastructure for software product management. In this thesis, we have made a first move towards this goal.

First of all, we have shown how process-deliverable can be used to perform assessments of the SPM processes by combining them with domain knowledge extracted from the reference framework for SPM and the SPM capability matrix (research question 1).

Additionaly, we have demonstrated how these PDD's, or the method fragments they represent, can be modelled and stored effectively using MetaEdit+. This solution is essential for the creation of a usable method-base, and very useful during the assessment phase (research question 5).

To the question on the current state of software productmanagement, we have attempted to give an overview of the currently implemented processes by performing case studies at six different small-to-medium Dutch productsoftware companies (research question 2). The result shows that most product management activities are generally performed at a rather low level, leaving a lot of room for improvement.

From the results of the case studies, we extracted some of the most common problems (research question 3). These problems have been generalized from several specific occurrences at each company, and include a lack of formal processes, missing activities and sub-optimal maturity levels.

The identified problems have provided inspiration for the elaboration of a detailed proposal for the product software knowledge infrastructure (research question 4). In a set of process-deliverable diagrams, the entire workflow is described. By providing a method-base with a large variety of methods and techniques, in combination with experience reports from other professionals, we hope that such a system will increase the maturity of product software companies, allowing the development of products with higher quality and more revenue.

Another important concept that is part of the proposed knowledge infrastructure is the method increment. In this thesis, we have tried to demonstrate how templates can be updated according to method increments (research question 6). For this purpose, a proof-of-concept has been created that demonstrates a basic operationalization of method increments in combination with template creation, in the form of a web application based on the Google platform.

The above answers to the research questions posed at the beginning of this thesis form a first step towards the realization of an effective solution for improving the function of software product management.

9.2 Reflection and Discussion

Although several important issues have been elaborated in this thesis, the product software knowledge infrastructure that we envision is far from being operational. The complexity of such a system was already known, and this thesis only strengthens the idea that we are dealing with an advanced concept requiring a lot of research effort.

By zooming in on the earlier developed vision of the PSKI, we have gained a better understanding of the requirements. However, at this point, the system remains fairly conceptual. From this point onwards, each of the areas of the PSKI needs to be addressed in detail, putting together the puzzle piece by piece. Expertise in several areas will be needed for this, as each part of the PSKI has very specific challenges, ranging from linguistic analysis for method assembly and data-optimization for the method base.

An important factor that can never be left out during the elaboration is the fact that the purpose of the PSKI is the improvement of SPM 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.

However, if it is done right, than the PSKI has great potential value. We believe that this solution can increase the maturity of the software industry significantly by providing requirements managers, product managers, CTO's and the like with the right tools to optimize their SPM process. In turn, product quality will rise and costs will fall.

Unfortunately, the detailed PSKI that is presented in this thesis has not been fully validated yet. As no concrete system exists yet, doing so would have involved asking potential users to imagine themselves using such a system. This is a tremendous effort, especially due to the complexity of it, and would likely not have resulted in a valid response. However, as development continues, the user should not be forgotten. Instead, at several points in time, his opinion should be asked and corrections should be made according to it.

Ultimately, the largest challenge that we face is of an academic nature. By developing the PSKI, we state that processes can be modeled, 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 thesis, these topics have hardly been touched upon. However, they will become very clear once work on the specific parts will begin, and it is important to balance both of them.

9.3 Further Research

The results described in this thesis form a strong basis for the creation of a useful product software knowledge infrastructure. However, many issues need to be solved before we have reached this state.

9.3.1 Method Engineering

One of the major questions that needs to be answered is how, on a conceptual level, method fragments can be trimmed down, extended, adapted or replaced. Also, we will have to determine how the deliverables that are related to the process can be adapted. Whenever a process changes in some way, the deliverables that are related to it will change as well.

Solutions are beginning to appear, but none of them are sufficiently advanced for the purpose of the PSKI. Therefor, we need to determine how methods can change, what are the most common increments, and how method fragments can be analyzed automatically in order to determine the way in which they can be combined.

Specific research questions on this topic include how method fragments can be divided into small, molecular pieces, and how similar activities and deliverables can be identified.

9.3.2 Assessment and Selection

One of the main advantages of the envisioned PSKI is the situational factor. Companies should be able to obtain a customized product management method based on the specifics of their situation.

Therefor, another major issue in relation to the PSKI will be the selection of appropriate method fragments based on situational factors.

Even before any method engineering takes place, we need to be able to determine the exact need of a company. Research regarding the assessment is already taking place, with very promising results. With the resulting process need in hand, we then face the problem of matching this need to method fragments that will fill this need. To make this matching possible, selection criteria will need to be devised. Ideas that have been proposed in this thesis are capabilities, situational indicators and ratings. However, a structure approach is needed, and more selection criteria might need to be devised.

The selection and combination of method fragments will for a large share fall within the domain of linguistics. In order to determine similarities between method fragments, which can then be used to determine compatibility, linguistic analysis will most likely be the most effective means next to meta-information.

9.3.3 Implementation

The above issues are related to method fragments and their deliverables on a conceptual level. However, to be able to use these concepts in a PSKI, we will need to understand what is the best way to store the method fragments. This issue is for a large part solved by the technique for storing and creating PDD's that is proposed in this thesis. Nonetheless, we need to keep looking for alternative approaches, as the current solution is not perfect. Preferably, a platform-independent solution is found.

Additionally, choices need to be made regarding the platform on which the PSKI will be implemented. Obviously, this is hardly a scientific dilemma. However, for the success of the PSKI, it is important to make the right decisions, based on a rigorous analysis of the context, the users, and their wishes.

9.3.4 Templates

A major improvement of the PSKI over the current software product management website is the addition of directly usable templates, dynamically generated based on the situational method of a company. Generation of such templates saves companies a significant amount of time and facilitates the adoption of a new or adjusted product management method. In this thesis, a first proof-of-concept has been created, which demonstrates that such functionality is feasible using modern web-techniques.

A next step is the elaboration of this proof-of-concept into a solution that is able to evolve or create templates based on actual method increments. Also, as the range of tools that are used within companies is very wide, more types of templates need to be adaptable. This includes not only spreadsheets and text-documents, but also tools such as JIRA and Sharepoint. Research is needed to determine to what extent this is possible.

9.3.5 Method as a Service

Also, by using new web-technologies, a wide range of possibilities becomes available for improving the visibility and transparency of all aspects of software product management. By visualizing products, milestones, requirements and stakeholders in new ways, the efficiency and effectiveness of product managers might be significantly improved. This is especially true when we evolve the PSKI towards Method-as-a-Service. With the addition of maturity levels, templates and visualizations to the original PSKI, and the formalization of method storage, selection and combination, this is a logical next step. Through this, a large portion of the burden that is related to the use of certain Software Product Management methods can be taken away.

There are still a lot of unknown factors in this respect. For example, the relation between method fragments and ´services´ will need to be investigated. One can think of automatic translation of method fragments into appropriate interfaces. For instance, recurring patterns on the activity-side might be translatable to well-known interface elements such as tabs, wizards, etc. On the deliverable side, specific patterns might indicate spreadsheets, matrices, etc.

9.3.6 Social Issues

To conclude, more research is needed on the social aspect of the PSKI. As the system deals with processes that are performed by people, it raises questions regarding the acceptance of changes to these processes, the maximum amount of changes that can be introduced at once, etcetera.

Investigating all the items above will take the product software knowledge infrastructure out of the conceptual domain, and make it a tool that helps product software companies in creating a solid, mature product management process.

9.4 Acknowledgements

I would like to thank Jolita Ralité for her hospitality during my stay in Geneva, Switzerland. My stay at the University of Geneva has been very fruitful, for which my appreciations.

Also, I thank my supervisors, Inge van de Weerd and Sjaak Brinkkemper, for their guidance and comments, always to the point and very supportive.

The final line has no dot
Dots indicate endings
I´ll put a name there instead
Jiska de Ligt

References

Abramovici, M., & Soeg, O. C. (2002). *Status and Development Trends of Product Lifecycle Management Systems.* Germany: Ruhr-University of Bochum.

Agerfalk, P. J., Brinkkemper, S., Gonzalez-Perez, C., Henderson-Sellers, B., Karlsson, F., Kelly, S., et al. (2007). Situational Method Engineering: Fundamentals and Experiences. In (pp. 359–368). Springer Boston.

Agerfalk, P. J., & Fitzgerald, B. (2006). Advanced Topics in Database Research Vol. 5. In K. Siau (Ed.), (pp. 63–78). Idea Group.

Aguiar, M., & Gopinath, G. (2007). Emerging Market Business Cycles: The Cycle Is the Trend. *Journal of Political Economy*, *115*(1), 69–102.

Akker, M. van den, Brinkkemper, S., Diepen, G. van, & Versendaal, J. (2005). Flexible Release Planning Using Integer Linear Programming. In *Proceedings of the 11th international workshop on requirements engineering for software quality (refsq'05).*

Ali, R., Yu, Y., Chitchyan, R., Nhlabatsi, A., & Giorgini, P. (2009). Towards a Unified Framework for Contextual Variability in Requirements. In *Proceedings of the 3rd international workshop on software product management.*

Appel, W. (2000). Architecture Capability Assessment. *Enterprise Planning and Architecture Strategies*, *4*(7).

Ardis, M., Daley, N., Hoffman, D. M., Siy, H., & Weiss, D. (2000). Software Poduct Lines: a Case Study. *Software Practice and Experience*, *30*(7), 825–847.

Aydin, M., & Harmsen, F. (2002). Making a method work for a project situation in the context of CMM. *Product focused software process improvement*, 158–171.

Babar, M. A., Verner, J. M., & Nguyen, P. T. (2007). Establishing and maintaining trust in software outsourcing relationships: An empirical investigation. *Journal of Systems and Software*, *80*(9), 1438–1449.

Bakker, J. L., & Jain, R. (2002). Next generation service creation using XML scripting languages. In *Ieee international conference on communications* (Vol. 4, pp. 2001–2007).

Bekkers, W., Spruit, M., Weerd, I. van de, Vliet, R. van, & Mahieu, A. (2010). A Situational Assessment Method for Software Product Management. In *Proceedings of ecis2010 (accepted).*

Bekkers, W., Weerd, I. van de, Brinkkemper, S., & Mahieu, A. (2008). The Influence of Situational Factors in Software Product Management: An Empirical Study. In *Iwspm '08: Proceedings of the 2008 second international workshop on software product management* (pp. 41–48). Washington, DC, USA: IEEE Computer Society.

Benbasat, I., Goldstein, D. K., & Mead, M. (1987). The Case Research Strategy in Studies of Information Systems. *MIS Quarterly*, *11*(3), 369–386.

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

Brinkkemper, S., Saeki, M., & Harmsen, F. (1999). Meta-modelling based assembly techniques for situational method engineering. *Information Systems*, *24*(3), 209–228.

Brinkkemper, S., Saeki, M., & Harmsen, F. (2001). A Method Engineering Language for the Description of Systems Development Methods. In *Caise '01: Proceedings of the 13th international conference on advanced information systems engineering* (pp. 473–476). London, UK: Springer-Verlag.

Carmel, E., & Abbott, P. (2006). Configurations of global software development: offshore versus nearshore. In *Gsd '06: Proceedings of the 2006 international workshop on global software development for the practitioner* (pp. 3–7). New York, NY, USA: ACM.

Clegg, D., & Barker, R. (1994). *Case Method Fast-Track: A Rad Approach.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Clements, P., & Northrop, L. (2001). *Software Product Lines: Patterns and Practice.* Reading, MA: Addison Wesley.

Cline, M., & Girou, M. (2000). Enduring business themes. *Commun. ACM*, *43*(5), 101–106.

CMMI Product Team. (2002). *Capability Maturity Model Integration* (Tech. Rep. No. CMU/SEI-2002-TR-012). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University.

Cooper, K. M. L. (2006). Can Agility be Introduced into Requirements Engineering for COTS Component Based Development? In *Iwspm '06: Proceedings of the international workshop on software product management* (pp. 35–37). Washington, DC, USA: IEEE Computer Society.

Cossentino, M., Gaglio, S., Henderson-Sellers, B., & Seidita, V. (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).*

Damian, D. (2007). Stakeholders in global requirements engineering: Lessons learned from practice. *IEEE software*, *24*(2), 21–27.

Deneckere, R., Iacovelli, A., Kornyshova, E., & Souveyet, C. (2008). From Method Fragments to Method Services. In *Proceedings of emmsad'08.*

Denning, P. J. (1997). A New Social Contract for Research. *Communications of the ACM*, *40*(2), 9–30.

Dzamashvili-Fogelström, N., & Gorschek, T. (2007). Test-case Driven versus Checklist-based Inspections of Software Requirements - An Experimental Evaluation. In *Wer* (pp. 116–126).

Ebert, C. (2006). Understanding the Product Life Cycle: Four Key Requirements Engineering Techniques. *IEEE Software*, *23*(3), 19–25.

Ebert, C. (2007). The Impacts of Software Product Management. *Journal of Systems and Software*, *6*(80), 850–861.

Ebert, C., & Smouts, M. (2003). Tricks and Traps of Initiating a Product Line Concept in Existing Products. In *Proceedings of the international conference on software engineering.*

Eisenhardt, K. M. (1989). Building Theories from Case Study Research. *The Academy of Management Review*, *14*(4), 532–550.

Fehlmann, T. M. (2008). New Lanchester Theory for Requirements Prioritization. In *Iwspm '08: Proceedings of the 2008 second international workshop on software product management* (pp. 35–40). Washington, DC, USA: IEEE Computer Society.

Fricker, S. (2005). Methodological Support for Engineering Strategic Requirements for Commercial Products using Goals (Doctoral Symposium). In *Proceedings of the ieee international conference on requirements engineering (re'05).* Paris, France.

Fricker, S. (2007). Explaining Stakeholder Negotiation Using Social Goal Networks. In *Proceedings of the international conference on requirements engineering* (pp. 387–388).

Fricker, S., Gorschek, T., & Glinz, M. (2008). Goal-Oriented Requirements Communication in New Product Development. In *Iwspm '08: Proceedings of the 2008 second international workshop on software product management* (pp. 27–34). Washington, DC, USA: IEEE Computer Society.

Fricker, S., & Grunbacher, P. (2008). Negotiation Constellations - Method Selection Framework for Requirements Negotiation. In *Proceedings of refsq08* (pp. 37–51).

Fricker, S., & Stoiber, R. (2008). Relating Product Line Context to Requirements Engineering Processes Using Design Rationale. In *Produktlinien im kontext: Technologie, prozesse, business und organisation (pik2008)* (pp. 240–251).

Gonzalez, R., Gasco, J., & Llopis, J. (2006). Information systems outsourcing: A literature analysis. *Information & Management*, *43*(7), 821–834.

Gorchel, L. (2000). *The Product Managers Handbook: The Complete Product Management Resource (2nd edition)*. NTC Business Books.

Gorschek, T., & Davis, A. M. (2008). Requirements engineering: In search of the dependent variables. *Information {&} Software Technology*, *50*(1-2), 67–75.

Gorschek, T., Svahnberg, M., Borg, A., Loconsole, A., Börstler, J., Sandahl, K., et al. (2007). A controlled empirical evaluation of a requirements abstraction model. *Information {&} Software Technology*, *49*(7), 790–805.

Graham, I., Henderson-Sellers, B., & Younessi, H. (1997). *The OPEN process specification*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co.

Grieves, M. (2005). *Product Lifecycle Management: Driving the Next Generation of Lean Thinking*. McGraw-Hill.

Gurp, J. van, Bosch, J., & Svahnberg, M. (2001). On the Notion of Variability in Software Product Lines. In *Proceedings of the working ieee/ifip conference on software architecture* (p. 45). Los Alamitos, CA, USA: IEEE Computer Society.

Guzélian, G., & Cauvet, C. (2007). SO2M: Towards a Service-Oriented Approach for Method Engineering. In *Proceedings of the international conference ike'07*.

Harmsen, F., Brinkkemper, S., & 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* (pp. 169–194). New York, NY, USA: Elsevier Science Inc.

Henderson-Sellers, B. (2002). Process Metamodelling and Process Construction: Examples Using the OPEN Process Framework (OPF). *Annals of Software Engineering*, *14*, 341–362.

Henderson-Sellers, B., Simons, A., & Younessi, H. (1998). *The OPEN toolbox of techniques*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co.

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design Science in Information Systems Research. *Management Information Systems Quarterly*, *28*(1).

Hippel, E. von. (1986). Lead Users: A Source of Novel Product Concepts. *Management Science*, *32*(7), 791–805.

Hoek, A. van der, Hall, R. S., Heimbigner, D., & Wolf, A. L. (1997). Software release management. *SIGSOFT Softw. Eng. Notes*, *22*(6), 159–175.

Jansen, S. (2007). *Customer Configuration Updating in a Software Supply Network*. Unpublished doctoral dissertation, Utrecht University.

Jansen, S., & Brinkkemper, S. (2008). Information Systems Research Methods, Epistemology, and Applications. In A. Cater-Steel & L. Al-Hakim (Eds.), (pp. 120–139). Idea Group Inc.

Jantunen, S., & Smolander, K. (2006). Challenges of Knowledge and Collaboration in Roadmapping. In *Iwspm '06: Proceedings of the international workshop on software product management* (pp. 19–26). Washington, DC, USA: IEEE Computer Society.

Jarke, M., Rolland, C., Sutcliffe, A., & Dömges, R. (1999). *The nature of requirements Engineering.* Shaker.

Kappel, T. (2001). Perspectives on Roadmaps: How Organisations Talk about the Future. *IEEE Engineering Management Review*, *29*(3), 36–48.

Karlsson, F. (2002). Bridging the gap between method for method configuration and situational method engineering. *Promote IT, Skvde, Sweden*.

Karlsson, F., & Å gerfalk, P. J. (2004). Method configuration: adapting to situational characteristics while creating reusable assets. *Information and Software Technology*, *46*(9), 619–633.

Karlsson, F., & Wistrand, K. (2006). Combining method engineering with activity theory: theoretical grounding of the method component concept. *European Journal of Information Systems*, *15*(1), 82–90.

Khurum, M., Aslam, K., & Gorschek, T. (2007). A Method for Early Requirements Triage and Selection Utilizing Product Strategies. In *Proceedings of the 14th asia-pacific software engineering conference* (pp. 97–104).

Khurum, M., Gorschek, T., & Pettersson, K. (2008). Systematic Review of Solutions Proposed for Product Line Economics. In *Proceedings of mespul09* (pp. 277–284).

Kiritsis, D., Bufardi, A., & Xirouchakis, P. (2003). Research issues on product lifecycle management and information tracking using smart embedded systems. *Advanced Engineering Informatics*, *17*(3-4), 189–202.

Kumar, K., & Welke, R. J. (1992). Methodology Engineering R: a proposal for situation-specific methodology construction. In *Challenges and strategies for research in systems development* (p. 269).

Lawrence, B., Wiegers, K., & Ebert, C. (2001). The Top Risks of Requirements Engineering. *IEEE Software*, *18*(6), 62–63.

Lee, J., Kang, K. C., & Kim, S. (2004). A Feature-Based Approach to Product Line Production Planning. In R. L. Nord (Ed.), *Proceedings of the software product lines conference* (pp. 137–140). Berlin / Heidelberg: Springer.

Lee, K., Kang, K. C., & Lee, J. (2002). Concepts and Guidelines of Feature Modeling for Product Line Software Engineering. In C. Gacek (Ed.), *Proceedings of the internation conference on software reuse* (pp. 62–77). Berlin / Heidelberg: Springer.

March, S. T., & Smith, G. F. (1995). Design and natural science research on information technology (Invited Paper). *Decision Support Systems (Special issue on WITS '92)*, *15*(4), 251–266.

Moon, M., & Yeom, K. (2004). An Approach to Develop Requirement as a Core Asset in Product Line. In *Lecture notes in computer science, no. 3107* (pp. 23–34).

Nguyen, P. T., Babar, M. A., & Verner, J. M. (2006). Critical factors in establishing and maintaining trust in software outsourcing relationships. In *Icse '06: Proceedings of the 28th international conference on software engineering* (pp. 624–627). New York, NY, USA: ACM.

Oza, N. V., Hall, T., Rainer, A., & Grey, S. (2006). Trust in software outsourcing relationships: An empirical investigation of Indian software companies. *Information and Software Technology*, *48*(5), 345–354.

Pastor, O., Fons, J., & Pelechano, V. (2003). OOWS: A method to develop web applications from web-oriented conceptual models. In *Proceedings of iwwost'03.* Oviedo: Luis Olsina, Oscar Pastor, Gustavo Rossi, Daniel Schwabe.

Pastor, O., Insfrán, E., Merseguer, J., Romero, J., & Pelechano, V. (1997). OO-METHOD: An OO Software Production Environment Combining Conventional and Formal Methods. In *Proceedings of caise'97* (Vol. 1250, pp. 145–159). Barcelona: Springer-Verlag.

Paul, M. C., Curtis, B., Chrissis, M. B., & Weber, C. V. (1993). *Capability Maturity Model for Software* (Tech. Rep. Nos. SEI/CMU-93-TR-24, ADA263403). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University.

Phaal, R., Farrukh, C. J. P., & Probert, D. R. (2000). Fast-start technology roadmapping. In *Management of technology, the key to prosperity in the third millennium. proceedings of the iamot 9th international conference* (pp. 275–284). Pergamon.

Pichler, M., Rumetshofer, H., & Wahler, W. (2006). Agile Requirements Engineering for a Social Insurance for Occupational Risks Organization: A Case Study. In *14th ieee international requirements engineering conference.*

Potts, C. (1995). Invented Requirements and Imagined Customers: Requirements Engineering for Off-the-Shelf Software. In *Proceedings of the second ieee international symposium on requirements engineering.*

Racheva, Z., Daneva, M., & Buglione, L. (2008). Supporting the Dynamic Reprioritization of Requirements in Agile Development of Software Products. In *Iwspm '08: Proceedings of the 2008 second international workshop on software product management* (pp. 49–58). Washington, DC, USA: IEEE Computer Society.

Ralyté, J., Deneckère, R., & Roll, C. (2003). Towards a generic model for situational method engineering. In *Proceedings of the international conference on advanced information systems engineering 2003, lncs 2681* (pp. 95–110). Springer-Verlag.

Ralyté, J., & Rolland, C. (2001). An Approach for Method Reengineering. In *Proceedings of the 20th international conference on conceptual modeling.*

Regnell, B., Svensson, R. B., & Olsson, T. (2008). Supporting Roadmapping of Quality Requirements. In *Ieee software* (Vol. 25, pp. 42–47). Los Alamitos, CA, USA: IEEE Computer Society.

Regnell, B., Svensson, R. B., & Wnuk, K. (2008). Can We Beat the Complexity of Very Large-Scale Requirements Engineering? In *Refsq '08: Proceedings of the 14th international conference on requirements engineering* (pp. 123–128). Berlin, Heidelberg: Springer-Verlag.

Rolland, C. (2007). Method Engineering : Achievements,Trends & Challenges. In *Keynote presentations of me'07.*

Rolland, C., Prakash, N., & Benjamen, A. (1999). A multi-model view of process modelling. *Requirements Engineering*, *4*(4), 169–187.

Rossi, M., Ramesh, B., Lyytinen, K., & Tolvanen, J.-P. (2004). Managing Evolutionary Method Engineering by Method Rationale. *Journal of the Association for Information Systems*, *5*(9), 356–391.

Ruhe, G., & Saliu, M. O. (2005). The Art and Science of Software Release Planning. *IEEE Software*, *22*(6), 47–53.

Sääksvuori, A., & Immonen, A. (2008). *Product Lifecycle Management* (3rd ed.). Springer.

Saaty, T. L. (1980). *The Analytic Hierarchy Process.* McGraw-Hill.

Saeki, M. (2003). Embedding metrics into information systems development methods: An application of method engineering technique. In *Advanced information systems engineering* (p. 1031).

Siakas, K., Maoutsidis, D., & Siakas, E. (2006). Trust facilitating good software outsourcing relationships. *Software Process Improvement*, *4257*, 171–182.

Simon, H. A. (1981). *The Sciences of the Artificial* (2nd ed.). Cambridge, MA: MIT Press.

Slooten, K., & Brinkkemper, S. (1993). A method engineering approach to information systems development. *Information System Development Process*, *1993*, 167–186.

Slooten, K., & Hodes, B. (1996). Characterizing IS development projects. In *Method engineering: Principles of method construction and tool support, proceedings of the ifip tc8, wg8. 7/8.2 working conference on method engineering.*

Stapleton, J. (1999). DSDM: Dynamic Systems Development Method. In *Tools '99: Proceedings of the technology of object-oriented languages and systems* (p. 406). Washington, DC, USA: IEEE Computer Society.

Stark, J. (2005). *Product lifecycle management: 21st century paradigm for product realisation.* Birkhäuser.

Sudarsan, R., Fenves, S. J., Sriram, R. D., & Wang, F. (2005). A product information modeling framework for product lifecycle management. *Computer-Aided Design*, *37*(13), 1399–1411.

Svahnberg, M., Gorschek, T., Eriksson, M., Borg, A., Sandahl, K., Börster, J., et al. (2008). Perspectives on Requirements Understandability – For Whom Does the Teacher's Bell Toll? In *Requirements engineering education and training* (Vol. 0, pp. 22–29). Los Alamitos, CA, USA: IEEE Computer Society.

Svahnberg, M., & Karasira, A. (2009). A Study on the Importance of Order in Requirements Prioritisation. In *Proceedings of the 3rd international workshop on software product management.*

Svensson, R. B., Olsson, T., & Regnell, B. (2008). Introducing Support for Release Planning of Quality Requirements — An Industrial Evaluation of the QUPER Model. In *Iwspm '08: Proceedings of the 2008 second international workshop on software product management* (pp. 18–26). Washington, DC, USA: IEEE Computer Society.

Tomer, A., Goldin, L., Kuflik, T., Kimchi, E., & Schach, S. R. (2004). Evaluating Software Reuse Alternatives: A Model and Its Application to an Industrial Case Study. *IEEE Transactions on Software Engineering*, *30*, 601–612.

Tsichritzis, D. (1997). Beyond Calculation: The Next Fifty Years of Computing. In (pp. 259–265). Copernicus.

Vähäniitty, J., Lassenius, C., & Rautiainen, K. (2002). An Approach to Product Roadmapping in Small Software Product Businesses. In *Conference notes of quality connection - 7th european conference on software quality (ecsq2002).*

van De Weerd, I., Brinkkemper, S., Souer, J., & Versendaal, J. (2006). A situational implementation method for web-based content management system-applications: method engineering and validation in practice. *Software Process: Improvement and Practice*, *11*(5), 521–538.

van De Weerd, I., Versendaal, J., & Brinkkemper, S. (2006). A Product Software Knowledge Infrastructure for Situational Capability Maturation: Vision and Case Studies in Product Management. In *Proceedings of the 12th working conference on requirements engineering: Foundation for software quality (refsq'06).*

Vlaanderen, K., Brinkkemper, S., Jansen, S., & Jaspers, E. (2009). The Agile Requirements Refinery: Applying SCRUM Principles to Software Product Management. In *Proceedings of the 3rd international workshop on software product management.*

Weerd, I. van de. (2005). *A Design Method for CMS-based Web Applications* (Tech. Rep. No. UU-CS-2005-043). Institute of Computing and Information Sciences, Utrecht University.

Weerd, I. van de, Bekkers, W., & Brinkkemper, S. (2009). *Developing a Maturity Matrix for Software Product Management* (Tech. Rep. No. UU-CS-2009-015). Utrecht University.

Weerd, I. van de, & Brinkkemper, S. (2008). Handbook of Research on Modern Systems Analysis and Design Technologies and Applications. In M. R. Syed & S. N. Syed (Eds.), (pp. 38–58). Hershey: Idea Group Publishing.

Weerd, I. van de, Brinkkemper, S., Nieuwenhuis, R., Versendaal, J., & Bijlsma, L. (2006a). On the Creation of a Reference Framework for Software Product Management: Validation and Tool Support. In *Iwspm '06: Proceedings of the international workshop on software product management* (pp. 3–12). Washington, DC, USA: IEEE Computer Society.

Weerd, I. van de, Brinkkemper, S., Nieuwenhuis, R., Versendaal, J., & Bijlsma, L. (2006b). Towards a Reference Framework for Software Product Management. In *14th ieee international requirements engineering conference.*

Weerd, I. van de, Brinkkemper, S., & Versendaal, J. (2007). Concepts for Incremental Method Evolution: Empirical Exploration and Validation in Requirements Management. In *Proceedings of the 19th international conference on advanced information systems engineering.*

Wiegers, K. (2009). First things first. *Software Development*, *7*(9), 48–53.

Wistrand, K., & Karlsson, F. (2004). Method Components – Rationale Revealed. In *Proceedings of the international conference on advanced information systems engineering.*

Wnuk, K., Regnell, B., & Schrewelius, C. (2009). Architecting and Coordinating Thousands of Requirements – An Industrial Case Study. In *Proceedings of refsq'09: Requirements engineering foundation for software quality* (pp. 118–123).

Xu, L., & Brinkkemper, S. (2005). Concepts of Product Software: Paving the Road for Urgently Needed Research. In *Proceedings of the first international workshop on philosophical foundations of information systems engineering.*

Yamazaki, S. (2009). Software Product Line Engineering with Personas. In *Proceedings of the 3rd international workshop on software product management.*

Yin, R. K. (2003). *Case Study Research - Design and Methods.* SAGE Publications.

Appendix A

Activity Tables PSKI

Activity	Sub-Activity	Description
Decide on focus		Before the process of method improvement can start, the company needs to decide on the area(s) that will be focussed on. It can decide to analyze the entire SPM process, or only a part of it.
Generate questionnaire		Based on the areas decided on before, a list of relevant questions can be generated, to be asked during the next activities.
Model process		If the company decides to provide full process-information, the process and its deliverables need to be modelled in the form of a process-deliverable diagram.
Perform interview situational factors		If a company decides it needs help during the method improvement process, an expert could perform parts of the process. For gathering the initial information, an interview is an effective instrument.
Perform interview spm maturity		The second interview is optional. It can be skipped if the company decides to provide full process information in the form of a PDD.
Perform questionnaire situational factors		In case the company wants to perform self-assessment, one or two questionnaires will be need to be filled in. The first one of these regards situational factors, and should not be optional.
Perform questionnaire spm maturity		The second questionnaire is optional. It can be skipped if the company decides to provide full process information in the form of a PDD. However, in the case of self-assessment, creating a PDD might be too complex, in which case it is easier to fill in a questionnaire regarding spm maturity.

Table A.1: Activity table for the phase 'Analysis of current situation'

Activity	Sub-Activity	Description
Determine delta		Once the two maturity profiles have been created, they can be combined into a new matrix, showing the differences between them and thus the areas that can be improved.
Determine optimal maturity		Based on the situational factors and the related situational factor effects, the system can calculate what the optimal maturity is for the selected areas.
Determine current maturity		Based on the provided maturity information, a company's maturity profile can be calculated.

Table A.2: Activity table for the phase 'Analysis of need'

Activity	Sub-Activity	Description
Combine method fragments		Combine the selected METHOD FRAGMENTS into SOLUTIONS that implement many to all of the required capabilities
Select method fragments		Select all METHOD FRAGMENTS that implement one or more of the required capabilities
Select solution		Based on all the information received, the user should the SOLUTION that best suits the need of his company. This SOLUTION will then be used during the rest of the process.
Verify compatibility		Determine whether the selected METHOD FRAGMENTS can be combined. This includes checking for compatible deliverables, non-competing processes, etc.
Determine effectiveness		Calculate the percentage of MISSING CAPABILITIES that is implemented by this SOLUTION.
Determine impact		Calculate the amount of changes that needs to be made to the current process when using this SOLUTION
Present solutions		Display (a subset of) the SOLUTIONS, so that the user can browse and evaluate them. This can be appended with recommendations; SOLUTIONS or METHOD FRAGMENTS that have been rated high by company's with similar SITUATIONAL PROFILES.
Present reviews		Users should have direct access to the REVIEWS associated to METHOD FRAGMENTS, to be able to form a clear image of each FRAGMENT's positive and negative aspects.

Table A.3: Activity table for the phase 'Selection of process alternatives'

Activity	Sub-Activity	Description
Create roadmap of improvement steps		The IMPLEMENTATION STEPS need to be layed out on a ROADMAP, describing the order of the STEPS, how much time is needed for each (approximately), and the maturity level after each STEP.
Implement step	Integrate method fragment	If the user has provided the system with full process information, than it should be possible to integrate each METHOD FRAGMENT into the current PROCESS. As all the METHOD FRAGMENTS in a SOLUTION come from different sources, it is probably best to do this integration step-by-step. The result is a METHOD INCREMENT, showing the differences between the old process and the new, proposed process.
	Present detailed step description	Once all the METHOD FRAGMENTS in a STEP have been integrated, the resulting process can be shown to the user. It can then be evaluated, and perhaps changed when needed.
	Perform step	Once the user accepts the proposed METHOD INCREMENT, it needs to be implemented in the company. This is a complex task, out of the scope of the PSKI.
Perform step		Based on the received information, the user can perform the STEP. This will be harder, however, than when full process information is available, as the user now needs to manually adapt the current SPM PROCESS at the company.
Present basic step description		If the user has only given maturity information, than no method integration can be performed. Instead, the METHOD FRAGMENTS that are related to a STEP can be shown, to provide the user with the necessary information.
Present roadmap		The created ROADMAP is presented to the user, who can evaluate it, and change it when needed.
Split selected solution into steps		The proposed SOLUTION should be split into smaller IMPLEMENTATION STEPS, as evolutionary improvements have more chance of success than revolutionary change.

Table A.4: Activity table for the phase 'Embedding of process advice'

Activity	Sub-Activity	Description
Generate templates		For DOCUMENTS that are not available to the PSKI, TEMPLATES can be generated based on the proposed METHOD FRAGMENTS. These can then be used by the company to create new DOCUMENTS, or to update old ones.
Update documents		If the DOCUMENTS that are used during the SPM PROCESS are accessible to the PSKI, than their structure can be changed according to the proposed changes to the process. In this way, the old data is maintained, while the DOCUMENTS are ready to be used in the new PROCESS.

Table A.5: Activity table for the sub-phase 'Template creation'

Activity	Sub-Activity	Description
Alter method fragment		Based on the submitted CORRECTION, the associated METHOD FRAGMENT is altered. A history of these changes should be kept.
Rate method fragment		Each METHOD FRAGMENT can be given a rating, based on the experiences of the USER with this FRAGMENT.
Review correction		The CORRECTION is reviewed by an expert. If too many CORRECTIONS are being made, then the review process can be limited to those CORRECTIONS that have been submitted by multiple users.
Review submission		REVIEWS should always be check by someone, to make sure that no unwanted text is published. However, no censorship should be applied.
Select type of input		Users should always be able to give feedback regarding METHOD FRAGMENTS or SOLUTIONS that they have implemented (or tried to implement). They have several options for giving this feedback; corrections, reviews and ratings.
Submit correction		If the expert deems the correction of good quality, than he submits it as final.
Suggest correction		The user first suggests a correction to a METHOD FRAGMENT.
Write review		The user writes some text regarding the complexity, applicability, effictiveness, etc. of the METHOD FRAGMENT of SOLUTIO that he implemented (or tried to implement).

Table A.6: Activity table for the phase 'Knowledge base improvement'

Appendix B

Concept Tables PSKI

Concept	Description
CAPABILITY	One of the activities identified by Weerd et al. (2009). **Properties:** • process • maturity level
IMPLEMENTED CAPABILITY	One of the activities identified by Weerd et al. (2009) that has been implemented at the company.
PDD	For the modelling of method fragments, an approach is used based on the proposal of Saeki (2003). The technique uses a combination of a UML activity diagram and a UML class diagram (Weerd & Brinkkemper, 2008). **Properties:** • processes • deliverables • capabilities
PROCESS AREA	Group of activities in the REFERENCE FRAMEWORK FOR SPM that belong to one level in the hierarchy of SPM artifacts (Weerd et al., 2006a).
QUESTION	An expression of inquiry.
QUESTIONNAIRE	A form containing a set of QUESTIONS.
REFERENCE FRAMEWORK FOR SPM	Framework in which the key process areas, stakeholders and their relations are modeled (Weerd et al., 2006a).
SELECTED AREA	A focus area of the reference framework for SPM that has been deemed important for the company.
SITUATIONAL FACTOR	Any factor relevant for product development and product services (Weerd et al., 2006a). **Properties:** • description • values

Table B.1: Concept table for the phase 'Analysis of current situation'

Concept	Description
AREAS OF IMPROVEMENT MATRIX	A CAPABILITY MATRIX indicating the status of each CAPABILITY, the different statuses being: 'implemented', 'missing', 'N/A', and 'extra' (Bekkers et al., 2010). **Properties:** • current maturity levels • processes • optimal maturity levels • delta
CAPABILITY	One of the activities identified by Weerd et al. (2009). **Properties:** • process • maturity level
CAPABILITY MATRIX	A matrix providing an overview of all the CAPABILITIES that need to be implemented to reach a full-grown maturity. The matrix consists of columns and rows, which represent the two dimensions of the maturity model (Bekkers et al., 2010).
CURRENT CAPABILITY PROFILE	A CAPABILITY MATRIX based on the IMPLEMENTED CAPABILITIES (Bekkers et al., 2010). **Properties:** • processes • current maturity levels
IMPLEMENTED CAPABILITY	One of the activities identified by Weerd et al. (2009) that has been implemented at the company.
OPTIMAL CAPABILITY PROFILE	A custom CAPABILITY MATRIX tailored to the situational context of the organization (Bekkers et al., 2010). **Properties:** • processes • optimal maturity levels
SITUATIONAL FACTOR	Any factor relevant for product development and product services (Weerd et al., 2006a). **Properties:** • description • values
SITUATIONAL FACTOR EFFECT	A method to model product manager's knowledge. It reflects what should be done under certain circumstances (a specific SF value, or range of values) Bekkers et al. (2010). **Properties:** • condition • effect

Table B.2: Concept table for the phase 'Analysis of need'

Concept	Description
AREAS OF IMPROVEMENT MATRIX	A CAPABILITY MATRIX indicating the status of each CAPABILITY, the different statuses being: 'implemented', 'missing', 'N/A', and 'extra' (Bekkers et al., 2010). **Properties:** • current maturity levels • processes • optimal maturity levels • delta
DELTA	Set of CAPABILITIES that have been marked as 'missing' in the AREAS OF IMPROVEMENT MATRIX.
METHOD BASE	The central database that stores all the information required for the PSKI. It stores METHOD FRAGMENTS, ASSEMBLY RULES, SITUATIONAL FACTORS, CAPABILITY MATURITIES, SITUATIONAL FACTORS EFFECTS and USER FEEDBACK.
METHOD FRAGMENT	A description of an IS engineering method, or any coherent part thereof (Harmsen et al., 1994). **Properties:** • rating • situational context • capabilities • pdd
MISSING CAPABILITY	One of the activities identified by Weerd et al. (2009) that has been identified as 'missing' by the AREAS OF IMPROVEMENT MATRIX. **Properties:** • process • maturity level
REVIEW	Feedback from a user regarding his experience with the quality, applicability, complexity, etc. of a METHOD FRAGMENT. The SITUATIONAL CONTEXT is also provided, to indicate the situation in which the experience was obtained. **Properties:** • situational context • text
SELECTED SOLUTION	SOLUTION that has been chosen by the user as most suited to his situation.

Table B.3: Concept table for the phase 'Selection of process alternatives'

Concept	Description
ASSEMBLY RULE	These govern both the combintation of METHOD FRAGMENTS from different sources, as well as the internal structure of methods. They are derived from experiences and existing methods (van De Weerd, Versendaal, & Brinkkemper, 2006). **Properties:** • condition • effect
DELTA	Set of CAPABILITIES that have been marked as 'missing' in the AREAS OF IMPROVEMENT MATRIX.
IMPROVEMENT ROADMAP	SOLUTIONS should be split into IMPROVEMENT STEPS, which are layed out on an IMPROVEMENT ROADMAP. This helps managers keep the change process managable.
IMPROVEMENT STEP	By dividing SOLUTIONS into STEPS, implementation becomes managable. Each STEP should serve a goal, to make implementation more successfull. Also, a complexity rating should be calculated, so as to give managers an indication of the amount of effort required. **Properties:** • complexity • goal • description
METHOD BASE	The central database that stores all the information required for the PSKI. It stores METHOD FRAGMENTS, ASSEMBLY RULES, SITUATIONAL FACTORS, CAPABILITY MATURITIES, SITUATIONAL FACTORS EFFECTS and USER FEEDBACK.
METHOD FRAGMENT	A description of an IS engineering method, or any coherent part thereof (Harmsen et al., 1994). **Properties:** • rating • situational context • capabilities • pdd
METHOD INCREMENT	A collection of METHOD FRAGMENTS that have been introduced in the method during the method adaptations between t_i and t_{i-1} (Weerd et al., 2007).
MISSING CAPABILITY	One of the activities identified by Weerd et al. (2009) that has been identified as 'missing' by the AREAS OF IMPROVEMENT MATRIX. **Properties:** • process • maturity level
PDD	For the modelling of method fragments, an approach is used based on the proposal of Saeki (2003). The technique uses a combination of a UML activity diagram and a UML class diagram (Weerd & Brinkkemper, 2008). **Properties:** • processes • deliverables • capabilities
SELECTED SOLUTION	SOLUTION that has been chosen by the user as most suited to his situation.
SPM PROCESS	The collection of activities that are performed within a company related to software product management.

Table B.4: Concept table for the phase 'Embedding of process advice'

Concept	Description
DOCUMENT	Many DOCUMENTS are used during the SPM PROCESS, including roadmaps, requirements definition, conceptual solutions, etc. **Properties:** • columns • sections • data
IMPROVEMENT STEP	By dividing SOLUTIONS into STEPS, implementation becomes managable. Each STEP should serve a goal, to make implementation more successfull. Also, a complexity rating should be calculated, so as to give managers an indication of the amount of effort required. **Properties:** • complexity • goal • description
METHOD FRAGMENT	A description of an IS engineering method, or any coherent part thereof (Harmsen et al., 1994). **Properties:** • rating • situational context • capabilities • pdd
TEMPLATE	DOCUMENTS can be abstracted into TEMPLATES, describing the structure of a DOCUMENT, i.e. which columns or sections it contains. **Properties:** • columns • sections

Table B.5: Concept table for the sub-phase 'Template creation'

Concept	Description
CORRECTION	Any type of suggestion for change, issued by users. This includes correction of the data-model, corrections of the process, updates of the descriptions, etc.
METHOD FRAGMENT	A description of an IS engineering method, or any coherent part thereof (Harmsen et al., 1994). **Properties:** • rating • situational context • capabilities • pdd
REVIEW	Feedback from a user regarding his experience with the quality, applicability, complexity, etc. of a METHOD FRAGMENT. The SITUATIONAL CONTEXT is also provided, to indicate the situation in which the experience was obtained. **Properties:** • situational context • text
SOLUTION	A combination of METHOD FRAGMENTS that combinedly implement (a subset of) the missing CAPABILITIES as identified by the AREAS OF IMPROVEMENT MATRIX. Two scores are calculated, effectiveness and impact, to indicate the quality of a solution. **Properties:** • effectiveness • impact

Table B.6: Concept table for the phase 'Knowledge base improvement'