Advancing in Software Product Management: An Incremental Method Engineering Approach

Inge van de Weerd

SIKS Dissertation Series No. 2009-34

The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

ISBN: 978-90-39351369

Advancing in Software Product Management: An Incremental Method Engineering Approach

PROEFSCHRIFT

Vooruitgang in Software Product Management:
Een Incrementele Method Engineering Aanpak

(met een samenvatting in het Nederlands)

ter verkrijging van de graad van doctor
aan de Universiteit Utrecht op gezag van
de rector magnificus, prof. dr. H. Stoof
ingevolge het besluit van het college
voor promoties in het openbaar te verdedigen op
9 september 2009 des middags te 4.15 uur
door

Gerdina Carolina van de Weerd

geboren op 3 januari 1981,
te Rhenen

Promotor: prof. dr. S. Brinkkemper

Co-promotor: dr. ir. Johan Versendaal

Preface

September 1, 2005 was the first day of my four-year PhD process. I spent that day at the Paris 1 Sorbonne University, where I visited my first conference. I remember the statues and paintings of Descartes, Bossuet, Hugo and Pasteur looking down at me in the big halls. One of the keynote speakers repeated Isaac Newton's words: "If I have seen further it is by standing on ye shoulders of giants". That's exactly how I felt that day. Presenting my first paper and then finding out that several of the authors that I cited were actually in the audience was a strange sensation.

Four years later, I can add an extra perspective: that of the one whose research is used by others. I am extremely grateful for that. On the one hand that my work is mostly well received in the academic world, but also that the industry shows interest in our theories and the educational program that we set up. That my research results are actually used is one of the finest compliments.

I could not have achieved these things without Sjaak Brinkkemper and Johan Versendaal. Starting in Paris, where you overthrew me with millions of ideas, to helping me in writing my first articles, and stepping back when it was time to stand on my own feet. Thank you for your support.

I would like to thank my colleagues, Eva, Henk, Slinger, Rogier, Marijn, Remko, Marco, Ronald, Jurriaan and others. A special thanks to Sandra who made the final busy days of preparing my manuscript a bit easier. I must also thank Willem and Lútzen who both carried out wonderful research projects for their Master theses and with whom I co-authored several papers. Also thanks to the other MBI students I supervised for providing me with new ideas and fresh insights. Furthermore, I would like to thank the reading committee, Frank Harmsen, Bjorn Regnell, Motoshi Saeki, Hans van Vliet and Roel Wieringa, for taking the time to read and judge my dissertation.

Finally, I would like to thank my family and friends for their advice, belief in me and interest in my work, but also for the great holidays, dinners, pool parties, and barbecues. Erik, thank you for supporting me, for forgiving me the boring evenings I had to work, and for reviewing almost my entire dissertation. In two years it will be my turn.

Inge van de Weerd, July 2009

Contents

CHAPTER 1

Introduction

1.1 Motivation

The software business has made a shift from developing software for one customer to developing standard software for an entire market. This shift from customized software to product software brings new challenges, especially concerning the processes for managing the product; the domain of Software Product Management (SPM). In the following paragraphs, three typical challenges in SPM are presented.

Internet browsers use different protocols to update their products. Duebendorfer and Frei (2009) found out that the more effort is needed to install a new version of the browser, the less willing users are to actually update. For example, the update process of Opera requires a manual re-install of the product. Consequently, only 24% of the users have installed a new version after 21 days. Apple gives users the choice to update automatically, manually or never. Their score after three weeks is 53%. Out-to-date software may cause serious problems, such as less secure and less stable products. In addition, it takes the software developers a lot of time, as they have to take old browser versions into account when creating plug-ins or add-ons; get more bug reports since multiple versions are in use; and have to keep on testing the old browser versions when new platforms, add-ons or plug-ins are developed. By implementing a more effective update protocol, these problems can be prevented. For example, Mozilla Firefox, with its one-click-update process, has a score of 87%. Google Chrome, who uses an automatic, silent updating protocol, performed best with 97%.

With the micro-blogging service Twitter, users can send small messages (“tweets”) to their social network. Users can subscribe to other users in order to follow their updates and react to each other’s tweets. In May 2009, the Twitter founders decided (without consulting the users) to remove part of the functionality that made it possible to follow conversations of others, since, they argued, it was an “undesirable and confusing option”. Within hours, many Twitter users voiced their discontent by using the "#fixreplies"-tag in their messages. The next day, Twitter reinstated the reply feature (cf. Stone, 2009).

In July 2001, Microsoft announced the development of the new Windows version: Microsoft Longhorn. Within a timeframe of three years, the Longhorn version would be released (Schultz, 2006). However, it turned out that Microsoft could not meet its goals and Longhorn (or Vista as it is called now) was released in November 2006; a delay of 2 years, causing disappointed customers and a suspension of the expected revenues.

The problems sketched above are anecdotes from the SPM world. The browser updating problem can be viewed from two perspectives. First, it entails the launching process of the product and handling old product versions. The other perspective involves the gathering of market requirements. Apparently, updating instruments should be as easy as possible for the user, in order to be used regularly. Product managers can identify this as a market requirement that, if implemented, will increase a product's security and stability. The second example concerns the balance between listening to the voice of the customer, the so-called market requirements, and planning your own product vision. Finally, the Vista delay touches the difficulty of realistic release planning; an important domain since a accurate planning of upcoming releases is crucial to a company's business performance.

Hardly any figures exist on the success of product software companies. What we do know is that a good SPM practice pays off. Ebert (2007) describes an empirical study with data from 178 industry projects that shows that time to market, schedule adherence and handover quality all improve with the strengthening of a coherent product management role. However, not many IT-professionals know how to implement SPM practices in their organization. In a research performed among 35 product managers in the Netherlands, it appeared that many companies do not have the proper SPM processes (such as prioritizing requirements or defining a product roadmap) in place (van de Weerd, Bekkers & Brinkkemper, forthcoming). One of the reasons of this low maturity in SPM practices is that hardly any education exists in this domain. Some commercial courses are offered in the US and Europe. However, software product management is not taught in colleges and universities. Many product managers acquire their job accidentally, as in the case of the product manager in the example above. As a consequence, they have to learn the practice of SPM on the job. Since no solid body of knowledge in the SPM domain exists, this can be a difficult task.

An approach to address the lack of SPM knowledge among product managers is to give them access to SPM methods and guide them in implementing them in their company. Immediately some other problems come to mind. For example, product software companies can be characterized by

various situational factors; they operate in diverse sectors, have varying sizes and use a range of development methods. Subsequently, companies need methods that are tuned to these situational facors. Karlsson, Dahlsted and Natt och Dag (2002) found that it is very difficult for companies to find a balance between the "elaborate and elementary development processes". They argue that the degree of elaborations is dependent on a company's maturity. For example, a company with five employees does not need an elaborate release planning method, whereas a large company, such as Microsoft, needs to have a very elaborate workflow process in place within the software product management domain. Each company operates in its own context that can be described by multiple situational factors. These situational factors have a great influence on the decision on implementing simple or elaborate SPM processes.

In this research, a knowledge infrastructure is proposed that provides methodical support to product software companies. The aim of this knowledge infrastructure is pto assess and thereby analyze a company's current situation and maturity level. Then, by using incremental method engineering and meta-modeling principles, previously stored method fragments can be selected and assembled into a process advice. By implementing this process advice in the existing processes, the overall maturity of the SPM practice increases.

1.2 Scientific relevance

Software engineering is a domain that has been extensively researched. The entire process, from requirements elicitation until software maintenance, is comprehensively described. Developers use standard works from for example Sommerville (2007) and Van Vliet (2008) as guide and reference. One initiative of making software engineering knowledge accessible is SWEBOK; a Guide to the Software Engineering Body of Knowledge (Abran, Moore, Bourque, Dupuis & Tripp, 2004). This guide describes generally accepted knowledge about software engineering and is freely accessible. Typical areas that are described in the guide are requirements, design, testing, etc. Although the above references are very good resources for software engineers, some information concerning the development of product software is lacking. For example, the requirements stage in software engineering usually focuses on custom-made software. In SWEBOK, the focus lies heavily on the elicitation, classification and analysis of requirements. Of course these activities play a role in product software companies, but there are some other important processes that are missing. Examples are the identification of market and product requirements, the prioritization of requirements, and the planning of upcoming releases.

Furthermore, the relation between requirements and the product roadmap, the organization of product lines, and managing the product lifecycle are missing in this body of knowledge. In this dissertation, that gap of knowledge is filled by developing a body of knowledge for SPM, the so-called reference framework for Software Product Management.

Another important concept in this dissertation is incremental method engineering. Method engineering is a research domain that has received much attention the last two decades. Especially situational method engineering (Harmsen, Brinkkemper & Oei, 1994; Brinkkemper, Saeki & Harmsen, 1999; Karlsson, 2002; Ralyté, Deneckère & Rolland, 2003) has been the subject of many studies. However, situational method engineering theory has focused mostly on constructing or adapting a method for a certain project, while the evolution of methods that takes place in most companies is not studied. In this dissertation, we show the mechanism of incremental situational method engineering. By formalizing the method increments that occur during method evolution, we provide insight in the method evolution process, which can be used when assembling a method advice. Furthermore, we present a formal approach to incremental process improvement that provides companies with an instrument to improve their methods in an evolutionary way. This vision on process improvement combines a capability-based approach with problem-based aspects. We show how this approach can be implemented in a knowledge infrastructure that uses an assessment to estimate a company's current maturity level, and selects method fragments, based on situational factors and the desired maturity level, to create a method increment advice.

1.3 Positioning the research

1.3.1 Software product management

The last two decades, many product software companies have made a shift from developing custom-made software to developing product software, which Xu and Brinkkemper (2005) describe as "a packaged configuration of software components or a software-based service, with auxiliary materials, which is released for and traded in a specific market. This meant that also their internal processes, and especially, the SPM processes, needed to change. Regular product management is a mature field; it's been established since the industrial revolution in the 19th century (Kilpi, 1997). Nowadays, many books on product management and product marketing can be found, such as Wheelwright and Clark (1992), and Kahn (2005). In the nineties Cusomano (1995), Kilpi (1997)

and Krishnan (1997) were the first to describe *software* product management as a separate business function. The last years, however, more and more professionals and researchers are paying attention to this research domain (Gorchels, 2000; Condon, 2002; Ebert & de Man, 2002; de Man & Ebert, 2003; Dver, 2003; Ebert, 2006).

Xu and Brinkkemper (2005) and Jansen (2007) presented the Research Framework for Product Software, which is depicted in Figure 1-1. Several areas in this framework are currently being researched. Examples are Terlouw, Terlouw and Jansen (2007) on delivery, Dolstra (2003) on deployment, Helms and Van Reijsen (2008) on knowledge management, Lehtola and Kauppinen (2006) and Berander (2007) on requirements and architecture. The SPM processes cannot be positioned on one single part of the research framework. Among others, it entails requirements management, release planning and licensing in the development perspective; market analysis, product lifecycle management and technology management in the company perspective; and intellectual property and markets in the social perspective.

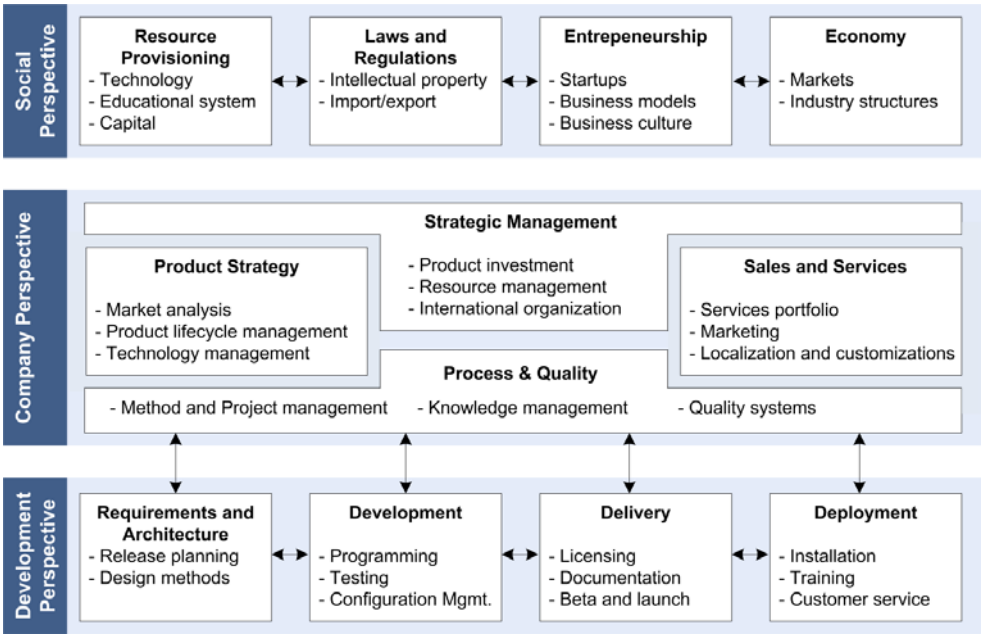


Figure 1-1. Research Framework for Product Software
(Xu & Brinkkemper, 2005; Jansen, 2007)

Software product management is a complex research area. It concerns many internal and external stakeholders and entails a wide range of activities. In this dissertation, we propose the reference framework for SPM, consisting of fourteen activities, divided over four business functions (van de Weerd, Brinkkemper, Nieuwenhuis, Versendaal & Bijlsma, 2006). These business functions are portfolio management, product roadmapping, release planning and requirements management. Portfolio management handles the different products in the product portfolio; product roadmapping deals with the different releases in each product; release planning handles the set of requirements of each release; and requirements management handles the content and administrative data of each individual requirement. Chapter 3 presents an extensive overview of this reference framework. Although the research on SPM in general is not extensive, much research exists on the different business functions that are part of SPM. Carlshamre and Regnell (2000) explored requirements lifecycle management and release planning in market-driven requirements engineering processes. Also Ruhe and Saliu (2005) carried out research in this area and developed a tool to support the release planning process. Karlsson and Ryan (1997) and Berander (2007) focused more specifically on the requirements prioritization process.

Product roadmapping and portfolio management have been less extensively researched than requirements management and release planning. Lehtola, Kauppinen and Kujala (2005) describe their view on long-term product planning by roadmapping as "linking the business view to requirements engineering". In addition, Vähäniitty, Lassenius and Rautiainen (2002) propose an approach to product roadmapping, but then focused on small companies. By the same research group, an approach for managing the development portfolio in small product-oriented software companies is proposed (Vähäniitty and Rautiainen, 2005).

Recently, the research domain of SPM has broadened. For example, Bekkers, van de Weerd, Brinkkemper and Mahieu (2008a) explore the influence of situational factors in software product management. Fricker, Gorschek and Glinz (2008) are working on a different SPM area, namely requirements communication in new product development. Finally, Kittlaus and Clough (2009) published a book on SPM and pricing, in which they describe key success factors for product software companies.

1.3.2 Method engineering

Kumar & Welke (1992) gave a first suggestion to method engineering research by stating that to improve the effectiveness of a method, it should be engineered to the situation at hand, by taking into account the uniqueness of a project situation. Subsequently, Harmsen, Brinkkemper and Oei (1994) and Brinkkemper (1996) laid the foundations for the field of situational method engineering. They defined a situational method as "an information systems development method tuned to the situation of the project at hand".
Several situational method engineering approaches have been described in literature, by e.g. Brinkkemper (1996); Aydin and Harmsen (2001); Saeki (2003); Ralyté, Deneckère and Rolland (2003); and van de Weerd, Brinkkemper, Souer and Versendaal (2006). In addition, method engineering is used for method development (cf. Henderson-Sellers, 2003; Rossi, Ramesh, Lyytinen & Tolvanen, 2004). Also recently, several researchers have used method engineering as a research technique (cf. Luinenburg, Jansen, Souer, Brinkkemper & van de Weerd, 2008; and Levantakis, Helms & Spruit, 2008).

To execute the method engineering process, methods need to be described for which several modeling techniques have been proposed. Saeki (2003), for example, proposed the use of a meta-modeling technique, based on UML activity diagrams and class diagrams, for the purpose of attaching semantic information to artifacts and for measuring their quality using this information. In Van de Weerd and Brinkkemper (2008) this technique is adopted and extended for the purpose of method engineering. With the technique, methods can be analyzed and expressed in so called Process-Deliverable Diagrams (PDDs). Recently, Jeusfeld, Jarke and Mylopoulos (2009) published the book 'Metamodeling for method engineering' which describes a meta-modeling approach in five case studies in different domains.

Finally, research has been done on tool support for method engineering. Jeusfeld, Jarke, Nissen and Staudt In Kelly, Lyytinen and Rossi (1996) and Tolvanen (2006) MetaEdit+ is described; an integrated modeling and meta-modeling environment for domain-specific languages. In Chapter 5, we explore ways to use this tool in our knowledge infrastructure.

1.3.3 Process improvement

One of the main focus areas in this dissertation is the improvement of SPM processes. Software process improvement has been extensively researched for several decades. Several approaches have been proposed to improve software

development processes (Paulk, Weber, Curtis & Chrissis, 1995; el Emam, Drouin & Melo, 1998). Van Steenbergen, Brinkkemper and Van den Berg (2007) call these maturity models continuous and staged 5-level models. These models are very comprehensive, but may typically take thirteen to twenty-four months to be implemented (Process Maturity Profile, 2006). Also, the organizational structure of small companies may not be suitable for a large process initiative (Demirors & Demirors, 1998). Staples at al. (2007) support this in their research where they found that "being a small organization" was one of the main reasons of not implementing a process improvement. Other reasons that they found are the high costs and the large time investment.

Van Steenbergen et al. identify focus area oriented models, in which each focus area has its own number of specific maturity levels. In this dissertation, we will focus on focus area oriented maturity models for SPM in order to make local analysis and incremental improvement possible. Similar model have been used for the testing domain (Koomen & Pol, 1999) and the architecture domain (van Steenbergen, Brinkkemper & Van den Berg, 2007).

1.4 Research description

1.4.1 Research questions

Based on the previous sections, the research question in this dissertation is stated as follows:

MRQ: *How can product software companies improve the maturity of their product management processes?*

The hypothesis on which this research is based is that this question can be answered by providing methodical support to companies in the form of an integrated knowledge infrastructure that manages the complexity of processes in product software companies, and in particular SPM. By using incremental method engineering and meta-modeling principles, method fragments can be stored, selected and assembled to improve existing SPM processes. Three research domains can be identified in the research question: software product management, method engineering and process improvement. Hence, the main research question can be divided into three sub research questions.

RQ1. Which main functions and stakeholders can be identified in the Software Product Management domain?

This research question is posed in order to define the structure and boundaries of the SPM domain. Although this domain received much attention in the business domain, not much scientific research has been done on SPM. To be able to develop a body of knowledge, and to gain insight in the current status of state-of-the-art research on SPM, a structure is needed.

RQ2. How can incremental method engineering be implemented in a knowledge infrastructure?

This research question touches on a few different research topics. First, to be able to store method knowledge in the knowledge infrastructure, a modeling-technique is required. Since existing techniques are not satisfactory for this purpose, a new technique needs to be engineered. Hence, the following sub research question is defined:
RQ2.1. How can we model activities and deliverables in order to reuse them for situational method engineering purposes?

Secondly, the mechanism behind incremental method evolution needs to be investigated. The first step in supporting companies in their method improvement is the exploration of incremental method engineering principles, so that these can be used in a gradual process of improvement. The corresponding research question is:
RQ2.2. How can product software companies improve their software product management methods in an evolutionary way, using method fragment increments?

After exploring the approach to incremental method engineering and formalizing the general method increments, the discovered increment types need to be validated. In addition, the drivers that lead to method increments need to be explored. Furthermore, general method increments that can be used in a tool for method improvement need to be identified and formalized. Consequently, the next research question is defined as:
RQ2.3. Which method increment types occur in incremental method evolution, and which general increment drivers can be identified?

RQ3. *How can we support process improvement in a knowledge infrastructure?*

The aforementioned research questions all contribute to the knowledge of incremental method engineering. These principles need to be implemented in a

knowledge infrastructure that companies can use to obtain a custom-made method advice. Therefore, the next sub research question is defined as:
RQ3.1.How can we use a knowledge infrastructure to support incremental method evolution?

To be able to give advice on a company's SPM processes, an assessment tool needs to be developed and validated that can measure a company's current and desired maturity level in SPM. This corresponds to the research question:
RQ3.2. How can we design a maturity matrix for software product management?

Finally, the incremental method engineering principles and the maturity matrix that will be integrated in the knowledge infrastructure need to be validated. This leads to the final research question:
RQ3.3. To what extent can a knowledge infrastructure, in which method engineering principles and the SPM maturity matrix are integrated, provide a useful method advice for product managers?

1.4.2 Research approach

Hevner, March, Park and Ram (2004) state that most IS research is a combination of two paradigms: design science and behavioral science. In design science, knowledge is obtained by developing artifacts (March & Smith, 1995). This type of science concerns two fundamental questions: "What utility does the new artifact provide?" and "What demonstrates that utility?" (Hevner et al., 2004). Behavioral IS research, on the other hand, "seeks to develop and justify theories that explain or predict organizational and human phenomena surrounding the analysis, design, implementation, management, and use of information systems" (Hevner et al., 2004). The research in this dissertation follows a mixture of both research paradigms, as is depicted in Figure 1-2: the Information Systems Research Framework for the understanding, execution, and evaluation of IS research (Hevner et al., 2004). This conceptual research framework combines behavioral-science and design-science paradigms.

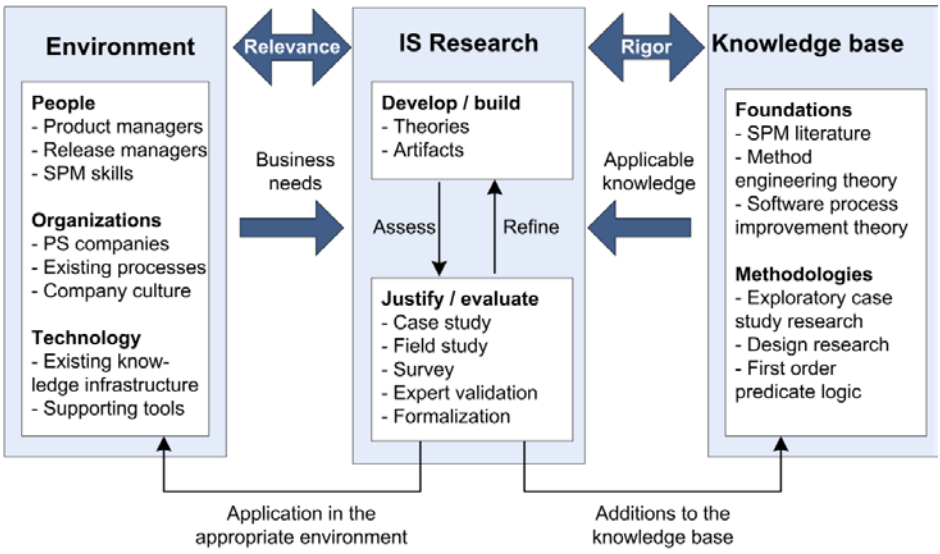


Figure 1-2. Dissertation research applied to the Information Systems Research Framework
(based on Hevner, March, Park & Ram, 2004)

The *Environment* consists of people, organizations and technology. These concepts form the problem space in which the research is conducted. The people that are important in our problem space are mainly the software product managers and release managers who all have their own characteristics and capabilities, but also have many similarities. The organizations in which this research is conducted are primarily product software companies, which all have their own set of implemented processes and situational factors that describe the context in which they operate. In addition, factors concerning company culture and process maturity influence the problem space. Finally, technology is an important factor in the environment. Technology includes the arrangement of the infrastructure and architecture of a company, but also the supporting tools that are used in the SPM processes. People, organizations and technology together define the business needs that are input to the research motivation of a research. The research is set up in such a way that it meets these business needs, hence assuring the research relevance.

The *Knowledge base* "provides the raw materials from and through which IS research is accomplished" (Hevner et al., 2004). Prior research on for example method engineering, software product management and process improvement lays down the foundation for this research in the form of existing theories, methods, models, etc. The methodologies on the other hand describe different

approaches on how we can carry out our research. It consists of data analysis techniques, formalisms, measures, and validation criteria (Hevner et al., 2004). Foundations and methodologies are both forms of knowledge that can be applied during the research. By using these existing foundations and theories research rigor can be realized.

IS Research, as said, is typically a combination of design science and behavioral science. Behavioral science concerns "the development and justification of theories that explain or predict phenomena related to the identified business need" (Hevner et al., 2004). Design science, on the other hand, uses building and evaluation to address the business needs. In this dissertation, different artifacts are built: the reference framework for SPM, the maturity matrix for SPM and PDD meta-modeling technique. By assessing them in field studies, case studies, surveys, and expert validations the artifacts are evaluated. In addition we developed the incremental method engineering theory. By assessing this theory through formalization, case studies and analysis, we justified this theory.

1.4.3 Research methods

Various studies are performed in this dissertation: case studies, survey research, literature study and design research. The use of different research method offers a broader view into the research area. For example, by solely using quantitative methods, it is difficult to find the level of detail that one can find with, for example, a case study.

Table 1.1 provides an overview of the various chapters. Per chapter is indicated whether it is design science and/or behavioral science. In addition, the used research methods are stated. Literature research and expert validation have been omitted in this overview, since these research methods are applied in all chapters.

Table 1.1. Applied research methods

Chapter	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Design science	x	x	x	x		x	x	
Behavioral science	x	x	x	x	x		x	x
Case study research	x	x	x	x	x			x
Survey research							x	

Case study research
In various chapters in this dissertation, case studies are used as a research method. Acoording to Eisenhardt (1989), a strength of using case study research for theory-building is "its likelihood of generating novel theory". By combining evidence of different, sometime contradictory, sources creative insights arise. Secondly, Eisenhardt explains that the resulting constructs are easily testable, since they already have been measured and verified during the theory-building process. Finally, a theory resulting from case study research is "likely to be empirically valid", since the theory is to closely tied to the evidence that it is likely that it will be "consistent with empirical observations" (Eisenhardt, 1989). In addition, case study research is a valid means of theory-testing in information systems research (Bensabat, Goldstein & Mead, 1987). A theory can be validated by a case study, or found to be inadequate, in case it may be refined or improved (Darke, Shanks & Broadbent, 1998).

In Chapters 2 and 4, explanatory parallel case studies are used as a research method. Exploratory case studies make it possible to get an in-depth understanding of a contemporary phenomenon where the investigator has little control over events (Yin, 2003). The two case studies consist of interviews, document studies and tool evaluations.

In Chapter 3, a descriptive case study is included in which the stakeholders' communication concerning the conception, development and launching of a new product at a major software vendor is analyzed.

The case study described in Chapter 5 and 6, is a retrospective case study. The retrospective nature of the case study made it possible to investigate the changes in an organization over a period multiple years; a time period that is almost impossible to investigate by a regular longitudinal research. In this case study several interviews were conducted as well as a thorough document study.

Finally, in Chapter 9, a comparative case study is described in which three cases are analyzed and compared. The comparison increases internal and external validity. In addition, carrying out a comparative case study may reveal patterns that are difficult to spot with other research methods.

Survey research
Chapter 8 describes a survey carried out among 35 software product managers. The goal of this survey is to validate the order of the earlier identified capabilities relative to each other in the SPM Maturity Matrix. The result is a validated maturity matrix for SPM.

1.5 Dissertation outline

The research questions listed in Section 1.4.1 are answered in the various chapters. Figure 1-3 provides an overview of the structure of this dissertation.

Apart from the introduction and conclusion, the dissertation consists of four main parts that correspond with the main research question (MRQ) and the three sub research questions. The first chapter describes the main research question and the vision on how this research question can be answered. The second part contains the domain study on SPM. In 'Analysis of incremental method evolution', the concepts for incremental method evolution are identified, formalized and evaluated in a case study. The chapters in the fourth part describe and evaluate the concept of the knowledge infrastructure.

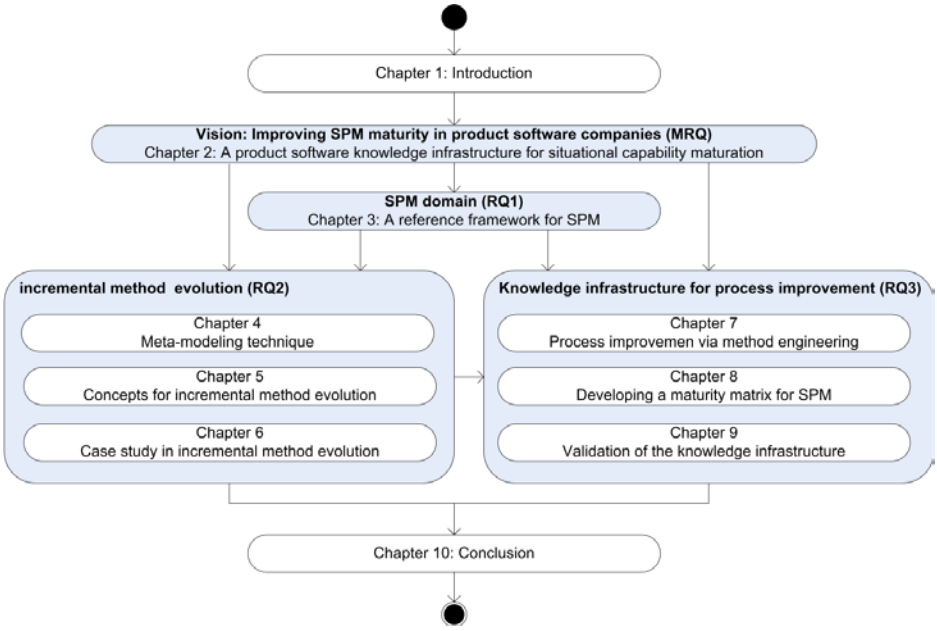


Figure 1-3. Dissertation structure and relations between chapters

Chapter 1. Introduction
The first chapter describes the motivation, relevance, research questions and outline of the research

Chapter 2. A product software knowledge infrastructure for situational capability maturation
In this chapter the overall research question is described. Also, the concept of the product software knowledge infrastructure is explained, which, when fully materialized, can help to increase the maturity of a company's processes. In constructing the infrastructure two exploratory case studies were performed. This chapter was presented and published as a full research paper in the Working Conference on Requirements Engineering: Foundation for Software Quality (van de Weerd, Versendaal & Brinkkemper, 2006).

Chapter 3. A reference framework for software product management
Based on literature studies and field studies, the reference framework for software product management is presented, in which the key process areas, stakeholders and their relations are modeled. To validate the reference framework, a case study is conducted in which stakeholder communication concerning the conception, development and launching of a new product at a major software vendor is analyzed. Finally, the Software Product Management Workbench is proposed, which provides operational support for product managers in product software companies. This chapter is presented and published as a short paper at the International Requirements Engineering Conference (van de Weerd, Brinkkemper, Nieuwenhuis, Versendaal, & Bijlsma, 2006a) and as a full research paper in the International Workshop on Software Product Management (van de Weerd, Brinkkemper, Nieuwenhuis, Versendaal, & Bijlsma, 2006b).

Chapter 4. Meta-modeling for situational analysis and design methods
This Chapter introduces an assembly-based method engineering approach for constructing situational analysis and design methods. The approach is supported by a meta-modeling technique, based on UML activity and class diagrams. Both the method engineering approach and meta-modeling technique are explained and illustrated by case studies. This work is published as a book chapter in the Handbook of Research on Modern Systems Analysis and Design Technologies and Applications (van de Weerd & Brinkkemper, 2008).

Chapter 5. Concepts for incremental method evolution
Chapter 5 describes the identification and formalization of general method increments that are found in an exploratory case study. In addition, common process needs are formalized by developing a root-cause map for software product management and by identifying the root causes and process alternatives that are related to them. The formalized method increments and process needs are validated by applying them to an extensive case study conducted at Infor

Global Solutions. The results show that the formalized method increment types cover all increments that were found in the exploratory case study, and that the root-cause map is a useful technique to model the root causes encountered in product software companies. This work is presented at the International Conference on Advanced Information Systems Engineering (van de Weerd, Brinkkemper & Versendaal, 2007).

Chapter 6. A case study in incremental method evolution
In this chapter, retrospective case study is conducted to explore the method increment types that are defined in Chapter 5. The results show that the method increment types cover all increments that were found in the case study. In addition, several lessons learned for company growth in a global software product management context are described. This research is conditionally accepted for publication in the Journal of Information & Software Technology (van de Weerd & Brinkkemper, forthcoming).

Chapter 7. Process improvement in requirements management
To increase process maturity in systems development, Chapter 7 presents an approach for incremental method evolution that combines capability-based and problem-based methods. With this method, new methods can be assembled, based on the process need of an organization. By using an example of the insertion of cost-value prioritization as a method increment in software product management, the utility of the Product Software Knowledge Infrastructure is shown. This work is presented and published in the International Working Conference on Requirements Engineering: Foundation for Software Quality (Brinkkemper, van de Weerd, Saeki & Versendaal, 2008).

Chapter 8. Developing a maturity matrix for software product management
In this chapter, the maturity matrix for SPM is proposed; a focus area oriented maturity model concentrating on the SPM functions Requirements Management, Product Roadmapping, Release Planning and Requirements Management. Chapter 8 describes the development of the SPM maturity matrix, consisting of (a) identification and description of capabilities, (b) positioning the capabilities at the right levels in the maturity matrix and (c) validating the maturity matrix with expert validation and a survey among 32 product managers. The result is a validated maturity matrix that will guide further development of methodical support in SPM. This work has been submitted for publication.

Chapter 9. Using a knowledge infrastructure for incremental process improvement
This chapter describes a comparative case study in which three case companies are researched. By assessing the companies' SPM processes, we create a maturity profile that serves as a basis for process improvement. The results indicate that the knowledge infrastructure is able to create a useful method advice.

10: Conclusion
In the last chapter, the main research question and the eight sub-research question are answered. In addition, the implications and future research are described.

CHAPTER 2

The product software knowledge infrastructure

Product software companies face the challenge of shipping new re-leases of their software products in time, within budget, with the right quality, and for a good price. As we encountered many performance failures in this respect, we conceptualized a product software knowledge infrastructure, which, when fully materialized, can help to increase the maturity of a company's processes. The infrastructure leverages earlier research on situational method engineering and incorporates the maturity concept. For the sake of in-depth theory demonstration, the infrastructure particularly focuses on product management processes. In constructing the infrastructure we performed case studies at two companies. We found that product management processes increment over time, and become gradually more mature. As such the study of evolution of product management processes is a promising step in building the full product software knowledge infrastructure.[1]

2.1 Situational maturity in product software companies

Product software (PS) companies are highly dependent on the maturity of their product software release processes. Indicators like time-to-market, best features inclusion, software quality, and development costs determine the success of these companies. Many examples of development failures can be found in product software literature. In Carmel (1995), time-to-completion factors are studied in 37 PS companies. Also large organizations, like Netscape and Microsoft, come across difficulties in their product development process Cusumano (2004).

PS companies face complex challenges: an existing customer asks for a bug-fix, and usable features. A sales manager asks for features that convince buyers;

[1] This work was originally published as:

Weerd, I. van de, Versendaal, J., Brinkkemper, S. (2006). A product software knowledge infrastructure for situational capability maturation: Vision and case studies in product management. *Proceedings of the 12th Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'06)*, Luxembourg, 97-112

development project managers ask for features that reduce the duration of a project; the technical architect wants to migrate to a new platform; the support people ask for technical refactoring of existing code; the company board asks for requirements that realize company vision; the finance executive asks for the highest revenue, versus lowest costs; a prospect asks for additional functionality not available in the current version of the product; the development team needs consistent bundles of requirements. Condon (2002) identifies a similar list of stakeholder wishes and needs. Optimal planning, development and shipment of product software are dependent on many situational factors that relate to many stakeholders.

It is our ultimate aim to develop an integrated knowledge infrastructure that manages the complexity of PS companies. As many PS companies have a separate product management function to cope with the mentioned challenges, we relate our infrastructure to product management. Product management covers four main process areas: portfolio management, roadmapping, release planning, and requirements management. Therefore, this area is highly relevant for requirements engineering research. In this research we focus on the process area of product management in the knowledge infrastructure. We include situational method engineering (providing the right method for the right situation, see for example Brinkkemper, Saeki and Harmsen (1999) and Ralyté, Deneckère, and Rolland (2003)) and the concept of evolving maturity: companies can grow in maturity with respect to product management.

2.1.1 Research question and methodology outline

With the described complexity of product management we define the following research question:

> *How can product software companies improve the maturity of their product management processes?*

We address this question by elaborating our vision on situational maturity and by researching the current status of software process improvement (SPI) literature. Next, we map this onto product management and method engineering. Subsequently, we present the Product Software Knowledge Infrastructure and an overview of product management maturity levels. To validate this, we describe case study results of two PS companies and identify their product management processes as applicable today and in the past (different point in time). With the assumption that PS companies have improved their product management processes incrementally over time, this provides us insight in maturity levels for those particular processes (in particular details of

methods and method fragments being used), thus contributing to the building of our knowledge infrastructure. We also identify situational factors that contribute to situational method engineering. We conclude by identifying directions for increasing maturity in product management using situational method engineering.

2.1.2 Situational maturity vision

The typical evolutionary growth of a PS company goes hand in hand with the evolution of its internal processes for product development, marketing, sales, implementation services, and support. Often when a weakness in these processes is identified, an improvement is made by the person in charge.

Important starting points in situational capability evolution are:

- *Incremental method evolution*. The introduction of a complete new development method, such as RUP or DSDM will never work, due to the longer discontinuity of work and the learning curve. Process improvements are performed in smaller incremental steps.
- *Company condition.* Process execution is heavily influenced by the status of the overall PS company. Business-economic circumstances determine the budget and time frame set for the improvements.
- *Organizational culture*. Most companies have their own languages, operational style, and technical platform and tools. Generic process improvements require a translation for the organizational embedding.

Methodical support for PS development requires therefore a situational approach that takes the maturity of the organization into account: simple methods for low maturity companies and more complex methods for companies with a higher maturity. The process maturity needs to be determined by assessing the process execution and deliverable quality, or by utilizing a normative framework for process improvement. The situational context of any process weakness is the starting point for our research. We aim at providing tailor-made support for process improvements based on this concept of situational maturity. Step-by-step the processes will evolve and grow in maturity.

2.1.3 Related literature

In PS design and development several problems can be recognized: a chaotic product concept and architectural design process, lack of a comprehensive and effective development strategy and process, and a weak formal and informal

review of designs, code, and documentation (Cusumano, 2004). To overcome these problems, several SPI approaches have been developed. CMM is a framework representing a path of improvements recommended for software organizations that want to increase their software process capability (Paulk, Curtis, Chrissis & Weber, 1993). The model is divided into five maturity levels: (1) initial, (2) repeatable, (3) defined, (4) managed, and (5) optimizing. When a certain maturity level is reached, the related capabilities of that level are managed. Success of the CMM for the software industry caused the creation of many likewise CMM models in other fields. Eventually this resulted in the development of the broader Capability Maturity Model Integration (CMMI Product Team, 2002). However, some organizations find CMMI too heavy to use [18]. Another SPI model is ISO 9001, the model for quality assurance in design, development, production, installation and servicing (ISO 9001, 1997). It contains 20 clauses that describe the minimal requirements for setting up a quality management system. Although ISO 9001 focuses mainly on the criteria to have an acceptable level of quality and CMM focuses on continuous process improvement, the approaches have a lot in common (Paulk, 1995). A third approach is the ISO/IEC TR 15504 standard, also known as SPICE, which stands for Software Process Improvement and Capability dEtermination (ISO/IEC-15504, 1998).

Although much research is done in SPI, organizations are confronted with limited results. We can identify several reasons for this. Firstly, it is often caused by a lack of strategy to effectively implement the approaches (Niazi, Wilson & Zowghi, 2005). Already in 1996, studies showed that 67% of SPI managers want guidance in implementing a new SPI approach (Herbsleb & Goldenson, 1996). Using tools are one of the ways to guide this implementation process. However, the number of software products produced for this purpose is quite low (Bareisa, Karciauskas & Blazauskas, 2005). Examples are tools with a visual approach to SPI (Hunter & McCallum, 1998) and the Seal tool that that sup-ports data collection and storage procedures (Lok & Walker, 1997).

A second drawback of most SPI approaches concerns the size of the increments used in the process. Incremental approaches are preferable because it is a fundamental way to reduce risk on complex improvement projects (Krzanik & Simila, 1997). However, most SPI approaches imply a revolutionary or large incremental change (Sweeney & Bustard, 1997), which increases the complexity and risk.

In this research, we use method engineering for analyzing and storing information on processes in PS companies. Method engineering is “the engineering discipline to design, construct and adapt methods, techniques and

tools for the development of in-formation systems" (Brinkkemper, 1996). Most research in method engineering is focused on the situation of the project at hand, see e.g. (Ralyté Deneckère & Rolland, 2003), where a generic process model for situational method engineering is developed. In our research, however, we will use method engineering on the organizational and product development process level. To support this, we will use a meta-modeling technique to model activities and work products in product-data diagrams (van de Weerd, Brinkkemper, Souer & Versendaal, 2006; Souer, van de Weerd, Versendaal & Brinkkemper, 2007). This technique is based on a meta-modeling technique for the purpose of attaching semantic information to the artifacts and for measuring their quality using this information (Saeki, 2003). In addition to the process of method engineering and a meta-modeling technique to support this, research has been done to the structuring of method knowledge, cf. Helms, Brinkkemper, van Oosterom and de Nijs (2005). This paper shows how ontologies can be used to model the structure of method knowledge in professional IT organizations using Knowledge Entry Maps that are based on ER diagrams.

2.2 A knowledge infrastructure for product software

To support PS companies with their processes, we propose a Product Software Knowledge Infrastructure (PSKI), envisioned to be accessible via the internet. With this infrastructure, PS companies can obtain a custom-made advice that helps them to improve their processes. The PSKI will be publicly accessible via the internet. It is set up in an academic environment and filled by experiences, not only from academics, but also form PS companies. A schematic overview of the context in which the PSKI functions is illustrated in Figure 2-1. We recognize two main elements in this figure: the PSKI and the PS company. The PSKI is loaded with experiences (acquired via case studies) and existing methods in the PS field. The information that is obtained from these two sources is stored in the method base. The method base stores four types of information:

- *Situational factors* - A situational factor (Brinkkemper, 1996) is any factor relevant for product development and product services. Examples are quantitative factors such as company size and the number of submitted requirements per month; and qualitative factors such as branch and the development method that is currently in use.
- *Capability maturities* – Several capabilities are identified and labeled with a (range of) maturity level(s). The capabilities help in assessing a company's current maturity level, and will help in identifying ways to a higher

maturity. The desired capability maturity depends on the situational factors of a company and the process need.

- *Method fragments* – A method fragment is a coherent part of a method, which contains part of the process (activities) and output (data) of a method and the relations between them. The methods that are administrated are divided into method fragments, in order to be able to easily reuse them (van de Weerd, Brinkkemper, Souer and Versendaal, 2006). In section 3.1 a modeling technique is provided for storing method fragments.
- *Assembly rules* – An assembly rule (Brinkkemper, Saeki & Harmsen, 1999) describes the way in which method fragments should be assembled to a new method, taking the situational into account. An example of such a rule from a situational perspective is that method fragments originating from a waterfall method are not suitable to be implemented in a method containing mostly rapid prototyping method fragments. On the other hand, the internal structure of the method should be consistent. For example, the 'requirements validation'-method fragment will only be applied when an associated requirements document is produced earlier.

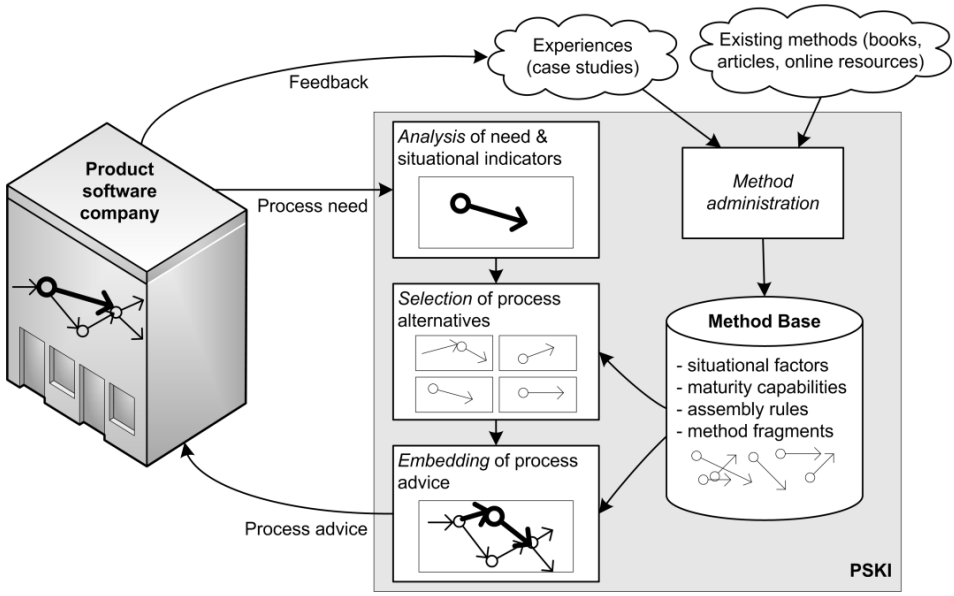


Figure 2-1. Product Software Knowledge Infrastructure

Situational factors, capabilities and method fragments are related to each other. A capability has a certain maturity level, which can range from 1 to 5. For every maturity level, the capability is related to a method fragment. These

method fragments are linked to situational factors, which also influence the desired maturity level. The knowledge of the relations between these concepts is captured in the assembly rules.

The starting point of the method engineering process is the need to improve one or more processes in the PS company. The first step is the analysis of the process need and the situational indicators. This entails an analysis of the current process in terms of activities and work products. Situational indicators contain information about the concerning process and its adjacent processes and general information about the company. Through an assessment of the company's process and situational factors, current capability maturities are determined. The second step is the selection of process alternatives. These alternatives are selected from the method base in the form of method fragments. This selection process is done by selecting method fragments that are linked to the right situational factors and to the desired new capability maturities, addressing the process need. The last step is the embedding step in which the chosen method fragment is fitted into the method. A process advice, which contains a process description, templates and examples, is sent to the PS company. Afterwards, feedback from the company is used for a constant growth and improvement of the knowledge infrastructure. In section 4.6 we elaborate further on the PSKI by giving an example application of the infrastructure.

2.3 Infrastructure operationalization

2.3.1 Process-data diagrams for method administration

For the method administration, a meta-modeling technique is used, which is based on UML (OMG, 2003). With these so-called process-data diagrams we model processes on the left-hand side and data on the right-hand side. In Figure 2-2, an example of a process-data diagram is depicted (taken from the case study reported in section 4).

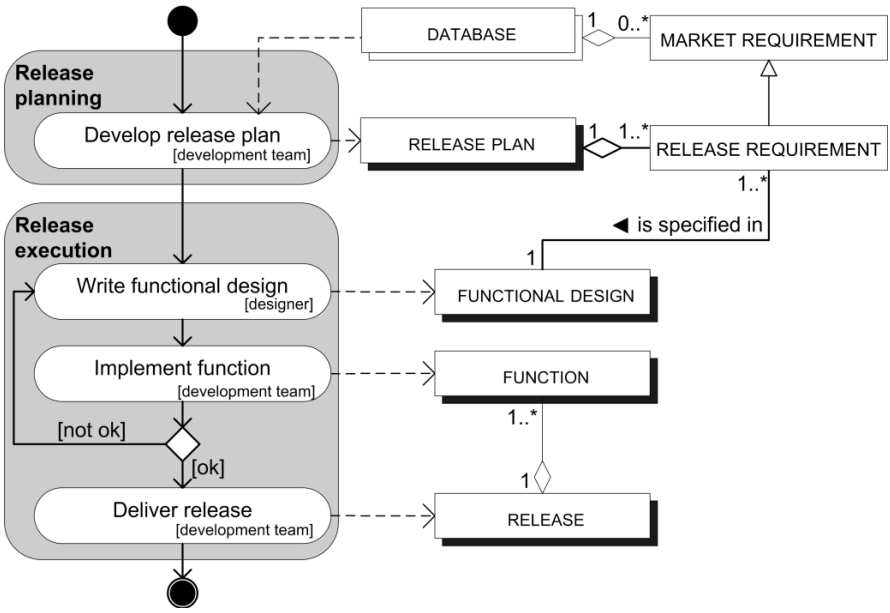


Figure 2-2. Process-data diagram of a release management process

The diagram represents the Release planning activity and the Release execution activity of the release management function. The starting point is the sub-activity Develop release plan. During this activity, data from a DATABASE is used. This DATABASE is created in another activity, which is not depicted here, namely the Requirements management activity. The process ends with the delivery of a new RELEASE.

Some adjustments to the standard UML notation have been made in order to model deliverables of methods. The most important one concerns the use of different types of concepts, which are used to indicate whether a concept is simple or compound. A simple concept does not contain any sub-concepts, whereas a compound concept is an aggregate of sub-concepts.

We define three different ways to model these concepts:

- A simple concept is a concept that contains no further (sub) concepts. A simple concept is visualized with a rectangle. An example is MARKET REQUIREMENT.
- An open concept is an expanded compound concept that consists of a collection of (sub) concepts. An open concept is visualized with an open shadow. An example of an open concept is DATABASE.

– A closed concept is an unexpanded compound concept that consists of a collection of (sub) concepts. A closed concept is visualized with a closed shadow. RELEASE PLAN is a close concept. In this case it is visualized as closed concept because the concept has just partly been expanded. The reason is that for RELEASE PLAN no standard format exists and the content differs from time to time. All we know is that a number of RELEASE REQUIREMENTS are specified in it.

Similarly, open and closed activities are part of the process-data diagrams. For more details of the meta-modeling technique, we refer to Chapter 4.

2.3.2 A maturity framework for product management

For PS companies, product managers are crucial (Condon, 2002). They operate in a complex setting where many internal and external stakeholders are involved, such as customers, partners, R&D, sales and support. Several product management responsibilities can be recognized: requirements management, release planning, product roadmap-ping, and portfolio management (van de Weerd, Brinkkemper, Nieuwenhuis, Versendaal & Bijlsma, 2006). As releasing a product version is a complex process in which much can go wrong we focus particularly on release management.

To be able to provide PS companies with advice to fix their process need and to mature their processes, we need to have a maturity model in which method fragments can be linked with a certain maturity level. Maturity is defined as "the extent to which a specific process is explicitly defined, managed, measured, controlled, and effective; and maturity level is a well-defined evolutionary plateau toward achieving a mature software process" (Paulk, 1995). We use CMM and derivatives such as SPICE (ISO/IEC-15504, 1998) as important sources for constructing a maturity framework for product management.

Adapting CMM has been done earlier, and with success, see for example the Requirements Capability Maturity Model (Beecham, Hall & Rainer, 2005). Others have tried to generalize the concept of maturity beyond the software and engineering domain and determine the impact of maturity on project performance on new product development (Cusumano, 2004).

We distinguish the following maturity levels for product management, derived from literature (Condon, 2002; Gorchels, 2000), in Table 2-1. Following, for each maturity level a number of capabilities will be listed.

Table 2-1. Maturity levels for product management

	Product Management maturity level
Increasing maturity ↑	*Continuous improvement lead by external orientation* Continuous process improvement is enabled by external orientation, e.g. innovative ideas and technologies, customers and partners.
	Organization-wide integration and optimization Detailed measures of the product management process and product quality are collected. Processes and products are quantitatively understood and controlled from an organization wide perspective, not only focussing on products, but on the entire portfolio.
	Product (line) orientation The different processes are standardized, documented and integrated into one standard product management process. The focus is on product level, controlled sequences of releases of product versions.
	Release orientation Basic product management processes are established to track costs, schedule and functionality. The necessary process discipline is in place to repeat earlier successes on similar releases.
	Ad hoc Product management processes are characterized as ad hoc. Few processes are defined and success depends on individual effort.

In this research, we focus on the release management process, which starts with the trigger for a new release of a product and ends with its market delivery. We distinguish three main activities in the release management function: requirements selection, release planning and release execution. Without pretending to be complete, we first identify the capabilities for this function. These capabilities come from literature sources Condon (2002), Dver (2003) and Gorchels (2000); from the second and third author's years of experience in software product management; from the authors' personal network in the Netherlands through the Platform of PS companies; and from the case studies discussed in the next section. In Table 2-2, we list the capability maturity matrix. Every capability is related to one or more maturity levels. We define this as the maturity level range. With each capability / maturity level combination comes one or more method fragments.

Table 2-2.Capability maturity matrix for the release management process

Capability	**PM maturity level**				
	Ad hoc	Release oriented	Product oriented	Organization oriented	External oriented
Portfolio management					
Competitive product strategy determination					x
Identification of product lines				x	x
Establishing product lifecycle management			x	x	x
Identification and translation of market trends into the		x	x	x	x
Distribution partner determination	x	x	x	x	x
Pricing / price determination	x	x	x	x	x
Roadmapping					
Define core assets to use in multiple products			x	x	x
Construct and monitor a roadmap		x	x	x	x
Strategic platform, localization and languages determination	x	x	x	x	x
Release planning					
Trust establishment with customers for release					x
Regulatory acceptance for release			x	x	x
Validation of functional design document		x	x	x	x
Definition of a scope change management process		x	x	x	x
Customer needs determination for release		x	x	x	x
Release promotion determination		x	x	x	x
Gathering release input	x	x	x	x	x
Preparation for release launch	x	x	x	x	x
Functional design document construction	x	x	x	x	x
Clear definition of team responsibilities	x	x	x	x	x
Collateral documentation	x	x	x	x	x
Prioritization of requirements	x	x	x	x	x
Requirements management					
Identifying and structuring similar or related requirements		x	x	x	x
Managing a requirements database		x	x	x	x
Validation of release requirements document		x	x	x	x
Construction of a requirements document	x	x	x	x	x

2.3.3 Example capability

To clarify Table 2-2, we use the capability Functional design document construction. The maturity level range covers all five levels, each with its own method fragment(s):

- *Ad hoc* –No formal rules or templates exist for constructing a functional design.
- *Release-oriented* - This level indicates that per release a functional design document should be delivered. Associated method fragments for this capability deal with communication for the determination of the release with members from the team, and especially the project manager.
- *Product-oriented* – Here, the entire release history is used in the functional design process. The accompanying method fragments treat features from earlier releases, as well as roadmapped release themes.
- *Organization-oriented* - This level implies that other departments and products are involved in constructing and reviewing the functional design document. The corresponding method fragments deal with portfolio and product line issues.
- *External-oriented* – In this level, partners are involved in the construction of the functional design document. An associated method fragment could, for example, cover issues related to the dependencies between the different software components, for which settlements should be made with the development partners.

2.4 Case studies

In order to validate the situational maturity approach for product management, we performed two industrial case studies. The goal of the case studies is:

- to fill the PSKI with method fragments, situational factors and capabilities;
- to analyze method incrementing by taking snapshot of the company's methods in different moments in time and deduct assembly rules from it; and
- to test the validity of the maturity levels defined in Table 2-1 and complement the capability maturity matrix in Table 2-2.

2.4.1 Research method

The case studies have been carried out in the context of the Dutch Platform for Product Software[1], an initiative of a number of PS companies in cooperation with a few research institutions. The purpose of the platform is to exchange knowledge and experiences and to encourage research in the field of product software in particularly the Netherlands. The two case study companies are located in the Netherlands and specialized in developing standard software and providing consultancy services. As for the first company, we focused on a business unit that develops product software for human resource management and payroll administration (hereafter referred to as HRM Software). The other company develops and sells facility management applications (FacMan Software). The case studies concentrate on the areas of release and requirements management.

Information has been collected from the following sources:

- *Interviews* – Explorative 2-hour interviews were conducted with product managers of the case study companies. A second interview session was conducted to cross-reference the obtained results of the first interview and the documentation.
- *Document study* – Documentation provided by the product managers was used to get an overview of the product management processes. Examples of these documents are process descriptions, release planning documents and functional designs.
- *Requirements management tools study* – Several software tools are used to store requirements. The functionality of these tools was studied with respect to their role in the concerning product management process.

In exploratory case studies, three types of validity are important (Yin, 2003). Firstly, construct validity concerns the validity of the research method. We satisfy this type of validity by using multiple sources of data and by maintaining a chain of evidence. Furthermore, we had key informants review the draft case study report. Secondly, the external validity concerns the domain to which the results can be generalized. We are carrying out several case studies in the product management domain in PS companies. In every case study, the same case study protocol is followed. Finally, to guarantee the reliability of the case study, all procedures should be recorded. This is done by using a case study protocol and by maintaining a case study database, which stores all relevant information used in the case study.

[1] www.productsoftware.nl

Each of the case studies resulted in several snapshots of the release management process. A snapshot is defined as a model of the process as it was at a certain moment in time. We collected relevant snapshots of the past (a few years back), and of the present situation. The subsequent snapshots then provide a historical account of the incremental evolution of the release management process. Furthermore, the case studies provided us with useful data for the PSKI. First of all, several method fragments could be distilled from the interviews, presentations and documentation. Secondly, the case studies gave us an indication on the relation of situational factors and capabilities. Finally, the evolution of the release management process also provided constructs to deduce assembly rules.

2.4.2 Situational factor analysis

To illustrate the extraction of situational factors, we describe the analysis from the HRM Software case study. We identified the following situational factors:

- business unit size: 24 employees
- business unit age: 4 years
- new requirements rate: 30-50 per month
- number of customers: 600
- customer base: SMEs, salary processing & accountancy, healthcare
- tools in use: Magic eContact (a CRM-application that is used by the help-desk to store customer wishes) and Mercury TestDirector™ (a profession-al requirements tool, which is only used for requirements storage)

We take two of these factors to describe the consequences they have on the method assembly process. First of all, the company size (in this case business unit size) influences the method choice. A business unit of 24 employees is relatively small. Short communication lines exist between the management and the rest of the employees. Therefore, less control mechanisms and formal processes are needed to get a good performance. Also, changing a process in a small company is much easier than in a large organization. These considerations can be taken into account during the method fragment assembly process.

The other situational factor we explain in detail is the new requirements rate. With a requirements rate of 30-50 per month it is important to keep a good overview of completed, developing, and pending requirements. Therefore all requirements should be documented in a structured way. Also, since not all requirements are equally important, some mechanism should be in place to make a choice of which one to implement in the upcoming release. This situational factor leads to the capability Prioritization of requirements.

2.4.3 Method increment analysis

Each case study provided us with snapshots of the release management process. To illustrate details from growth in maturity, we use an example derived from the case study of HRM Software. In Figure 2-3, the same process example as in section 2.3.1 is illustrated, with the difference that the process snapshot is taken a few years later. All differences between the snapshots of Figure 2-2 and Figure 2-3 are with thick lines.

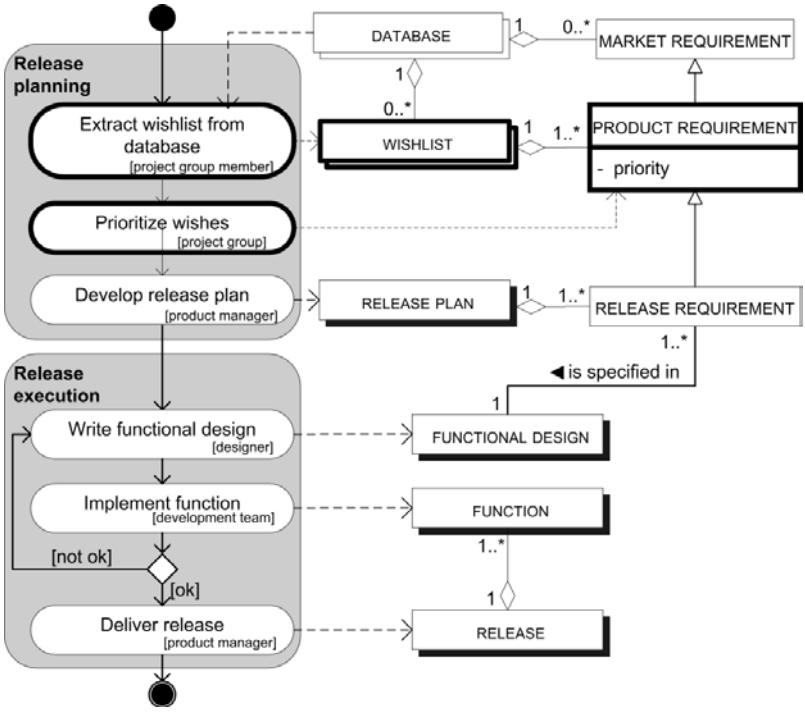


Figure 2-3. Incremented snapshot of the release planning and execution process

We notice that the Release planning activity has evolved. The concept of WISHLIST is used to make a selection of MARKET REQUIREMENTS that should be implemented. The rest of the MARKET REQUIREMENTS are either bugs that are solved directly, or unreasonable requirements that can be omitted immediately by the responsible product group member. Also, an activity Prioritize wishes is added to decide which MARKET REQUIREMENTS will be in the new release. The extension is marked by giving the new activity and data a thicker line. Another difference is the fact that roles are assigned to the sub-activities of the Release planning activity, instead of giving the development team the sole responsibility

for this activity. The Release execution activity has stayed the same. Summarizing the results, the release management process of HRM Software matured in four years from ad hoc to (mostly) the release-oriented level.

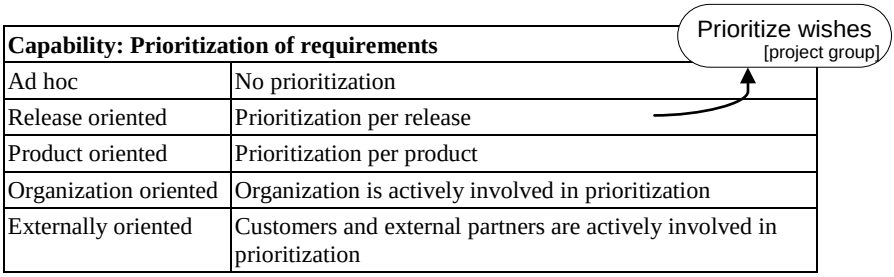
In a similar way the release management process of FacMan Software in our second case study company was analyzed, which cannot be reported here due to space limitations. Three snapshots were identified:

- The process after introduction of the product management function.
- After introduction of the project management method PRINCE2 and the introduction of the conceptual solution document.
- After introduction of the development approach SCRUM (Schwaber & Beedle, 2002).

Accordingly, FacMan software matured in six years from ad hoc to (partly) the product-oriented level.

2.4.4 Capability analysis

When we project the difference between the two process snapshots, described in the section above, to our capability maturity matrix, we can see that the release planning activity has matured. In the first snapshot prioritization is not present in the process. Probably, this was done implicitly by the product manager, which implies an ad hoc maturity. In the second snapshot, the prioritization was done by the entire project group, in order to find the right set of requirements for every release. In the latter case, the method fragment Prioritize wishes can be linked to the release-oriented capability 'Prioritization of requirements', as is illustrated in Figure 2-4.

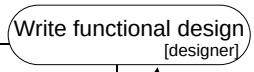


Capability: Prioritization of requirements	
Ad hoc	No prioritization
Release oriented	Prioritization per release
Product oriented	Prioritization per product
Organization oriented	Organization is actively involved in prioritization
Externally oriented	Customers and external partners are actively involved in prioritization

Figure 2-4. Capability - method fragment mapping: Prioritization of requirements

For the capability 'Functional design document construction', we can create a similar figure. Figure 2-5 is a summary of the example in section 2.3.3. The

method fragment Write functional design is linked to the release-oriented level of this capability.



Capability: Functional design document construction	
Ad hoc	No formal process is in place for writing a functional design
Release oriented	Per release a functional design is written
Product oriented	Functional design is derived from functional requirements of the product including multiple releases
Organization oriented	Other departments and products are actively involved in functional design
Externally oriented	Partners are actively involved in functional design

Figure 2-5. Capability - method fragment mapping: Functional design

2.4.5 Deduction of assembly rules

By analyzing situational factors, method fragments and capabilities, we can deduct assembly rules, which describe the way in which method fragments should be assembled to a new method. Looking at the increment that is visualized in Figure 2-3, we ob-serve that the concept PRODUCT REQUIREMENT is placed under the concept MARKET REQUIREMENT by using specification. We can deduce from this the following rule:

> *Inserting a priority activity in a release planning process is only possible if there is a collection of requirements present.*

Another example rule is:

> *Inserting a requirements verification test requires a deliverable as input which includes the requirements selected for the release.*

2.4.6 Example application of the PSKI

In order to illustrate the PSKI in integral operation, we use a fictive example based on the snapshots of the case study we conducted at HRM Software. This PS company notices that the customers appreciated receiving a notification when a requirements suggested by them was completed in the upcoming release. The product manager needs a way to test during release execution whether the requirements were appropriately included in the release. He enters this need into the PSKI by: (a) filling in information about the organization (company size, business unit size, company age, current standard methods, new

requirements rate, etc.) and (b) describing the current release management process in terms of (sub) activities and deliverables. By analyzing situational factors the current maturity level (for most capabilities) is deter-mined as ad hoc.

The PSKI stores the information in the method base and compares the situational indicators and desired maturity of the company with the situational factors and capabilities in the method base, such that matching method fragments can be selected. The situational indicators business unit size (25 employees) and new requirements rate (30-50 per month) imply that:

- a verification test activity is necessary to determine whether the customer requirements have been included in the release; and
- the release plan includes the release requirements and is set as input for such a verification test.

Based on this information, and on the fact that the company wants to grow in maturity, a method fragment of at least the release-oriented level is selected from the method base, which is the release-oriented fragment Perform verification test. In the next step the new method, containing this selected method fragment, is put together and embedded into the existing process (Figure 2-6).

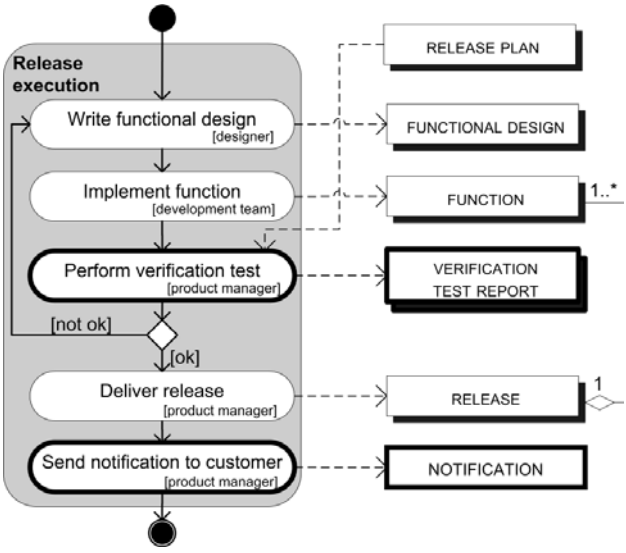


Figure 2-6. Incremented snapshot of the release execution process

This is presented to the company in an advice report. This report contains a process description of the release planning process; an instruction document for

the verification test activity, which can be used by the project team members; and a filled in verification test document as example. Also, the reasons for choosing this method fragment, as well as the origin of the method are explained. Finally, the advice provides guidelines on the best way of implementing the new method into the company.

2.5 Conclusions and further research

Our research question, stated in section 2.1.1 is: how can PS companies improve the maturity of their product management processes using concepts of method engineering and situational capability maturation? We answered this question by describing our vision on situational capability maturation in PS companies.

We introduced the Product Software Knowledge Infrastructure PSKI), to enable PS companies to obtain custom-made advice that helps them to improve their product development processes. An initial version of the capability maturity matrix for particularly the release management process in PS companies is currently developed.

Furthermore, two case studies are carried out to make a first analysis of situational factors, method fragments and capability maturities. The results of the case studies helped us to partly fill the method base of the PSKI for the release management process. Also, the case studies gave us insight in the dependencies between maturity, method fragments and capabilities.

We realize that the capability maturity matrix needs further refinement and broadening. By performing literature studies and additional case studies, we will improve and extend the matrix. Also, matrices for the other product management functions should be developed.

Currently, the PSKI is a concept. We are working on its development and we plan to test it at PS companies. This is not only done by engineering the PSKI, but also by filling the method base with situational factors, method fragments and assembly rules that we will derive from general available theories and case studies.

CHAPTER 3

A reference framework for software product management

Software product management does not get as much attention in scientific research as it should have, compared to the high value product software companies ascribe to it. In this paper, we give a status overview of the current software product management domain by performing a literature study and field studies with product managers. Based on these, we are able to present a reference framework for software product management, in which the key process areas, stakeholders and their relations are modeled. To validate the reference framework, we perform a case study in which we analyze the stakeholder communication concerning the conception, development and launching of a new product at a major software vendor. Finally, we propose the Software Product Management Workbench for operational support for product managers in product software companies.[1]

3.1 Product management

In the past decades, the software market has made a shift from primarily developing customized software to developing software as a standard product. With this shift, a new function within product software companies emerged: the product manager function. In other industrial sectors, especially in

[1] This work was originally published as:

Weerd, I. van de, Weerd, S. de, Brinkkemper, S. (2007). Developing a Reference Method for Game Production by Method Comparison. *Proceedings of the IFIP WG 8.1 Working Conference on Situational Method Engineering: Fundamentals and Experiences, IFIP Volume 244*, 313-327. (short paper)

and

Weerd, I. van de, Brinkkemper, S., Nieuwenhuis, R., Versendaal, J., Bijlsma, L. (2006). On the creation of a reference framework for software product management: Validation and tool support. *Proceedings of the 1st International Workshop on Product Management*, Minneapolis/St. Paul, Minnesota, USA, 3-12. (full research paper)

manufacturing, product management has been established since the industrial revolution in the 19th century (Kilpi, 1997). Recently, product software companies like Microsoft (Cusumano, 1995) and Alcatel (Ebert, 2006; Ebert & de Man, 2002; de Man & Ebert, 2003) paid attention to product management as well. In addition, scientific literature has covered software product management (Kilpi, 1997).

Product management is of critical strategic value in many companies. However, it is also rather complex, since a product manager has many responsibilities covering requirements management, release definitions, and new product launches. What makes these responsibilities even more complex is the fact that the product manager must take the many internal and external stakeholders into account (Condon, 2002; Unger, 2003). Although product management has been established for several decades, software product management has some new challenges. Software products differ from other products in the fact that the manufacturing and distributing of extra copies do not require extra costs for the company (Cusumano, 2004). Also, software products can be changed or updated relatively easy by using patches or release updates. The downside of these advantages lies in the fact that due to the nature of software products, the requirements organization is highly complex. Furthermore, the release frequency is high, since the product can be altered easily. Finally, a software product manager has many responsibilities, but does not have the authority over the development team. Because of these problems, we claim that it is necessary to integrate research efforts in this key domain.

In a few (software) product management areas know-how for research and educational purposes is available, but it is very fragmented. The domain is in need for an integrated body of knowledge, as exists in software development (Bourque & Dupuis, 2004) and project management (PMBOK, 2000). In this paper, we aim to develop a (preliminary) body of knowledge for software product management, by providing a reference framework for all its activities and deliverables. This reference framework has been based on an extensive overview of state-of-the-art literature, industrial case studies, and by exploring opportunities for operational tool support.

The organization of the paper is as follows. In the next section we elaborate on the rationale for the reference framework, and the research method we have applied to develop it. Then, in section 3, we discuss the basic structure of the reference framework. The four process areas are elaborated on in section 4. In section 5, we describe a case study at a major Enterprise Resource Planning software vendor. Subsequently, in section 6, we describe the Software Product

Management Workbench, for operational tool support for the product manager. Finally, we describe our conclusions and future research.

3.2 Rationale and research method

In many fields, reference frameworks have proven to be valuable for research and practice. Examples are the ISO/OSI layers for the layering of network services (ISO, 1994) and the ANSI/SPARC 3-schema architecture for database management systems (Tsichritzis & Klug, 1978). The desire to get an understanding of the complete software product management domain can be satisfied by developing a reference framework. Both research contributions as well as developments in the software industry can be positioned in this reference framework. In this way, the consequences can be interpreted in a uniformed context. Also, the software product management reference framework can provide as a starting point for (a) a definition of key terms in software product management and the identification of open research questions; (b) the education of product managers and competence building; (c) the development of improved, integrated tool support.

The available industrial and scientific knowledge on software product management is limited and fragmented. Therefore, we use a proper mix of empirical and theoretical research steps for conceptualizing the reference framework, which are:

1. Field interviews and discussions with experienced product managers;
2. Literature review on both non-software product management as well as on software product management;
3. Creation of a draft reference framework;
4. Validation by an extensive case study at a large product software company;
5. Validation with input from an industrial workgroup on product management;
6. Finalization of the reference framework.

The resulting draft framework was adjusted several times after suggestions from practitioners and researchers. We do not claim that we now have produced the definitive version of the reference framework. Small enhancements might still be needed, but we are convinced that the basic structure has been established. The framework served furthermore as input for the design of the architecture of the product management workbench.

3.3 Basic framework structure

The nature of software products has a major impact on how the product management function is carried out. Therefore, we base the reference framework on its core, the software product itself, structured in a hierarchical way. Since part of the complexity is caused by the communication with the various stakeholders, we position them to reveal their interactions concerning product management.

3.3.1 Artifact hierarchy

Professional software product management is in essence a matter of well-organized processing of issues related to requirements, products and releases (Condon, 2002; Dver, 2003). A hierarchical ordering of these artifacts (see Figure 3-1) imposes a structure on the process areas.

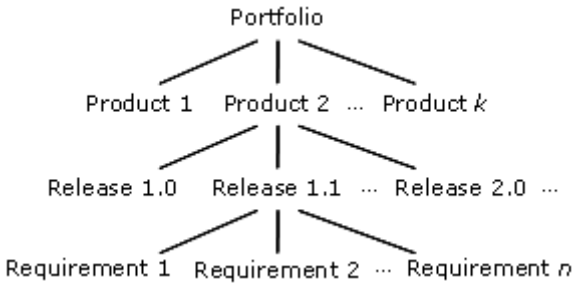


Figure 3-1. Artifact hierarchy of product management

Starting on top, the scope of work of software product management concerns the complete set of products of the company, the so-called *product portfolio*. Small or young companies may have a portfolio of just one product, whereas larger companies have several, due to acquisitions and/or product derivation.

All products have a release sequence of past, present and future *releases*. The release numbering is usually determined by internal conventions, where major changes in the technical architecture are a reason to call it an X.0 release. Marketing reasons may lead to commercial numbering using the year of release or the same release code as an important customer.

Finally, each release definition consists of a set of selected *requirements*. Each requirement implies the addition of a technical or functional feature to the

product. Non-functional requirements are also considered, such as performance constraints or availability requirements.

The type of work differs when dealing with artifacts from the distinct hierarchy levels. The hierarchy gives rise to a subdivision of software product management into four process areas: *portfolio management* to deal with the products in the product portfolio; *product roadmapping* to deal with the different releases each product has, also called roadmapping; *release planning* to deal with the set of requirements of each release; and *requirements management* to deal with the content and administrative data of each individual requirement.

Observe however, that for the sake of diagram clarity, we have swapped the positions of requirements management and release planning in the reference framework (Figure 2). Release planning processes communicate about complete releases to internal stakeholders, whereas requirements management interacts with all stakeholders.

3.3.2 Stakeholder interaction

Software product managers are dealing with many requirements, originating from internal and external stakeholders. We distinguish the following internal stakeholders (Condon, 2002; Dver, 2003):

- The *Company board* 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. Occasionally, requirements are communicated through its strategy, but it can occur that a requirement is sent directly to the product manager.
- *Research & innovation* has two core responsibilities: (1) doing research to new opportunities for product innovations and (2) finding ways to incorporate improvements or new features into the existing products. The first one results in requirements in the form of technology drivers that are communicated to the product manager.
- The consultants of the S*ervices* department are responsible for the implementation of the software product at the customer organization. They need to be aware of new release features and they gather new requirements from the customers.
- *Development* has as main responsibility the execution of the release plan. The release definition also includes functional explanation of the product requirements that serve as input for the functional and technical design. It

may occur that during the development process new requirements can arise, due to more complex requirements than was anticipated.
- *Support* stands for the helpdesk to answer questions (1st line support) and for small defect repair unit (2nd line support). Large defect repair is usually performed by Development (3rd line support).
- *Sales & marketing* is the first contact with a potential customer. Through these contacts new requirements can be gathered.

The following external stakeholders are recognized (Lehtola, Kauppinen & Kujala, 2005):

- The *Market* is an abstract stakeholder, standing for potential customers, competitors and analysts, such as Gartner and Aberdeen. Numerous trends may be recognizable in the market, either in an explicit way by one of the market players, or in an implicit way by product management.
- Most companies have different kinds of *Partners*: (1) implementation partners, who implement the product at a customer; (2) development partners, with whom product components are developed; and (3) distribution partners, selling the product.
- *Customers* often have new feature requests in the process of closing the deal or during the usage of the product. These requests can be communicated to Services, Sales & marketing, Support, but also directly to the product manager.

Observe that the stakeholder names are generic, so that naming or grouping may differ in product software companies. It is obvious that external stakeholders are harder to be influenced in their operational execution and decision making, whereas internal stakeholders should act according to the corporate strategy.

3.4 Reference framework

Little scientific literature explicitly addresses the software product management domain. Only some sub domains, like requirements engineering (e.g. Nuseibeh & Easterbrook, 2000; Potts, 1995; Regnell & Brinkkemper, 2005) and release planning (e.g. Carlshamre, 2002; Karlsson & Ryan, 1997; Saliu & Ruhe, 2005) are covered. Vähäniitty and Rautiainen (2005) found that product portfolio management is largely overlooked in literature, and if it is addressed, it does not mention small and medium sized product software companies. Although software development is largely addressed it adheres to project-related development (Glass, Vessey & Ramesh, 2002). In this section we provide an

overview of the existing state-of-the-art research on (software) product management.

Figure 3-2 shows the reference framework for software product management. The framework was developed after literature research and field interviews with experienced product managers that were employed at six product software companies from the Netherlands. The size of the companies ranges from 75 to 2,700 employees.

Besides the four process areas, we show the sub functions of the product management domain the relations with the internal and external stakeholders. The elements are connected with information flows, indicated with arrows. Note that these flows should not be read as a linear route, but as a continuing, iterative process. In the remaining of this section, each of the four process areas is provided with an explanation supported by research contributions.

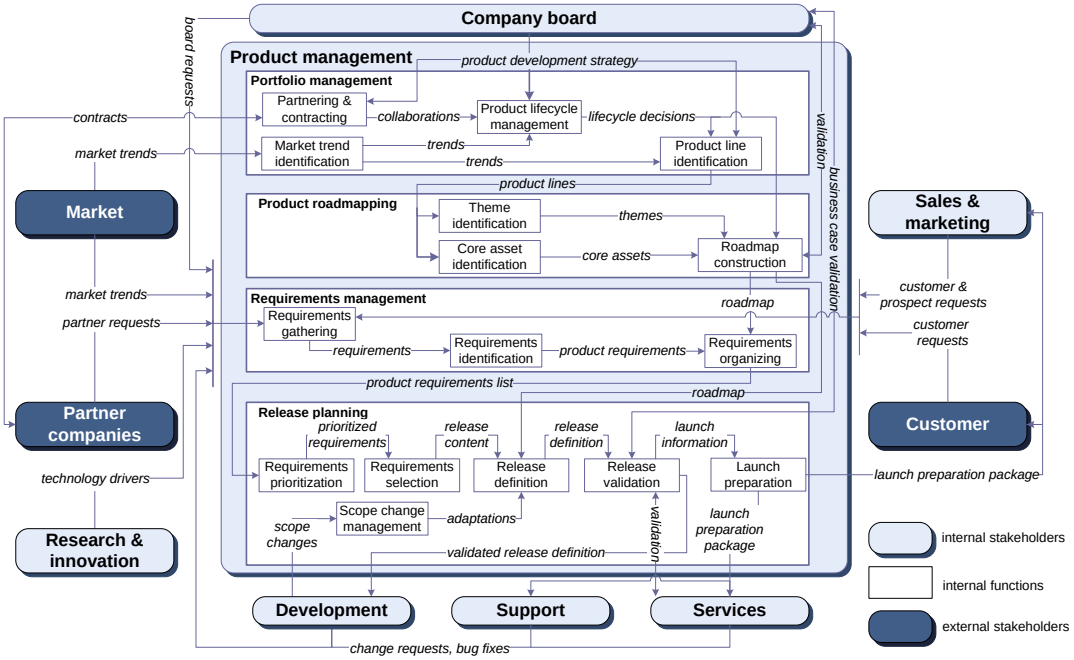


Figure 3-2. Reference framework for software product management

3.4.1 Portfolio management

Portfolio management covers decision making about the set of existing products; introducing new products by looking at market trends and the product

development strategy; making decision about the product lifecycle; and establishing partnerships and contracts. Product line management is positioned in this area as well. Clements and Northrop (2001) define a software product line as "a set 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". Several case studies have shown that introducing product lines organizations improves performance (Bosch, 1999; Brownsword & Clements, 1996; Thiel, Ferber, Fischer, Hein & Schlick, 2001). They are most popular in telecommunication organizations (de Man & Ebert, 2003), but the last years, also the software industry pays more and more attention to this topic (Abramovici & Soeg, 2002; Ardis, Daley, Hoffman, Siy & Weiss, 2000; Clements & Northrop, 2001). Some research has been done to tool support for product lines. An example is Laqua (2002), who proposes a product line content & knowledge base on top of arbitrary configuration management system. Abramovici and Soeg (2002) define product lifecycle management as "a comprehensive approach for product-related information and knowledge management within an enterprise, including planning and controlling of processes that are required for managing data, documents and enterprise resources throughout the entire product lifecycle". This is a key process in decision making about the product portfolio. Also partnering & contracting are important issues in product management (Bonaccorsi & Lipparini, 1994).

Portfolio management is placed on top of the reference framework. It contains the following main processes: partnering & contracting, market trend identification, product lifecycle management and product line identification. The Company board, Market and Partner companies provide input for this process area.

3.4.2 Product roadmapping

According to Vähäniitty, Lassenius and Rautiainen (2002), roadmapping is "a popular metaphor for planning and portraying the use of scientific and technological resources, elements and their structural relationships over a period of time". It is complex due to dependencies on other related products (even from partners), technology changes, and the distributed development (Carmel, 1999). The origins of roadmapping lie in the manufacturing industry. Here, is it used for business oriented long-term planning and technology forecasting (Lehtola, Kauppinen & Kujala, 2005). In the product software industry roadmaps are used for planning purposes (Vähäniitty, Lassenius, & Rautiainen, 2002). Kappel (2001) uses the term roadmapping in two perspectives:

forecasting and planning. Forecasting concerns technology or market trends; and planning concerns products, product lines, resources or the entire company. We use the definition of Regnell and Brinkkemper (2005): "a roadmap is a document that provides a layout of the product releases to come over a time frame of three to five years". It is written in terms of expectations, plans and themes and core assets (Moon & Yeom, 2004) of the product.

As is illustrated in Figure 3-2, product roadmapping receives input regarding product lines from portfolio management. This input is used to identify themes and core assets. Themes are used give a clear direction to the roadmap and later on to structure the requirements. Core assets are components that are shared by multiple products, for example an authorization function that is used by multiple software products. All information is gathered and described in the product roadmap.

3.4.3 Requirements management

Requirements management entails the activities of gathering, identifying and revising incoming requirements and organizing them by keeping in mind dependencies, existing core assets, product lines and themes. Sources are customers, sales and marketing, development, support, R&D and the company's management.

Requirements management is a key area in product software companies (Carlshamre, Regnell, 2000), but Potts (1995) already recognized that requirements engineering for product software is different than for customized software. Regnell and Brinkkemper (2005) recognize the following core requirements engineering activities: eliciting requirements, modeling and analyzing requirements, communicating requirements, agreeing requirements, and evolving requirements. Especially analyzing requirements costs a lot of time in product software companies, due to the (often) high requirements rate, and the different sources of requirements. An example is the use of linguistic engineering to link customer wishes to requirements (Natt och Dag, Gervasi, Brinkkemper & Regnell, 2005). Another problem is the integration of a software product with other systems. Customers cannot expect that all their requirements are met, which may lead to a software product that does not integrate with their existing systems. In Lauesen (2004) several improvements are suggested to this practice. Ebert (2006) investigated the requirements process in 246 industry projects and the results show that four techniques improve schedule performance, if used in parallel: installing of an effective core team for each product release; focusing on the product-lifecycle on upstream

gate reviews; evaluating requirements from various perspectives; and assuring a dependable portfolio and release planning implementation.

The position of requirements management in the reference framework is between product roadmapping and release planning. The process starts with gathering all requirements from within the company and from external stakeholders. The requirements gathered and organized into product requirements. Product requirements are identified by removing the duplicates, connecting requirements that describe a similar functionality, and by rewriting the requirements in understandable product requirements. Then, the requirements are organized per product and core asset. Also, the mutual dependencies between the different product requirements are described. Natt och Dag, Gervasi, Brinkkemper and Regnell (2005) make a distinction between *market requirements*, which refer to wishes related to future products, defined in the customer's perspective and context; and *business requirements*, a product requirement to be covered by the company's products, described in the company's perspective and context. We use a similar distinction. However, we make a distinction between *requirements* and *product requirements*. Requirements refer to all incoming wishes and change requests. This are not only market requirements, but also service requirement, board requests, technological drivers form research & innovation, etc.

3.4.4 Release planning

Software release management is "the process through which software is made available to, and obtained by, its users" (van der Hoek, 1997). Core functions in this process are requirements prioritizing; release planning; constructing and validating a release requirements document; and scope management

Much research has been carried out on the domain of release planning, where the set of requirements for the next release is determined. Examples are release planning using integer linear programming (van den Akker, Brinkkemper, van Diepen & Versendaal, 2005), the analytical hierarchy process (Saaty, 1980), stakeholders' opinions on requirements importance (Ruhe & Saliu, 2005) and linear programming techniques using requirement interdependencies (Carlshamre, 2002). More techniques can be found in the work of Berander and Andrews (2005).

In the reference framework, release planning starts with the product requirements prioritization. Not only the product management is responsible for this, but also the other stakeholders can influence this process. After the prioritization, product requirements are selected that will be implemented in the

next release. This can be done in multiple ways: one can choose the product requirements with the highest priority or use integer linear programming to estimate the best set of requirements. During this process, also the resources have to be applied in the calculations. When the product requirements are selected, a release definition is written that is validated by different stakeholders. A business case is sent to the company board. When this has been approved by the board, a launch preparation package is constructed and sent to the stakeholders.

3.5 Case study

In finding confirmation for the validity of the identified context, activities and relations depicted in the reference framework, we analyzed the conception, development and launching of a new product at a major Enterprise Resource Planning (ERP) software vendor during the period September 2000 to June 2002. The responsible product manager at this company provided us with all incoming e-mail traffic regarding this new product as a source for our analysis. In the mentioned period the product manager received about 1,200 emails related to this product, which serve as the source for this case study.

3.5.1 ERP vendor case

After an organizational repositioning, the management board of the ERP vendor decided to focus on providing add-on products, so-called solutions, next to ERP products. So, from September 2000 onwards, an integrated procurement product was planned, including direct materials purchasing, indirect materials purchasing, e-procurement, e-invoicing and e-kanban (a Just-In-Time purchasing strategy solution), to be integrated via one Supplier Trading eXchange (STX). Note that at the start, some of the functionality was already available in existing products (e.g. direct materials purchasing in the ERP-product), while other functionality needed to be created. Existing and new functionality needed to be disclosed through STX.

As for portfolio management, a number of e-mails represented the assignment of solutions, including the STX solution. Although the board indicated (based on market signals) the necessity of solutions, the product manager verified the need for a specific procurement solution through industry analysts, important customers and competitor analysis. Specifically the successful implementation at Komatsu of a predecessor application of the STX, i.e. the E-Collaboration tool, encouraged the product manager to further prepare

development of the STX. In one of the e-mails the product manager was invited by someone from the ERP vendor's consultancy department to attend a knowledge transfer on E-Collaboration based on of the successful implementation at Komatsu: "I spoke with Komatsu today just to see how things are going and to ask permission to access their site tomorrow for a knowledge transfer session that I am doing for some of the consulting folks and Sales Managers; you are most welcome to call-in". Note that this particular implementation has been described in a case study in a separate paper (Versendaal & Brinkkemper, 2003).

A potential partner company was approached to further enhance functionality regarding the so-called 'round-trip' requisitioning (i.e. linking into suppliers' item catalogues at the suppliers' websites in order to purchase goods from suppliers' sites directly). Integration between the partner's product and STX would make this possible, as one of the e-mails states: "Supplier's product information is dynamically available through agent technology in the partner product's Java code".

Regarding product roadmapping, it became clear that not all topics and themes for an (according to the product manager) ideal procurement solution through the STX could be covered in one release. An example was the late discussion of e-kanban and its incorporation in the future: in one of the e-mails the product manager asked a colleague to "provide me with some compelling arguments why it is good to develop E-Kanban in STX from the business perspective". In general, in many e-mails dealt with themes projection over future anticipated releases of the STX. This included communication with the management board of the company.

Many of the 1,200 e-mails dealt with requirements management and release definition. A number of detailed requirements became clear from the previous implementation at Komatsu. In addition, communication with the support and consultancy departments provided other requirements for the STX. At the end of 2000, an early version of a release definition was communicated with a number of internal departments, including marketing & sales, development, and the release management department. Later on, the architect of the development department interpreted the requirements in a functional design document: "Here is the first draft of functional design document" (e-mail of 9 March 2001). Subsequent e-mails from the development department mainly dealt with requirement clarification ("I need clarification about the off-line purchase in the STX") and scope changes ("shouldn't we support RosettaNet message exchange?").

In cooperation with other departments and associated country organizations the product manager prepared the launch of the STX: e.g. a white paper was written on the product with involvement of marketing and sales (“Sure thing! I'll make sure this is in the plan and we can work together to get it done”.).

3.5.2 Case analysis

In the case study on STX we note that all main product management areas (portfolio management, product roadmapping, requirements management and release management) were addressed. Some areas and some topics within each of the areas were more subject in e-mails than others. For example, product lifecycle management in portfolio management was not so much addressed, as it concerned the first releases of STX, therefore roadmap construction was more extensively addressed. Also, requirements prioritization and selection was not addressed extensively, the scope of the STX, and the list of all requirements was rather small However, proposed scope increases were weighed carefully in order to either include or exclude them: the product manager had to balance between allowing scope creep for development and satisfying sales & marketing.

All identified stakeholders in figure 2 were extensively involved in the communication with the product manager, even for research & innovation: the development department prototyped the round-trip functionality with the STX’s partner product.

The largest category of all the 1200 e-mails came from development. This can be explained by the fact that development took place in another country than the country of origin of the product manager. Much communication was through conference calls and e-mails.

3.6 The Software Product Management Workbench

Product management is key to product software companies and should be addressed and supported well. Although there are several tools supporting part of the product management functionality, they do not provide a coherent and complete set of features dedicated to software product management. Because there is a need for an integrated tool to support the product manager, we propose the software Product Management Workbench. At the same time, we use this workbench to validate our reference framework. We use the identified process areas to outline the architecture of the system.

3.6.1 Existing support tools

Several portfolio management support tools exist, e.g. ProSight's Application Portfolio Management, supporting top-down portfolio management solutions for a company, and UMT's Portfolio Manager Software Suite, a web-based application for portfolio management.

Few support tools for product roadmapping exists. ReleasePlanner (Ruhe & Salie, 2005) covers part of it. ReleasePlanner is a web-based system solution to enable intelligent planning, priority and road-mapping decisions.

Tools that focus especially on requirements management are Borland's CaliberRM for managing requirements throughout the software delivery process and IBM's RequisitePro, a requirements and use case management tool. ReqSimile (van den Akker, Brinkkemper, van Diepen & Versendaal, 2005) is a tool that supports the linkage process in large-scale requirements management, by using a linguistic engineering approach.

Some tools exist in the release planning area. The Accept 360° platform form Accept Software is a product planning and delivery solution that addresses the spectrum of business requirements in all levels of the organization. ReleasePlanner (Ruhe & Saliu, 2005), earlier mentioned in this section, is a uses integer linear programming and prioritization of features for purposes of release planning. This tool focuses on (but is not limited to) software companies. In the Release Planner Provotype (Carlshamre, 2002) a selection algorithm is implemented that presents a number of valid and good release suggestions.

3.6.2 An integrated solution

To provide operational support for software product management, we propose a tool: the Software Product Management Workbench. As explained further, it supports portfolio management, product roadmapping, release planning and requirements management, in an integrated way.

The workbench is divided into four main modules, all intended to aid the product manager with his daily routines. The four modules are: *requirements module*, *release planning module*, *roadmap module*, and *product portfolio module*, their names corresponding to the functionality they provide.

The workbench is designed for different user types. The *product manager* is the main user, but there are three other users that are able to login into the system, all with their own privileges. These three users are: *administrator*, *core asset developer*, and *employee*. Product software companies usually have

multiple software products all furnished with new releases every once in a while. The main task of the administrator is to start new products or new product releases. When a new core asset has been identified, the core asset developer can login into the system and add this new core asset to the system. In this way the product manager can use the core asset in defining a release, and the development team has always access to information on the latest core assets. An employee can logon to the system for reading the latest news of the development progress or report some news about his work on an upcoming release.

3.6.3 Architecture

The Software Product Management Workbench is a so-called *enterprise application*. Building enterprise applications is a hard and daunting task (Fowler, 2003), because they deal with a lot of persistent data, concurrent data access, multiple users with different roles, and are built in a distributed way. In the workbench the difficulties are found in the great amounts of requirements that have to be persistent, different actors that can login into the system, and more. J2EE is a platform that enables the easy creation of enterprise applications, since J2EE handles all the difficult tasks described above for you. This means that enterprise programmers only have to deal with programming the business logic. For technical information of J2EE see Bodoff et al. (2002). In addition, Szyperski (2002) provides a thorough evaluation of the J2EE platform.

Figure 3-3 gives a high level overview of the architecture. The tool uses two types of clients: a *web client* and an *application client*. Application clients run on the client machine and offer the ability to perform heavy calculations on the client machine, without affecting the server. Enterprise Java Beans (EJB) form the core of the J2EE platform that makes the life of an enterprise programmer easier. Two types of EJBs are used in the architecture, namely *entity* and *session* beans. One entity bean represents one row in a database table (or a row in the result of a join operation). Two types of *session* beans exist, which are *stateful* and *stateless* session beans. A stateful session bean can maintain conversational state for one client. A stateless session bean offers its services to multiple clients.

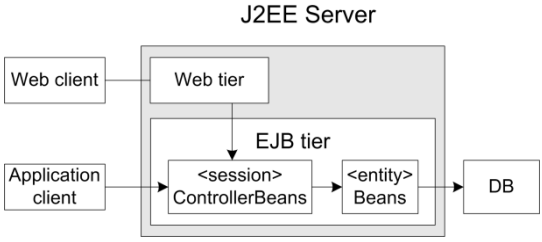


Figure 3-3. High level architecture

The response time, the amount of time it takes for the system to process a request from the outside, is of great importance (Fowler, 2003). The product manager uses functionalities of the tool that require a lot of processing time, so he is the only one able to login into the application client to execute these calculations. The web tier handles all the requests generated by the web client and directs these requests to the controller beans that are deployed into the EJB tier, which provide coarse grained access to the entity beans. The application client accesses the controller beans directly. Note that the web tier and the EJB tier do not have to reside on the same machine.

The extendibility of the tool is also an important issue. As mentioned before there are now four main modules, but the tool should be able to be extended with minimal effort to provide other kinds of functionality. Figure 3-4 shows a small part of the full architecture, but captures some of the patterns used.

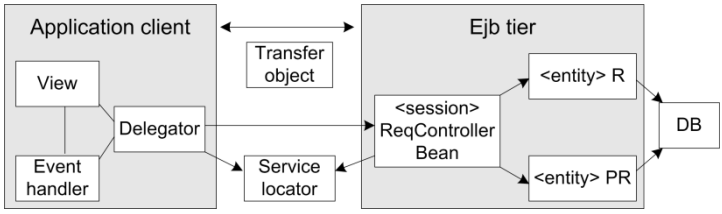


Figure 3-4. Requirements administrator module

The figure shows part of the requirements module, where incoming requirements are connected to product requirements. Remote calls and calls from the web tier to the EJB tier are relatively very slow, so this number should be minimized. The system uses transfer objects that capture as much data that the client possibly wants to get his hands on, instead of getting one piece of data at the time at the cost of one remote call every time. The different components of the system have to be located with so called “look-ups”. It is efficient to extract this code from all the components and put all the look-up code in a

service locator object. In this way, references to components can be cached for other components that may need a reference to that component, minimizing the look-ups. There is no tight coupling between the components, which makes the tool easy to change. If for example the presentation logic has to be adapted, only the view has to be changed leaving the other components unharmed.

3.6.4 Prototype

Figure 3-5 shows a screenshot of the prototype of the Software Product Management Workbench. It shows the requirements window, in which the product manager can link requirements with product requirements that refer to the same functionality (Natt och Dag, Gervasi, Brinkkemper, & Regnell, 2005). At the top of the screen, a list of product requirements is depicted. A product requirement can be selected in order to find matching requirements from the requirements list at the bottom of the screen. After the system has found all the possible candidates, the requirements are displayed together with the source, similarity ratio and the option to link this requirement to a product requirement. When the preferred requirements are selected, the linkage can be saved.

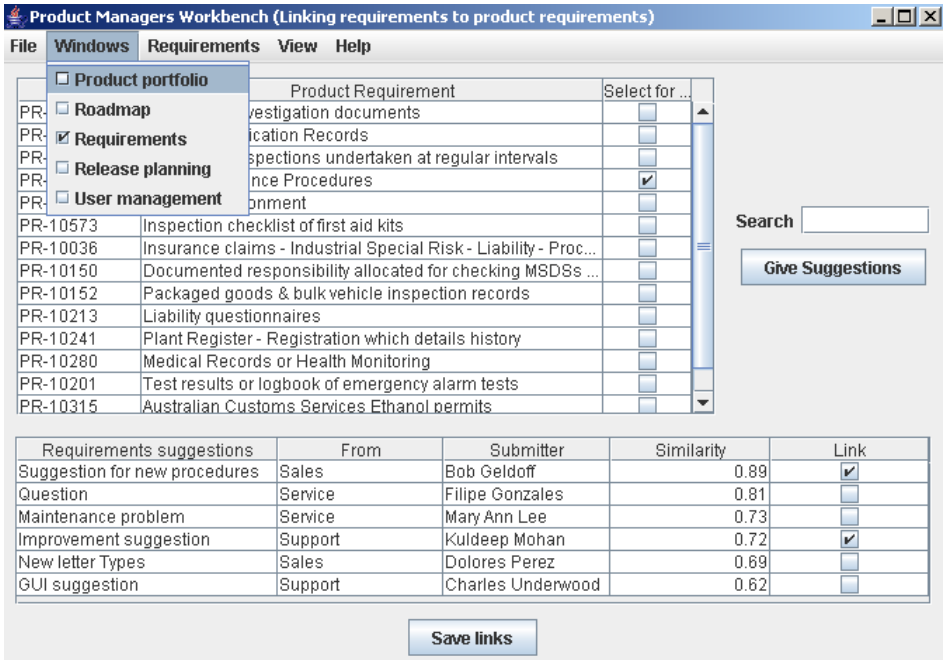


Figure 3-5. Screenshot of the Software Product Management Workbench

3.7 Conclusions and further research

In this article we discussed the difference between product management and software product management, and the need for operational support for the latter. By performing field interviews and discussions with product managers and by doing a literature review on (software) product management, we developed a reference framework for software product management. Furthermore, we provided an overview of state-of-the-art literature on software product management. By carrying out a case study, we found confirmation of the validity of the identified context, processes and relations in the reference framework for software product management. Finally, we proposed the Software Product Management Workbench, which integrates several software product management areas. This workbench is currently being developed. When it is finished, several industrial case studies will be performed to test the functionality.

We are convinced that the software product management reference framework is a first step to position this important industrial domain in the field of scientific research on software product management. In the future, we hope to contribute to further refinements of the reference framework and to its application in various domains.

CHAPTER 4

Meta-modeling for situational analysis and design methods

This chapter introduces an assembly-based method engineering approach for constructing situational analysis and design methods. The approach is supported by a meta-modeling technique, based on UML activity and class diagrams. Both the method engineering approach and meta-modeling technique will be explained and illustrated by case studies. The first case study describes the use of the meta-modeling technique in the analysis of method evolution. The next case study describes the use of situational method engineering, supported by the proposed meta-modeling technique, in method construction. With this research, the authors hope to provide researchers in the information system development domain with a useful approach for analyzing, constructing, and adapting methods.[1]

4.1 Introduction

Many methods for developing information systems (IS) exist. The complexity of the projects in which they are used varies, as well as the situational factors that influence these projects. Over the years, more methods will be developed, as the technology will continue to diversify and new ISs are being developed. However, often methods are too general and not fitted to the project at hand. A solution to this problem is situational method engineering to construct optimized methods for every systems analysis and design situation, by reusing parts, the so-called method fragments, of existing established methods.

In this chapter, an overview of current method engineering research is given. A general approach on situational method engineering is described, as well as a

[1] This work was originally published as:

Weerd, I. van de, Brinkkemper, S. (2008). Meta-modeling for situational analysis and design methods. In M.R. Syed and S.N. Syed (Eds.), *Handbook of Research on Modern Systems Analysis and Design Technologies and Applications* (pp. 38-58). Hershey: Idea Group Publishing.

meta-modeling technique, which supports the process of situational method engineering. The technique will be illustrated in two examples. Finally we describe our future research and conclusions.

4.2 Background

No IS development method exists that is best in all situations. Therefore, to improve the effectiveness of a method, it should be engineered to the situation at hand, by taking into account the uniqueness of a project situation (Kumar & Welke, 1992). Brinkkemper (1996) defined this as "*method engineering*: the engineering discipline to design, construct and adapt methods, techniques and tools for the development of information systems".

A special type of method engineering is situational method engineering. The term *situational method* is defined as "an information systems development method tuned to the situation of the project at hand" (Harmsen, Brinkkemper & Oei, 1994). Situational method engineering is often used in combination with route maps, high-level method scenario's, which can be used to tune the method into situational methods (Van Slooten & Hodes, 1996). Different routes are used to represent the different situations: new IS development, COTS (commercial of the shelf) tool selection, re-engineering, etc.

Several situational method engineering approaches have been described in literature, by e.g. Brinkkemper (1996); Saeki (2003); Ralyté, Deneckère and Rolland (2003); van de Weerd, Brinkkemper, Souer and Versendaal (2006). To execute the method engineering process, methods need to be described for which several modeling techniques have been proposed. Saeki (2003), for example, proposed the use of a meta-modeling technique, based on UML activity diagrams and class diagrams, for the purpose of attaching semantic information to artifacts and for measuring their quality using this information. In Rolland, Prakash and Benjamin (1999) and Ralyté, Deneckere and Rolland (2003), a strategic process meta-model called Map is used to represent process models.

In all research on situational method engineering, several steps are followed in the process to develop a situational method. By comparing the different approaches, we could distinguish the following generic steps in a situational method engineering approach (van de Weerd, Brinkkemper, Souer & Versendaal, 2006):

– Analyze project situation and identify needs.

- Select candidate methods that meet one or more aspects of the identified needs.
- Analyze candidate methods and store relevant method fragments in a method base.
- Select useful method fragments and assemble them in a situational method by using route map configuration to obtain situational methods.

The third and fourth steps are supported by a meta-modeling technique, especially developed for method engineering purposes. This technique, in which a so-called *Process-deliverable diagram* (PDD) is built, is used in analyzing, storing, selecting and assembling the method fragments. The meta-modeling technique is adopted from Saeki (2003), who proposed the use of a meta-modeling technique for the purpose of attaching semantic information to the artifacts and for measuring their quality using this information. In this research the technique is used to reveal the relations between activities (the process of the method) and concepts (the deliverables produced in the process). We will elaborate on this in the next section.

4.3 Process-deliverable diagrams

This section describes the technique used for modeling activities and artifacts of a certain process. As we are modeling methods and not the artifacts of an IS, this type of modeling is called meta-modeling. We express the meta-models of method in PDDs, which consist of two integrated diagrams. The process view on the left-hand side of the diagram is based on a UML activity diagram (OMG, 2003), and the deliverable view on the right-hand side of the diagram is based on a UML class diagram (OMG, 2003). In this chapter first the left-hand side of the diagram is explained, then the right-hand side, and finally the integration of both diagrams.

The meta-modeling technique is explained, where notational explanation will be illustrated by an example in practice. Those examples are taken from a project to create a situational method for implementing web-applications with a content management system (van de Weerd, Brinkkemper, Souer & Versendaal, 2006).

4.3.1 Meta-process modeling

Meta-process modeling is done by adapting the UML *activity diagram.* According to Booch, Rumbaugh and Jacobson (1999), an activity diagram is “a

diagram that shows the flow from activity to activity; activity diagrams address the dynamic view of a system". This diagram consists of activities and transitions. Activities can be decomposed into sub-activities, if necessary, and thereby creating a hierarchical activity decomposition. Transitions can be used to show the control flow from one activity to the next. A simple arrow depicts this. Four types of transitions exist: unordered, sequential, concurrent and conditional activities.

Activity types

We use different types of activities for several reasons. Firstly, activity types are used for scope definition. It may be interesting to know which processes exist adjacent to the process that is focused on. The details of this process are not interesting and are therefore not shown. Secondly, for purposes of clarity of the diagram, some processes are expanded elsewhere. Finally, in some cases the sub activities are not known or not relevant in the specific context.

The following activity types are specified (see Figure 4-1):

- Standard activity: an activity that contains no further (sub) activities. A standard activity is illustrated with a rounded rectangle.
- Complex activity: an activity that consists of a collection of (sub) activities. They are divided into:
    - Open activity: a complex activity whose (sub) activities are expanded. This expansion can be done in the same diagram or in another diagram. Therefore, we use two notational variants, which are:
        - a rounded rectangle, containing two or more sub activities;
        - a rounded rectangle with a white shadow, to indicate that the activities are depicted elsewhere.
    - Closed activity: a complex activity whose (sub) activities are not expanded since it is not known or not relevant in the specific context.

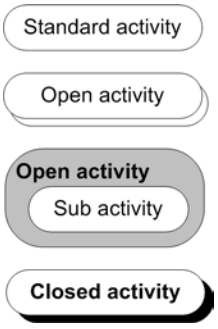


Figure 4-1. Activities types

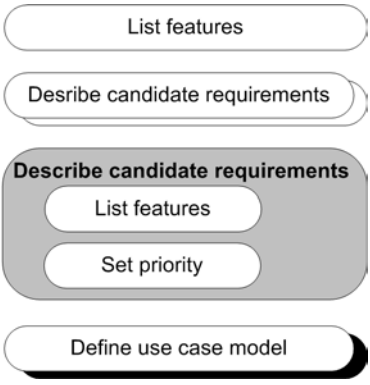


Figure 4-2. Example activity types

In Figure 4-2 we give some examples of activity types. List features is a standard activity that has no further sub activities. In case of Describe candidate requirements, the activity is open, since we find it interesting to describe its sub activities. Define use case model is closed, since we are not interested in the details of this process. The naming conventions for activity names are that the names always consist of a verb and an object, except when the activity name is literally copied from a method source, such as a book. Activities for the highest level are excluded from this naming convention, since they usually get the name of a stage or phase. In the following sections the four activity transitions will be introduced and exemplified.

Sequential activities

Sequential activities are activities that need to be carried out in a *predefined* order. The activities are connected with an arrow, implying that they have to be followed in that sequence, although the activity completion before the next can be started is *not* strictly enforced. Both activities and sub-activities can be modeled in a sequential way. In Figure 4-3 an activity diagram is illustrated with one activity and two sequential sub-activities. A special kind of sequential activities are the start and stop states, which are also illustrated in Figure 4-3.

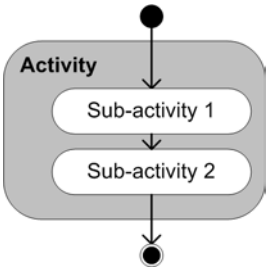


Figure 4-3. Sequential activities

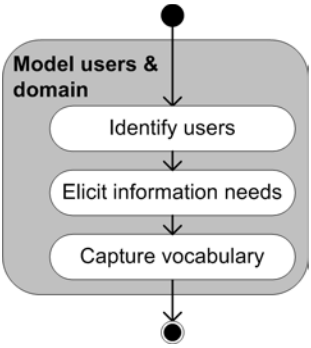


Figure 4-4. Example sequential activities

In Figure 4-4 an example is illustrated. The example is taken from the requirements capturing workflow, of which the main activity, Model users & domain, consists of three activities that need to be carried out in a predefined order.

Unordered activities

Unordered activities are used when sub-activities of an activity can be executed in any order, i.e. they do not have a predefined execution sequence. Only sub-activities can be unordered. Unordered activities are represented as sub-activities without transitions within an activity, as is shown in Figure 4-5.

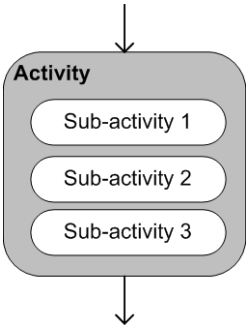


Figure 4-5. Unordered activities

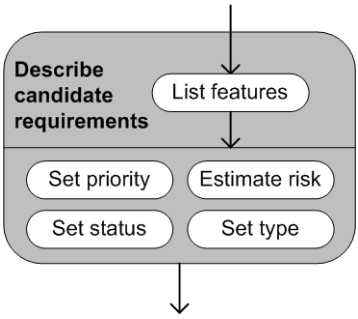


Figure 4-6. Example of unordered activities

An activity may consist of sequential and unordered activities, which is modeled by dividing the main activity in different parts. In Figure 4-6 an example is taken from the requirements analysis workflow. The main activity, Describe candidate requirements, is divided into two parts. The first part is a sequential activity. The second part consists of four activities that do not need any sequence in order to be carried out correctly. Note that all unordered activities must be carried out, before continuing to the next activity.

Concurrent activities

Activities can occur concurrently. This is handled with forking and joining. By drawing the activities parallel in the diagram, connected with a synchronization bar, one can fork several activities. Later on these concurrent activities can join again by using the same synchronization bar. Both the concurrent execution of activities and sub-activities can be modeled. In the example of Figure 4-7, Activity 2 and Activity 3 are concurrent activities.

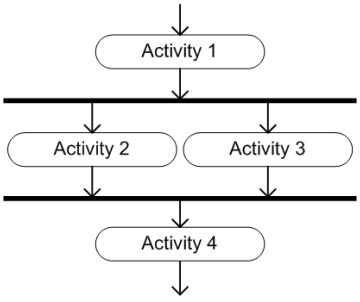


Figure 4-7. Concurrent activities

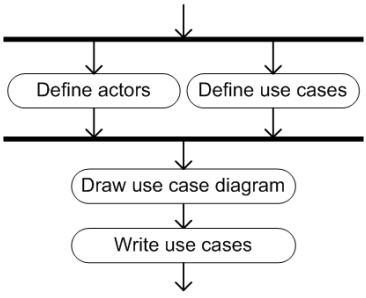


Figure 4-8. Example concurrent activities

In Figure 4-8 a fragment of the requirements capturing process is depicted. Two activities, Define actors and Defining use cases, are carried out concurrently. The reason for carrying out these activities concurrently is that definition of actors and of use cases influence each other to a high extend.

Conditional activities

Conditional activities are activities that are only carried out if a predefined condition is met. This is graphically represented by using a branch. Branches are denoted with a diamond and can have incoming and outgoing transitions. Every outgoing transition has a guard expression, the condition, denoted with square bracket '[ ]'. This guard expression is actually a Boolean expression,

used to make a choice which direction to go. Both activities and sub-activities can be modeled as conditional activities. In Figure 4-9 two conditional activities are illustrated.

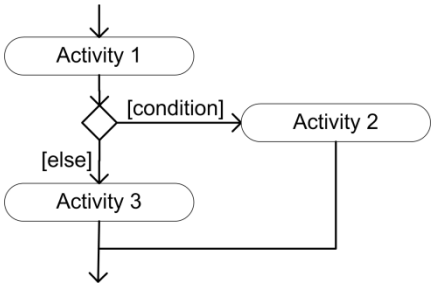


Figure 4-9. Conditional activities

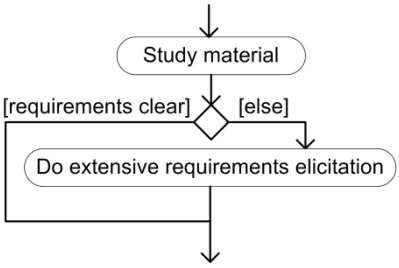


Figure 4-10. Example conditional activities

In Figure 4-10, an example from a requirements analysis starts with studying the material. Based on this study, the decision is taken whether to do an extensive requirements elicitation session or not. The condition for not carrying out this requirements session is represented at the left of the branch, namely [requirements clear]. If this condition is not met, [else], the other arrow is followed.

Roles

In some methods it is explicitly stated by which individuals or organizational role the process is to be carried out. In this case, the role is indicated in the activity. In Figure 4-11, an activity with three sub activities is depicted. In the lower right corner, the role is positioned. Please note, that the usage of roles is not restricted to activities, but, if necessary, can also be used in sub-activities.

In Figure 4-12, the usage of roles is illustrated. The same example as in Figure 4-4 is used with as difference that the role, in this case Consultant, is indicated.

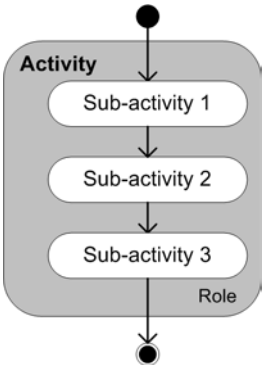


Figure 4-11. Roles

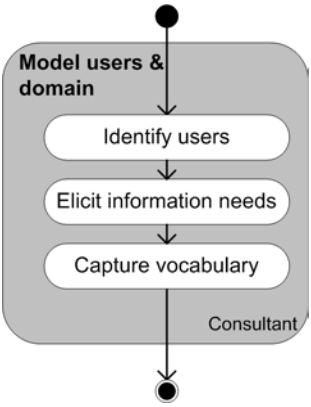


Figure 4-12. Example role

4.3.2 Meta-deliverable modeling

The deliverable side of the diagram consists of a *concept diagram*. This is basically an adjusted class diagram as described by Booch, Rumbaugh and Jacobson (1999). Important notions are concept, generalization, association, multiplicity and aggregation.

Concept types

A concept is a simple version of a UML class. The class definition of Booch, Rumbaugh and Jacobson (1999) is adopted to define a concept, namely: "a set of objects that share the same attributes, operations, relations, and semantics".

The following concept types are specified:

- STANDARD CONCEPT: a concept that contains no further concepts. A standard concept is visualized with a rectangle.
- COMPLEX CONCEPT: a concept that consists of an aggregate of other concepts. Complex concepts are divided into:
- OPEN CONCEPT: a complex concept whose sub concepts are expanded. An open concept is visualized with a white shadow border. The aggregate structure may be shown in the same diagram or in a separate diagram.
- CLOSED CONCEPT: a complex concept whose sub concepts are not expanded since it is not relevant in the specific context. A closed concept is visualized with a black shadow border.

Similar as for activities, this usage of different concept types is introduced for clarity and scope definition.

In Figure 4-13, the three concept types that are used in the modeling technique are illustrated. Concepts are singular nouns and are always capitalized, not only in the diagram, but also when referring to them in the textual descriptions outside the diagram.

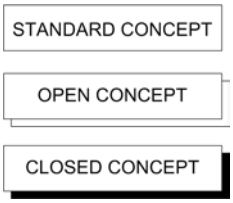


Figure 4-13. Standard, open and closed concepts

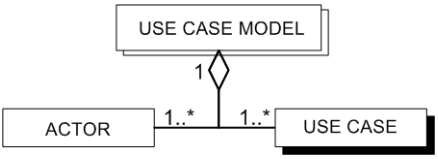


Figure 4-14. Example of standard, open and closed concepts

In Figure 4-14, all three concept types are exemplified. Part of the PDD of a requirements workflow is illustrated. The USE CASE MODEL is an open concept and consists of one or more ACTORS and one or more USE CASES. ACTOR is a standard concept, it contains no further sub-concepts. USE CASE, however, is a closed concept. A USE CASE consists of a description, a flow of events, conditions, special requirements, etc. Because in this case we decided it is unnecessary to reveal that information, the USE CASE is illustrated with a closed concept.

Generalization

Generalization is a way to express a relationship between a general concept and a more specific concept. Also, if necessary, one can indicate whether the groups of concepts that are identified are overlapping or disjoint, complete or incomplete. Generalization is visualized by a solid arrow with an open arrowhead, pointing to the parent, as is illustrated in Figure 4-15.

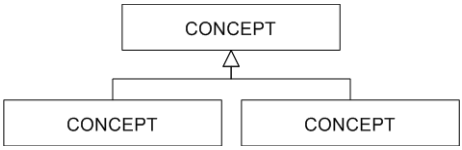


Figure 4-15. Generalization

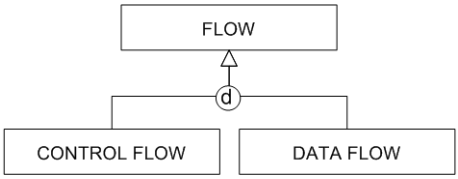


Figure 4-16. Example generalization

In Figure 4-16, generalization is exemplified by showing the relationships between the different concepts described in the preceding paragraph. CONTROL FLOW and DATA FLOW are both a specific kind of FLOW. Also note the use of the *disjoint* (d) identifier. This implies that occurrence of one concept is incompatible with the occurrence of the other concept; that is, there are no CONTROL FLOWS that are also DATA FLOWS and vice versa. The second identifier we use is *overlapping* (o), which would indicate in this example that there may be occurrences that are CONTROL FLOWS or DATA FLOWS, but also occurrences that are both. Finally, *categories* (c) mean that the disjoint concepts have no occurrences in common and that the decomposition is exhaustive. In the example this would imply that every occurrence of flow is either a CONTROL FLOW or DATA FLOW. No other FLOWS exist, and there are no occurrences that are both CONTROL FLOW and DATA FLOW.

Association

An association is a structural relationship that specifies how concepts are connected to another. It can connect two concepts (binary association) or more than two concepts (n-ary association). An association is represented with an undirected solid line. To give a meaning to the association, a name and name direction can be provided. The name is in the form of an active verb and the name direction is represented by a triangle that points in the direction one needs to read. Association with a name and name direction is visualized in Figure 4-17.

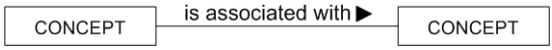


Figure 4-17. Association

Figure 4-18. Example association

In Figure 4-18, an example of a fragment is shown of the PDD of the requirements analysis in the Unified Process (Jacobson, Booch & Rumbaugh, 1999). Because both concepts are not expanded any further, although several sub concepts exist, the concepts are illustrated as closed concepts. The figure reads as "SURVEY DESCRIPTION describes USE CASE MODEL".

Multiplicity

Except name and name direction, an association has another characteristic. With *multiplicity* one can state how many objects of a certain concept can be connected across an instance of an association. Multiplicity is visualized by using the following expressions: *1* for exactly one, *0..1* for one or zero, *0..** for zero or more, *1..** for one or more, or for example *5* for an exact number. In Figure 4-19 association with multiplicity is visualized.

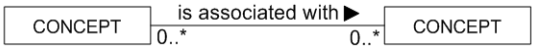


Figure 4-19. Multiplicity

Figure 4-20. Example multiplicity

An example of multiplicity is represented in Figure 4-20. It is the same example as in Figure 18, only the multiplicity values are added. The figure reads as "exactly one SURVEY DESCRIPTION describes exactly one USE CASE MODEL". This implies in an IS project that a SURVEY DESCRIPTION will always be related to just one USE CASE MODEL and the other way round.

Aggregation

A special type of association is *aggregation*. Aggregation represents the relation between a concept (as a whole) containing other concepts (as parts). It can also be described as a 'has-a' or 'consists of' relationship. In Figure 4-21, an

aggregation relationship between OPEN CONCEPT and STANDARD CONCEPT is illustrated. An OPEN CONCEPT consists of one or more STANDARD CONCEPTS and a STANDARD CONCEPT is part of one OPEN CONCEPT.

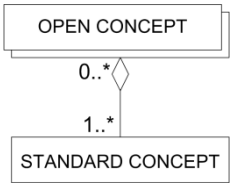


Figure 4-21. Aggregation

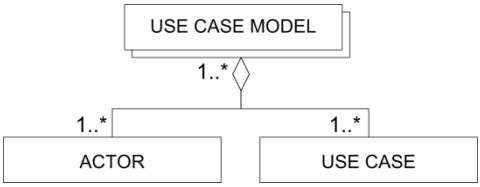


Figure 4-22. Example aggregation

In Figure 4-21, aggregation is exemplified by a fragment of a requirements capture workflow. A USE CASE MODEL consists of one or more ACTORS and USE CASES.

Properties

Sometimes the need exists to assign properties to CONCEPTS. Properties are written in lower case, under the CONCEPT name, as is illustrated in Figure 4-23.

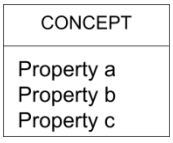


Figure 4-23. Properties

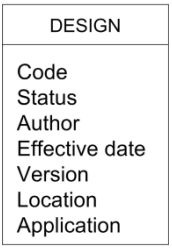


Figure 4-24. Example properties

In Figure 4-24, an example of a CONCEPT with properties is visualized. DESIGN has seven properties, respectively: code, status, author, effective date, version, location and application.

4.3.3 Process-deliverable diagram

The integration of both types of diagrams is quite straightforward. Activities are connected with a dotted arrow to the produced deliverables, as is demonstrated in Figure 4-25. Note that in UML class diagrams, dotted arrows are used to

indicate dependency from one class on another. However, this construction is not used in PDDs. We have some extra remarks concerning the PDD. Firstly, all activities in Figure 4-25 result in deliverables. However, this is not compulsory. Secondly, it is possible for several activities to point to the same deliverable, see for example sub-activity 2 and 3. Finally, we assume that all artifacts in the diagram are available to all roles.

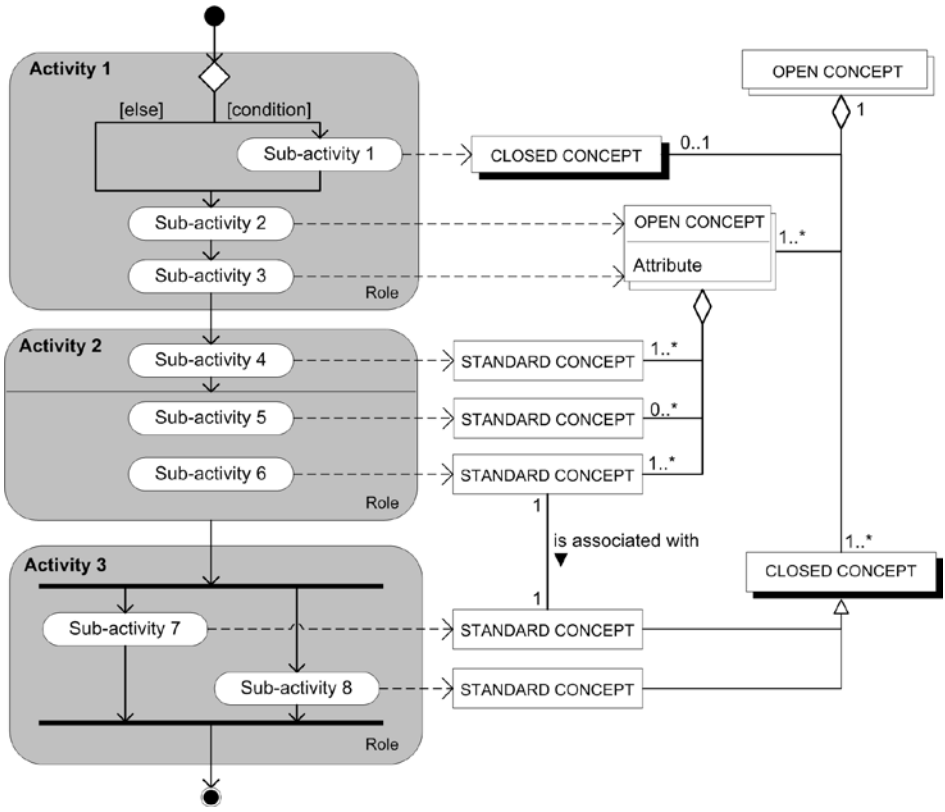


Figure 4-25. Process-deliverable diagram

4.3.4 Example of a Process-deliverable diagram

In Figure 4-26, an example of a PDD is illustrated. It concerns an example of drawing an Object chart, as described in Brinkkemper, Saeki and Harmsen (1999). This is an example of method assembly, in which an Object model and Harel's State chart are integrated into Object charts.

Notable is the use of an open concept: STATE TRANSITION DIAGRAM. This concept is open, since it is a complex concept, and should be expanded elsewhere. However, due to space limitations, this is omitted. The activities of 'Drawing an Object chart' are carried out by one person. Therefore, no roles are depicted in the diagram.

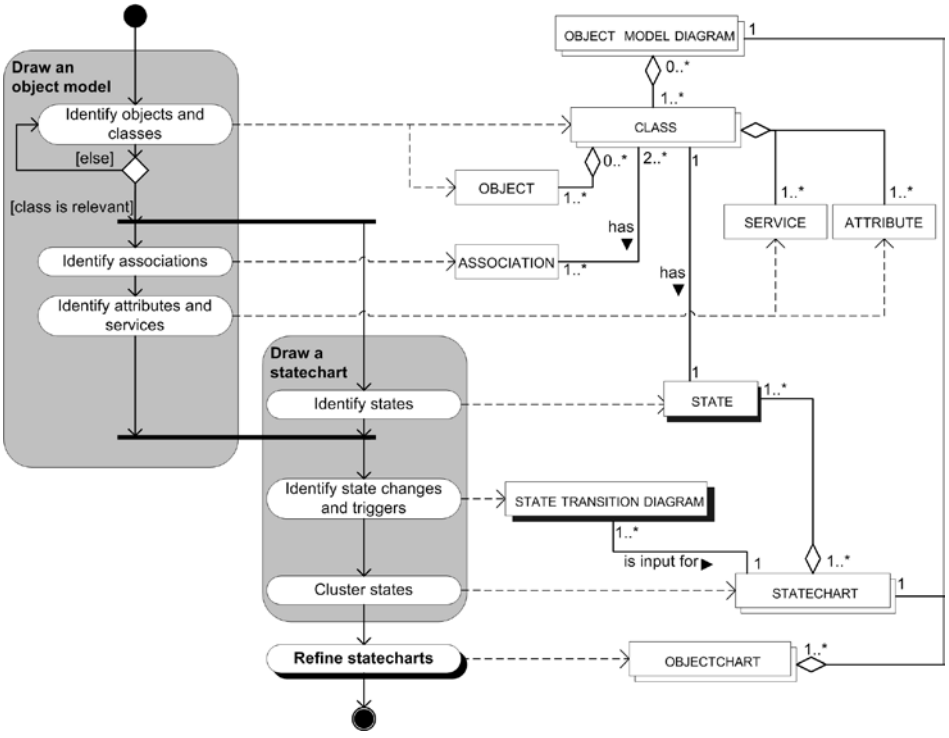


Figure 4-26. Example PDD– Drawing an Object chart

In Table 4-1, the activities and sub-activities, as well as their relations to the deliverables, which are depicted in the diagram, are described.

Table 4-1. Activities and sub-activities in Drawing an Object chart

Activity	Sub-Activity	Description
Draw an object model	Identify objects and classes	Key phenomena that require data storage and manipulation in the IS are identified as OBJECTS. OBJECTS that share the same attributes and services are identified as a CLASS.

	Identify associations	Relationships between CLASSES are identified when instance OBJECTS of different CLASSES are to be related to each other in the IS.
	Identify attributes and services	For each CLASS, the descriptive ATTRIBUTES and the operational SERVICES are identified.
Draw a statechart	Identify states	Lifecycle statuses of OBJECTS that serve a process in the IS are to be identified as STATES. Similarly for CLASSES.
	Identify state changes and triggers	Events that trigger changes of a STATE and the conditions under which they occur are determined.
	Cluster states	STATES that are sub statuses of a higher level STATE are groups together in a cluster. Furthermore, state transitions between STATES are depicted resulting in a STATECHART.
Refine statecharts		Based on the OBJECT MODEL DIAGRAM and the STATECHART, an OBJECTCHART is created by linking STATECHARTS to each CLASS.

In Table 4-2, an excerpt if a concept table is shown. Every concept is provided with a description and a reference.

Table 4-2. Excerpt of a concept table

Concept	**Description**
ASSOCIATION	An association specifies a semantic relationship that can occur between typed instances. It has at least two ends represented by properties, each of which is connected to the type of the end (OMG, 2004).
STATE	A state models a situation during which some (usually implicit) invariant condition holds (OMG, 2004).
OBJECTCHART	An extension of a State chart to model reactive systems from an object-oriented view (Brinkkemper, Saeki & Harmsen, 1999).
…	…

The process in Figure 4-26, where Object model diagrams and State charts are amalgamated into Object charts, is an example of situational method engineering for a project situation where is was required to model State charts for each class. Similarly, situational factors may require various kinds of adaptations of the method. Different analysis and design situations in a project

give rise to a huge variety of method adaptations: in a simple project just one Class diagram may be sufficient to analyze the object domain, whereas in a more complex project, various Class diagrams, Object models and State charts are required. To support the adaptations, PDDs are very instrumental as both modifications of activities as of concepts can be easily documented.

4.4 Meta-modeling for method evolution analysis

4.4.1 Introduction

PDDs can be used to analyze the method evolution of a company over the years. In van de Weerd, Brinkkemper and Versendaal (2007), general method increments were deducted from literature and case studies. The resulting list of general method increments was then tested in a case study at Infor Global Solutions (specifically the former Baan company business unit), a vendor of ERP (Enterprise Resource Planning) software. The time period that is covered in the case study ranges from 1994 to 2006. We analyzed 13 snapshots of the evolution of the software development process at Baan, with emphasis on product management activities. An overview of these method increments is listed in Table 4-3.

Table 4-3 Overview of method increments at Baan

#	**Increment**	**Date**
0	Introduction requirements document	1994
1	Introduction design document	1996
2	Introduction version definition	1998, May
3	Introduction conceptual solution	1998, November
4	Introduction requirements database, division market and business requirements, and introduction of product families	1999, May
5	Introduction tracing sheet	1999, July
6	Introduction product definition	2000, March
7	Introduction customer commitment process	2000, April
8	Introduction enhancement request process	2000, May
9	Introduction roadmap process	2000, September
10	Introduction process metrics	2002, August
11	Removal of product families & customer commitment	2003, May
12	Introduction customer voting process	2004, November
13	Introduction master planning	2006, October

In the next sections, we will illustrate two of these increments, namely increment #3 and increment #4.

4.4.2 Snapshot of increment #3

In Figure 4-27, increment # 3 of the Requirements management and Release definition process a Baan is illustrated.

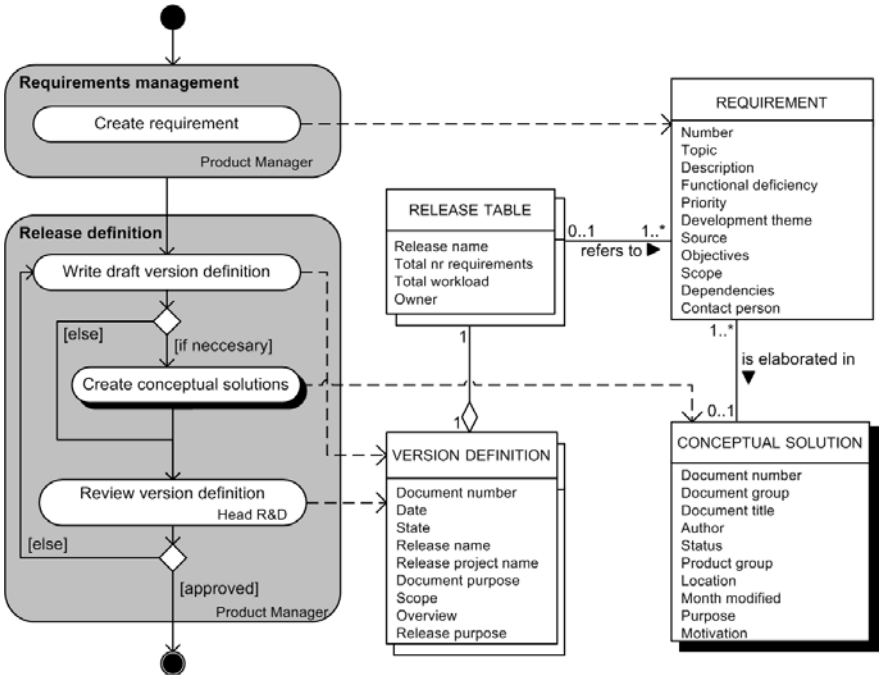


Figure 4-27. Snapshot of increment #3

We can distinguish two main activities in the figure, namely Requirements management and Release definition. The first main activity consists of one sub activity, namely Create requirement, in which the REQUIREMENTS are created by the Product Manager. The other main activity consists of three sub activities. Firstly, the Product Manager writes a first draft of the VERSION DEFINITION. Secondly, the Product Manager writes, if necessary, a CONCEPTUAL SOLUTION per REQUIREMENT. Finally, the VERSION DEFINITION is reviewed by the Head of the Research & Development department. If he approves the VERSION DEFINITION, they can continue with the next activity. If he does not approve it, the Product Manager has to rewrite the VERSION DEFINITION. A more elaborated description

of the activities and the concepts is provided in Table 4-4 and Table 4-5. Note that the activity Create conceptual solutions and the concept CONCEPTUAL SOLUTION are both closed. This implies that they are both complex, i.e. consisting of sub activities or concepts, respectively. In this work, it is not relevant to elaborate further on both elements. However, they are specified in van de Weerd, Brinkkemper and Versendaal (2006).

Table 4-4. Activity table of increment #3

Activity	Sub activity	Description
Requirements management	Create requirement	The Product Manager adds the REQUIREMENT and its properties to the requirements Excel sheet.
Release definition	Write draft version definition	The Product Manager writes a draft of the VERSION DEFINITION.
	Create conceptual solution	If necessary, the product manager elaborates on the requirements in a CONCEPTUAL SOLUTION.
	Review version definition	The Head of the Research & Development department reviews the RELEASE DEFINITION. Either he approves it, in which case the process continues in a new activity, or he disapproves it, in which case the Product Manager rewrites the VERSION DEFINITION.

Table 4-5. Concept table of increment #3

Concept	Description
REQUIREMENT	A REQUIREMENT is a functional description of a new functionality that is to be implemented in the new release of a product.
RELEASE TABLE	The RELEASE TABLE is part of the VERSION DEFINITION, and lists the references to the REQUIREMENTS that are implemented in the new release.
VERSION DEFINITION	A document with the listing of REQUIREMENTS of the new release along with the needed personnel resources. (Natt och Dag et al., 2005)
CONCEPTUAL SOLUTION	A document with a sketch of the business solution for one preferred or more requirements. (Natt och Dag et al. , 2005)

4.4.3 Requirements management & release definition increment

In Figure 4-28, the snapshot of method increment #4 is depicted. Again, the snapshot consists of two main activities: Requirements management and Release definition. The first main activity now consists of three unordered sub activities: the Product Manager creates MARKET REQUIREMENTS, release independent BUSINESS REQUIREMENTS and he maintains the product line.

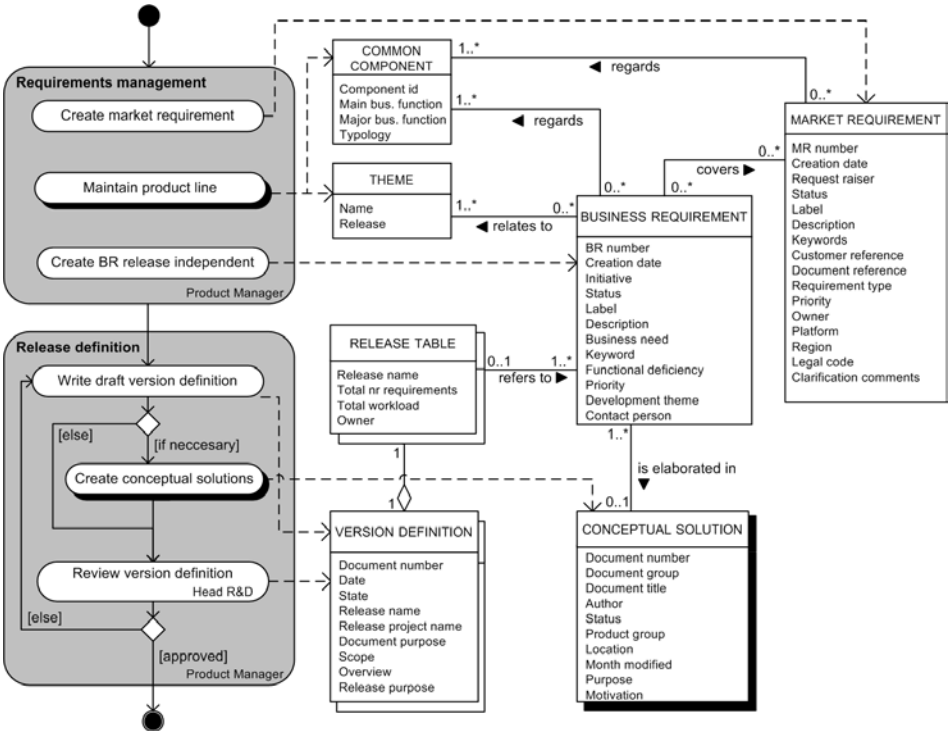


Figure 4-28. Snapshot of increment #4

Notable in this part of the snapshot, is the division between MARKET REQUIREMENTS and BUSINESS REQUIREMENTS. Natt och Dag et al. (2005) give a good elaboration of this separation. They state that MARKET REQUIREMENTS are expressions of the perceived market need, written in the form of a customer's wish. They change frequently, according to the market need changes. BUSINESS REQUIREMENTS, on the other hand, are written from the company's perspective. These requirements are those that finds the company worthwhile to implement one of the following product releases. Both types of requirements are important. MARKET REQUIREMENTS for communicating with the customer and tracing which

customer wishes actually get implemented in the product. PRODUCT REQUIREMENTS are important as a basis for release planning, for conducting feasibility studies, and for a functional description of the feature to be implemented. The separation of the two types ensures a solid basis for decision-making about future releases, as well as a clear tracing device which improves the communication with the customer (Natt och Dag et al., 2005).

The other change in the Requirements management activity is the insertion of an activity called Maintain product line. This activity is not further elaborated in this work, but globally, the Product Manager has to ensure that requirements are linked to themes and common components, in order to maintain a unambiguous product line. More information can be found in van Weerd, Brinkkemper and Versendaal (2006).

The rest of the snapshot is the same as the snapshot of increment # 4. Concluding, the contents of the Requirements management activity have been changed from one to three sub activities. Secondly, the REQUIREMENT concept has been split up into two new concepts: MARKET REQUIREMENT and BUSINESS REQUIREMENT. Finally, the concepts COMMON COMPONENT and THEME have been added. In Table 4-6 and Table 4-7 the activities and concepts are further specified.

Table 4-6. Activity table of increment #4

Activity	Sub activity	Description
Requirements management	Create market requirement	The Product Manager adds a customer wish to the requirements database.
	Maintain product line	Requirements are structured according the THEME they relate to and the COMMON COMPONENT that is affected, in order to maintain the product line.
	Create BR release independent	The Product Managers adds a BUSINESS REQUIREMENTS to the requirements database, by linking MARKET REQUIREMENTS that cover the same subject and by describing the requirement from the company's perspective.
Release definition	Write draft version definition	The Product Manager writes a draft of the VERSION DEFINITION.
	Create conceptual solution	If necessary, the product manager elaborates on the requirements in a CONCEPTUAL SOLUTION.
	Review version definition	The Head of the Research & Development department reviews the RELEASE DEFINITION.

		Either he approves it, in which case the process continues in a new activity, or he disapproves it, in which case the Product Manager rewrites the VERSION DEFINITION.

Table 4-7. Concept table of increment #4

Concept	Description
MARKET REQUIREMENT	A customer wish related to current or future markets, defined using the perspective and context of the user. (Natt och Dag et al., 2005)
BUSINESS REQUIREMENT	A generic customer wish to be covered by a product described in the vendor's perspective and context. (Natt och Dag et al. , 2005)
RELEASE TABLE	The release table is part of the VERSION DEFINITION, and lists the references to the REQUIREMENTS that are implemented in the new release.
COMMON COMPONENT	A software component that is shared by multiple products in the product line.
THEME	A predefined release THEME that is set by the board.
VERSION DEFINITION	A document with the listing of BUSINESS REQUIREMENTS of the new release along with the needed personnel resources. (Natt och Dag et al., 2005)
CONCEPTUAL SOLUTION	A document with a sketch of the business solution for one preferred or more BUSINESS REQUIREMENTS. (Natt och Dag et al., 2005)

4.5 Meta-modeling for method construction

4.5.1 Introduction

The meta-modeling example described in this section covers the assembly of a new method for implementing CMS-based web applications. The research was carried out at GX creative online development, a web technology company in the Netherlands. The company implements web applications, using GX WebManager: a generic CMS-based web application tool that enables people without a specific technological background in creating, maintaining and integrating several dynamic websites and portals. In addition to their product, GX also provides a service, which is creating a web application 'around' the CMS-based Web application. The development of this web application is currently carried out by a proprietary method. However, the need existed to optimize this method in order to save time and money. Also, the need for a standardized web application development method exists, which can be used by

implementation partners of the company. At time of the research, no methods existed for implementing CMS-based web applications. Development methods for conventional Information Systems, as well as for web applications do not cover the needs for this particular type of development. Therefore we developed the GX WebEngineering Method (WEM).

4.5.2 Assembly-based method engineering

We applied an assembly-based situational method engineering approach to develop the new method, consisting of the following steps:

1. Analyze implementation situations and identify needs.

Three implementation situations were identified, namely standard, complex and migration situations. In Table 4-8, example needs are given for the standard and complex implementation situations.

Table 4-8. Example implementation situation needs

Situation	Need
Standard & Complex	The method should deliver a requirements document that is understandable for the customer and informative for the stakeholders at GX.
Standard	Standard project often have a small budget. This implies that the amount of time for specifying the requirements is limited. Therefore, the method should make it possible to translate the requirements quickly into WebManager solutions.
Complex	A solution has to be found to the problem of changing requirements after the contract is signed. Although one can expect the requirements to change during the requirements analysis, the customer often does not understand that this affects the budget.

2. Select candidate methods that meet one or more aspects of the identified needs.

The candidate methods that were selected are: the Unified Software Development Process (UP), UML-based Web-Engineering (UWE) and the proprietary company method (GX).

3. Analyze candidate methods and store relevant method fragments in a method base.

We analyzed the methods by modeling them into PDDs. The method base was filled with four GX PDDs, 2 UP PDDs and four UWE PDDs. In total 10 process data diagrams were stored in the method base.

4. Assemble a new method from useful method fragments and use route map configuration to obtain situational methods.

Based on the implementation situations needs we chose the method fragments for the new method. The resulting method consists of three phases (acquisition, orientation, and definition) and three routes (standard, complex and migration). The rest of the method (design, realization, and implementation) were subject to further research. In the next section we will further elaborate on the resulting method.

4.5.3 Routemap of the definition phase

Instead of showing the entire PDD, we will depict the standard and complex routes in one diagram, to make clear what the differences are between the two implementation situations. Therefore, we omitted the data-side of the diagram, as can be seen in Figure 4-29.

The main activities in the diagram are marked to indicate the original method. A checked pattern indicates that this method fragment originates from the proprietary method at GX; grey indicates that it is a UWE fragment; and, finally, white indicates a Unified Process origin. Note that the roles in this diagram are omitted, because all activities are handled by the same person, namely the Consultant.

The main difference between the standard and complex route is, next to the extensive requirements elicitation and validation, the use of use case modeling. In the complex route this is used to describe the people who will interact with the Web application, as well as how they will interact with the system. In the standard route this is partly handled in the user and domain modeling fragment and partly in the application modeling.

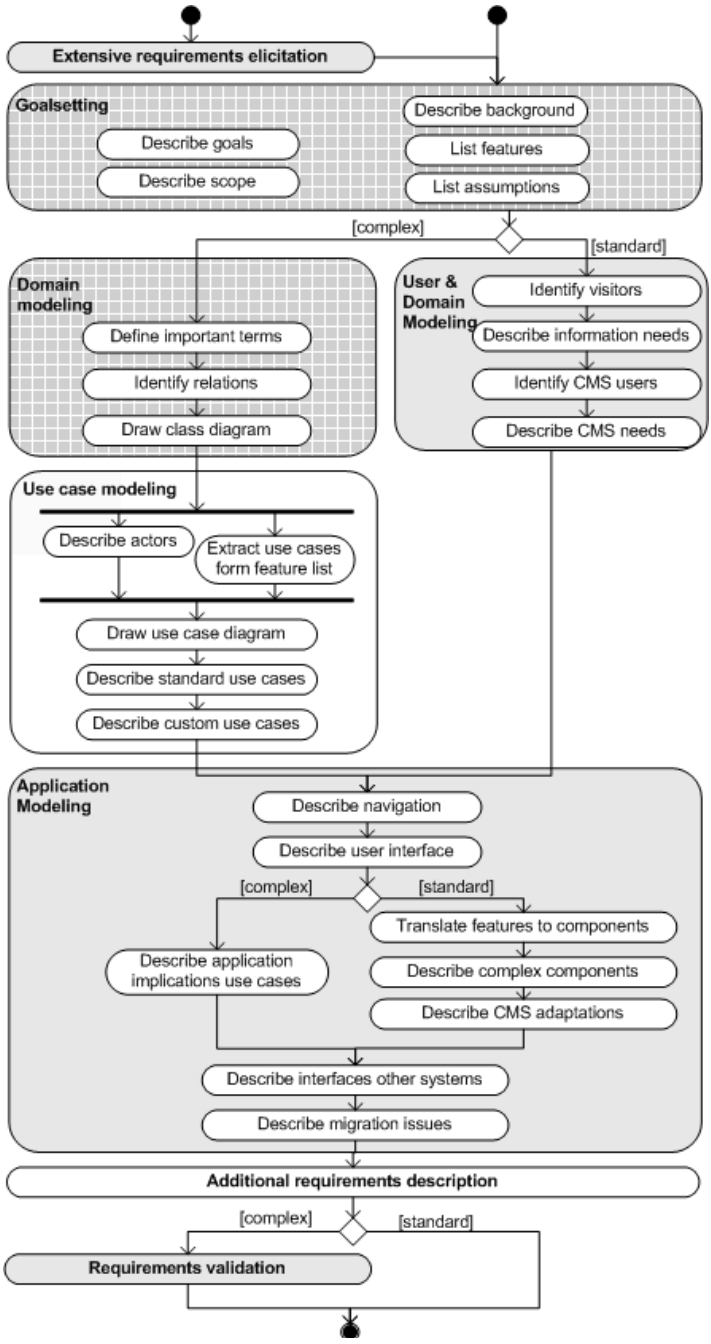


Figure 4-29. Routemap of the definition phase in WEM

4.5.4 Method implementation

WEM was developed with input of the requirements management workgroup. The goal of this workgroup was an overall improvement in the requirements process at GX. Members of the workgroup were consultants and project managers of GX and one external consultant. After validating the WEM method in an expert review and two case studies, the method was implemented in the company. Firstly, templates were written for every document that had to be delivered after completing a method stage. Secondly, explanations were provided with the templates. This information was then published on the intranet of the company and presented for the developers, consultant, architects and project managers.

4.6 Future trends

History has proven that new types of information systems are conceived frequently. Emergent technologies for mobile systems, web applications and intelligent agent solutions have triggered innovative IS applications. At the same time, new methods are developed to design and support these information systems. Existing method knowledge can play an important role, as parts of it can be reused in new project situations, and is codified into new methods. This has also happened when the well-known State transition diagrams where reused into UML (OMG, 2004) for the analysis and design phases of object-oriented IS.

The trend is to build method knowledge infrastructures, i.e. a web-enabled knowledge resource that can be accessed, shared and enriched by all IS project workers. Those knowledge infrastructures can be open to anyone on the internet, such as the Open Modeling Language (Firesmith, Henderson-Sellers & Graham, 1998), an object oriented modeling language. Many large IT service companies have development method on their corporate internet, where every IS project on any location worldwide, can be executed according to the same method. Usually, these corporate methods are enriched by borrowing new concepts from the public ones on the internet. In the future, more and more open method knowledge infrastructures for specific types of IS will be established and gradually expanded into full-blown environments to execute complete IS development projects. The facilities for situational method engineering to adapt the method to specific project circumstances are to be included. An example for the product software vendors is the Product Knowledge Software Infrastructure

that enables product software companies to obtain a custom-made advice that helps them to mature their processes.

4.7 Conclusion

PDDs have proven to be effective means for the meta-modeling of methods, especially for the analysis and design stages. The meta-modeling can serve different purposes: in the examples two purposes are explained, namely method analysis or method construction. Other possible applications are method comparison and method adaptation. Providing the explicit description of the activities and concepts of a method in a PDD allows for a more formal addition of activities and deliverables. Method engineering tools for modeling and adapting methods will aid in the creation of high quality situational methods.

CHAPTER 5

Concepts for incremental method evolution

Product software companies are confronted with performance failures in their processes for which standard theories on situational method engineering need to be revisited. By developing a knowledge infrastructure, we support these companies with their method evolution by increasing the maturity of their processes incrementally. We first identify and formalize general method increments that are found in an exploratory case study. Then, we formalize common process needs, by developing a root-cause map for software product management and by identifying the root causes and process alternatives that are related to them. We validate the formalized method increments, and process needs by applying them to an extensive case study conducted at Infor Global Solutions. The results show that the formalized method increment types cover all increments that were found in the exploratory case study, and that the root-cause map is a useful technique to model the root causes encountered in product software companies.[1]

5.1 Introduction: Incremental method evolution

Many organizations are struggling with the evolution of their information systems development methods (Conradi, Fernström & Fuggetta, 1993). To control this, several software process improvement methods have been proposed (e.g. Emam, Melo & Drouin, 1997; Paulk, Curtis, Chrissis & Weber, 1993), which can be implemented in different ways and which are evolutionary in nature. In our research, we focus on such an evolutionary approach instead of a mere revolutionary approach for several reasons: a) it is a fundamental way to reduce risk on complex improvement projects (Krzanik & Simila, 1997); and b) we observe in practice that this is the natural way for method evolution (van de

[1] This work was originally published as:
Weerd, I. van de, Brinkkemper, S., Versendaal J. (2007). Concepts for Incremental Method Evolution: Empirical Exploration and Validation in Requirements Management. *Proceedings of the 19th International Conference on Advanced Information Systems Engineering, LNCS 4495*, 469–484.

Weerd, Brinkkemper & Versendaal, 2006; van de Weerd, Versendaal & Brinkkemper, 2006).

This evolutionary approach has been subject of research in various scientific studies: methods have been developed to measure and to increase a company's maturity (El Emam, Melo & Drouin, 1997; Paulk, Curtis, Chrissis & Weber, 1993); studies have been carried out to find the best approach to instigate a process improvement (Richardson & Ryan, 2001; Stelzer & Mellis, 1998); and research has been done on the key success factors that influence software process improvement (Rainer & Hall, 2002). However, in 2002, it was estimated that still 70% of software process improvement projects failed (SEI, 2002).

In this work, we choose to take the existing research on software process improvement a step further. Our aim is to develop a knowledge infrastructure that supports product software (PS) companies that build off-the-shelf software products for a market (Xu & Brinkkemper, 2007) in the *incremental evolution* of their methods, by dealing with their process needs and guiding them to higher maturity levels. We keep the increments local (i.e. one process at a time is changed) and small (in comparison to existing incremental approaches with larger increments like CMM (Paulk, Curtis, Chrissis & Weber, 1993) and SPICE (El Emam, Melo & Drouin, 1997)).

In the next section, we describe our research approach, introduce process-deliverable diagrams for modeling methods, and describe the context of this research. In section 3, we define and formalize the process needs. In section 4, we validate the formalized method increments by carrying out a case study at Infor Global Solutions. Finally, in section 5, we describe our conclusions and future research.

5.2 Research approach

Our aim is to support PS companies in their method evolution, by improving parts, or fragments, of their existing methods in an automated way. Method engineering (Brinkemper, 1996) has been used successfully to engineer (parts of) methods for specific situations (Aydin & Harmsen, 2001; Ralyté & Rolland, 2001); to serve as an instrument in software process improvement (van de Weerd, Versendaal & Brinkkemper, 2006); and to use as an approach to manage evolutionary method development by integrating formal meta-models with an informal method rationale (Rossi, Ramesh, Lyytinen & Tolvanen, 2004).

For scoping reasons we limit our research to the software product management domain of PS companies, covering requirements management, release planning, product roadmapping, and portfolio management. In industry, software product management is a clearly defined function, but in science research is fragmented (van de Weerd, Brinkkemper, Nieuwenhuis, Versendaal & Bijlsma, 2006).

5.2.1 Research question and methodology outline

We define the following research question:

> *How can product software companies improve their software product management methods in an evolutionary way, using method fragment increments?*

We address this question by applying method engineering theory. Incremental method engineering has been subject to research by e.g. Krzanik and Simila (1997) and Tolvanen (1998). However, a definition of method increment seems not to be available. Therefore, we define a *method increment* as: a method adaptation, in order to improve the overall performance of a method. Note that adaptation can mean insertion, editing or removal of method fragments.

Actual method increments in industry are explored in an explorative case study at a HRM software vendor (from now on: HRM case study), in order to derive a list of method increment *types* that occur during the evolution. By formalizing and generalizing the increments, we model incremental evolution of a product software company's processes. The formalized increments are then validated in an ERP case study. Using Root Cause Analysis (Root Cause Analysis Handbook, 1999) techniques we determine an initial set of root causes of process needs that PS companies may encounter in the software product management domain. Root Cause Analysis (RCA) has been applied to process improvement and incident prevention in software and non-software industries; see for example (Leszak, Perry & Stoll, 2000). With respect to the HRM case study we determine an initial set of root causes that may lead to process improvement alternatives. This set and our RCA application are also validated in the ERP case study.

5.2.2 Meta-modeling with process-deliverable diagrams

For the analysis of method increments, we use process-deliverable diagrams (PDDs), a meta-modeling technique that is based on UML activity diagrams and

UML class diagrams (van de Weerd, Brinkkemper, Souer & Versendaal, 2006). The resulting PDDs model the processes on the left-hand side and deliverables on the right-hand side (see Figure 5-1). Examples of PDDs can be found in Figure 5-5 and Figure 5-6.

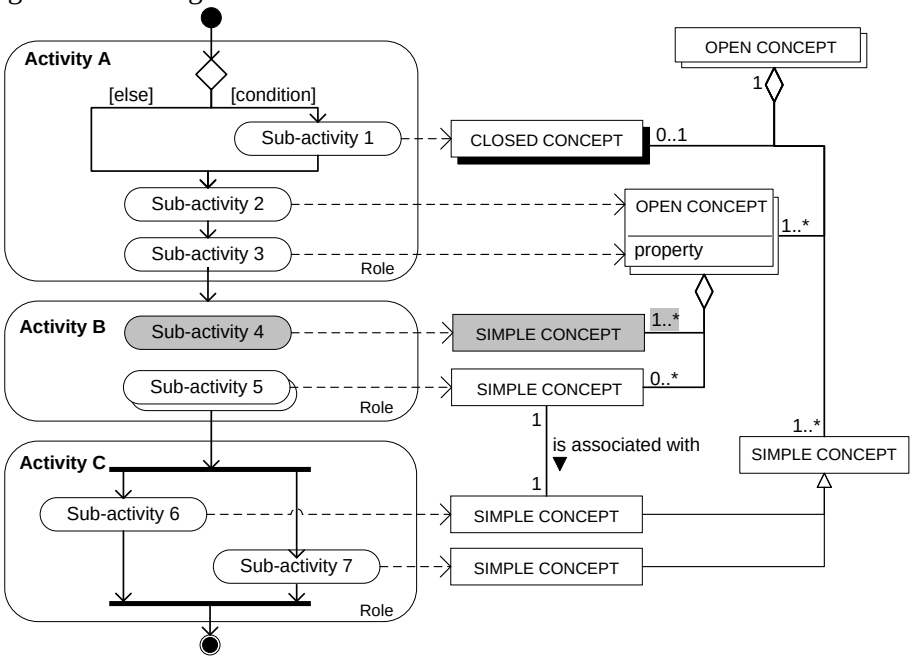


Figure 5-1. Process-data diagram

We follow standard UML conventions (Object Management Group, 2004), but some minor adjustments have been made for modeling development processes. Firstly, deliverables can be **simple** or **compound**. Simple deliverables do not contain any sub deliverables and are visualized with a rectangle. Compound deliverables contain one or more sub deliverables. Compound deliverables can be **open**, visualized with an open shadow, to indicate that it contains sub deliverables. The sub deliverables can be shown in the same diagram, by using aggregation, or in another diagram (for example for space saving). **Closed** compound deliverables, visualized with a closed shadow, indicate that that sub deliverables exist, but are not relevant in this context. Similarly, open en closed activities are used in the diagram. The dotted arrows indicate which deliverables result from the activities. More details on this modeling technique can be found in van de Weerd, Brinkkemper, Souer and Versendaal (2006) and van de Weerd, Versendaal and Brinkkemper (2006).

The PDD, visualized in Figure 5-1, is called a **snapshot**, a model of the process as it was at a certain moment in time (van de Weerd, Versendaal and Brinkkemper, 2006). The evolution of a method over time exists of a number of these snapshots. By comparing snapshots, method increments can be analyzed. In Figure 5-1, we marked sub activity 4 and its corresponding concept. We use this notation to show the method increment of this snapshot compared to a snapshot earlier in time.

5.2.3 A knowledge infrastructure for incremental method evolution

The context in which we want to support PS companies with the incremental evolution of their processes is described in van de Weerd, Versendaal and Brinkkemper (2006), where we propose the Product Software Knowledge Infrastructure (PSKI). Several knowledge repositories for software development methods have been proposed and developed (e.g. the OPEN Process Framework of Henderson-Sellers (2002)). However, the PSKI is not only a knowledge repository, but it also analyzes the process need of a company in order to deliver meaningful advice. In Figure 5-2, the PSKI is illustrated as well as the PS company that interacts with it. The PSKI contains a method base, in which method fragments, situational factors, maturity capabilities and assembly rules are stored.

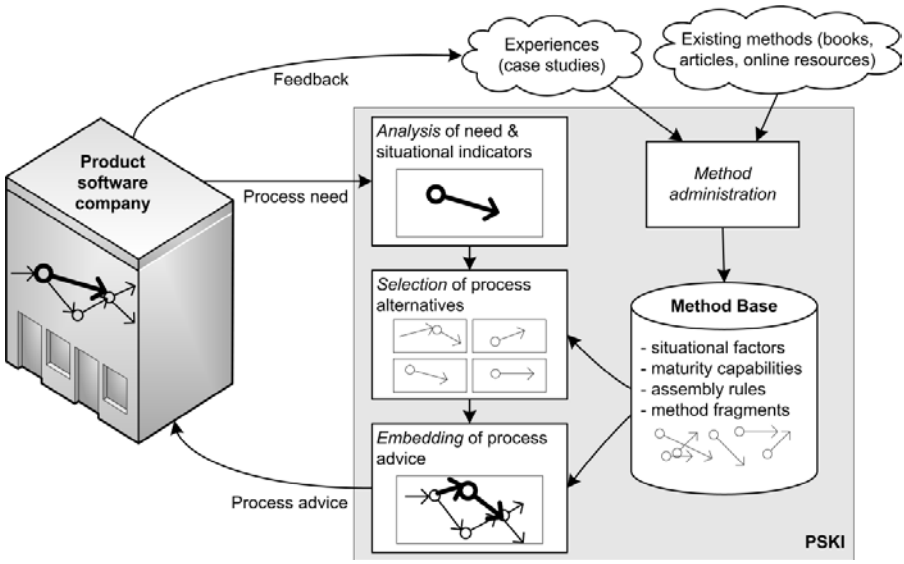


Figure 5-2. Product Software Knowledge Infrastructure

Analysis of need and situational indicators

The first step is the analysis of the process need and situational indicators. The process need is analyzed using Root Cause Analysis, (RCA). Through RCA the root causes of a process need are determined using the following sequence (cf. Leszak, Perry and Stoll, 2000; Root Cause Analysis Handbook, 1999): 1) which *process difficulties* actually occur; 2) what are the so-called *causal factors* of the difficulties; and 3) what are the actual *root causes* per causal factor, using a root cause map designed for PS companies. We define a root cause as (one of) the underlying reasons of a process need, solving one or more causal factors, and relating to one or more actors, activities and deliverable concepts (referring to Figure 5-1). Situational indicators contain information about the process and the company. Examples are company size, development platform and sector.

Selection of process alternatives

Once the root causes are known for a process need, directions for software process improvement can be sought taking into account situational factors. For this, the method base is used. Links between maturity capabilities and root causes are available in the method base in order to identify possible process improvement alternatives. Examples of maturity capabilities are listed by van de Weerd, Brinkkemper & Versendaal (2006). We define a process alternative as a method fragment of a particular maturity capacity that settles one or multiple root causes of the process need.

Embedding of process advice

The last step is embedding the process advice in the company's existing processes. A process advice, which contains a process description, templates and examples, is sent back to the company. The person responsible for process improvement at the company will then start the organizational deployment of the process advice. This roll-out process also includes the insertion of the increment in the existing processes.

5.3 Definition and formalization

This section defines and formalizes method increments and the development problems that lead to these increments. The rationale for this formalization is twofold: First, we use it to analyze the method increments that we found in the ERP case study (see Section 5.4). Secondly, the formalization is used as a first step to develop a formal structure for the method base of the PSKI in which

method fragments can be edited. Firstly, we define method evolution, snapshot and method increments. Then, based on the meta-meta model of PDDs we present a list of all possible increment types with some method fragment insertion rules. Thirdly, we analyze problems that lead to the method increments and develop a root cause map for software product management (RCM for SPM).

5.3.1 Definitions of incremental method evolution

As the PDD technique is based on UML, we can utilize the available formalizations in the literature. There appears to be two kinds of formalizations: those based on the formal language **Z**, e.g. Clark, Evans and Kent (2001) and Saeki (2000) and those using first order predicate logic, e.g. Berardi, Cali, Calvanese and de Giacomo (2005), of which we chose the latter due to its concise presentation.

We start the formalization with the assumption that there is some kind of universe of consistent methods, called M. We assume furthermore, that these methods in M can be executed by project members, i.e. the method descriptions are available, complete, and consistent. The evolution of the method in a particular company can then be seen as a series of methods $m_1, m_2, \ldots, m_n \in M$. For reasoning about time we introduce the time dimension T. The set of method fragments is called F.

Definition 3.1 The mapping method: $T \rightarrow M$, where $m = \text{method}(t)$ means that the method $m \in M$ is the valid method at time t.

The methods change in the course of time, and this allows us to define the notion of snapshot of a method.

Definition 3.2 A *method adaptation time* is a point of time where the method has been adapted. Let T be the set of method adaptation times, i.e. $T = \{t_1, t_2, t_3, \ldots, t_n\}$ such that $\forall i \; \forall t$: $\text{method}(t_i) = \text{method}(t) \neq \text{method}(t_{i+1})$.

Definition 3.3 A *method snapshot* is a method $m \in M$ that was valid at a particular time, i.e. $\exists t_i \in T$; $m = \text{method}(t_i)$.

Definition 3.4 A *method evolution* is a set $S \subseteq M$ consisting of the method snapshots, i.e. $S = \{\text{method}(t_i) | \; t_i \in \check{T}\}$. So S is the set of methods that have been valid in the course of time.

We are now able to define method increments. As in common method engineering practices a method is seen as being composed of method fragments or method chunks (Brinkkemper, 1996; Ralyté & Rolland, 2001). Such a method is consistently created using well-formedness rules of process composition and deliverable configuration. These rules are not elaborated here, as they can be found in Brinkkemper, Saeki and Harmsen (1999).

Definition 3.5 The predicate contains: F x S : contains(f,s) ≡ fragment f is contained in snapshot s.

Then we can define an method increment as a method fragment that is part of method(t_i) but not in method(t_{i-1}).

Definition 3.6 A *method increment* is a method fragment $f \in F$ such that $\exists i$ contains(f,method(t_i)) $\wedge \neg$contains(f,method(t_{i-1}))

This means that the method increments are a collection of method fragments that have been introduced in the method during the method adaptations between t_i and t_{i-1}. In the following section we will then formalize the various types of increments

5.3.2 Formalization of method increments

In Figure 5-3 the meta-meta model of PDD is given, denoted in (again!) a UML Class diagram.

The meta-meta model is a simplified view of the full UML definition of Class diagrams and Activity diagrams (Object Management Group, 2004) with special emphasis on the adaptations discussed in Section 5.2.2 and the definitions in 5.3.1. Figure 5-3 shows that a method consists of method fragments, that we distinguish as process fragments for the process part of a method and deliverable fragments similarly. Note that the creation of deliverables is modeled in the association *edits* between Activities and Concepts.

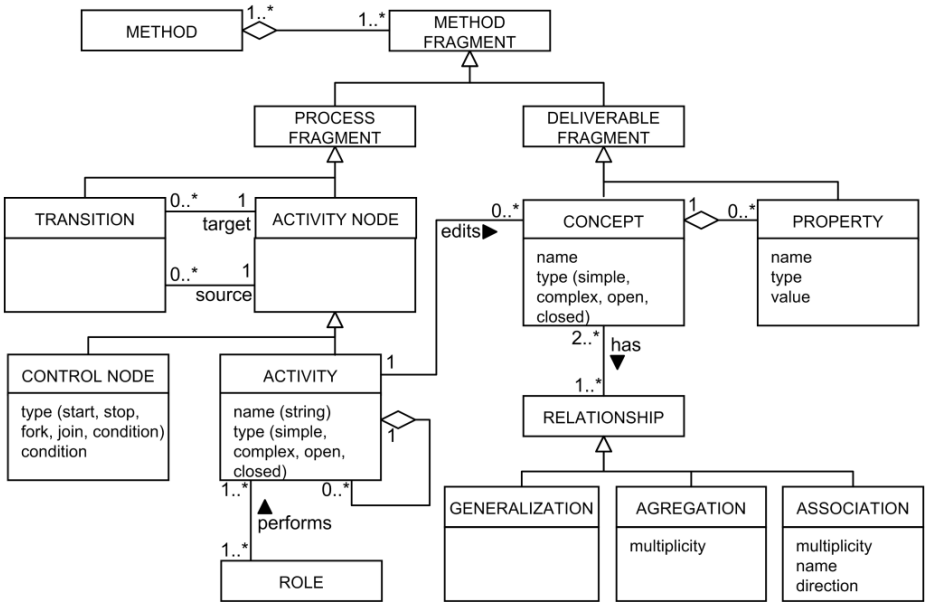


Figure 5-3. Meta-meta model of PDD

The structure of the meta-meta-model and the earlier case studies (van de Weerd, Versendaal & Brinkkemper, 2006) to method evolution revealed that 18 elementary increment types can be distinguished:

- *insertion* of a concept, property, relationship, activity node, transition, role
- *modification* of a concept, property, relationship, activity node, transition, role
- *deletion* of a concept, property, relationship, activity node, transition, role

The complete method increments from one snapshot to another can then be seen as a composition of elementary increment types.

The UML formalization of Berardi, Cali, Calvanese and de Giacomo (2005) postulates the existence of unary predicates for each class in a class diagram, e.g. concept(*c*) means that *c* is a concept in the model. However, in our research we require evolution of methods over the various snapshots, so we enhance these unary predicates to binary predicates with the method as an additional parameter. So concept(*c*,*m*) means that *c* is a concept in the method *m*. Method increments can now be defined as polymorphic mappings on the set of method fragments and methods.

Definition 3.7 The mapping insert: F x M → M: insert(f,m_1) = m_2 means that the method fragment f has been inserted in the method m_1 resulting into method m_2.

Definition 3.8 The mapping modify: F x F x M → M: modify(f_1,f_2,m_1) = m_2 means that the method fragment f_1 in the method m_1 has been modified to the fragment f_2 in method m_2.

Definition 3.9 The mapping delete: F x M → M: delete(f,m_1) = m_2 means that the method fragment f has been deleted from the method m_1 resulting into method m_2.

The rules for the elementary increments can then be formulated. For the sake of brevity we list the rules for the insertion of concepts and properties. Both rules are illustrated with an example that is taken from the increment example in Section 5.4.3.

Rule 3.1 Insertion of concepts:

$$\text{insert}(c,m_i) = m_{i+1} \Rightarrow \neg\text{concept}(c,m_i) \wedge \text{concept}(c,m_{i+1})$$

Rule 3.1 states that when a concept c has been inserted into method m_i to get method m_{i+1}, then c is not a concept present in m_i, and is present as concept in m_{i+1}. So, for instance:

insert(RELEASE TABLE,BaanIncr2) = BaanIncr3 ⇒ ¬concept(RELEASE TABLE,BaanIncr2) ∧ concept(RELEASE TABLE,BaanIncr3)

This means that when the concept RELEASE TABLE is inserted into BaanIncr2 resulting into BaanIncr3, then RELEASE TABLE is not a concept present in BaanIncr2 and is present as concept in BaanIncr3.

Rule 3.2 Insertion of properties:

$$\text{insert}(p,m_i) = m_{i+1} \wedge \text{property}(p,m_{i+1}) \Rightarrow [\forall c: \text{concept}(c,m_i) \wedge \neg\text{contains}(p,c)] \wedge [\exists 1 c: \text{concept}(c,m_{i+1}) \wedge \text{contains}(p,c)]$$

Rule 3.2 tells that when property p is inserted into snapshot m_i resulting into snapshot m_{i+1}, then p is not a property of any concept in m_i and there is just one concept in m_{i+1} of which p is the property. So, for instance:

insert(topic,BaanIncr2) = BaanIncr3 ∧ property(topic,BaanIncr3) ⇒
[∀*c*:concept(REQUIREMENT,BaanIncr2) ∧ contains(topic,REQUIREMENT)]
∧ [∃1*c*:concept(REQUIREMENT,BaanIncr3) ∧
contains(topic,REQUIREMENT)]

This means that when the property topic is inserted into snapshot BaanIncr2, resulting into BaanIncr3, then topic is not a property of any concept in BaanIncr2 and there is just one concept, namely REQUIREMENT, in BaanIncr3 of which is topic the property.

Analogously, rules for the other 16 elementary method increments can be formulated, while taking the method assembly rules in Brinkkemper, Saeki and Harmsen (1999) into account. Based on our earlier work on method assembly this formalization is extremely straightforward and will support the construction of the PSKI currently under development.

5.3.3 Root cause analysis for product software

Based on the general Root Cause Map (RCM) (Root Cause Analysis Handbook, 1999), the reference framework for software product management (van de Weerd, Brinkkemper, Nieuwenhuis, Versendaal & Bijlsma, 2006), and the HRM case study (van de Weerd, Versendaal & Brinkkemper, 2006), we are able to construct an initial RCM for SPM, as is depicted in Figure 5-4.

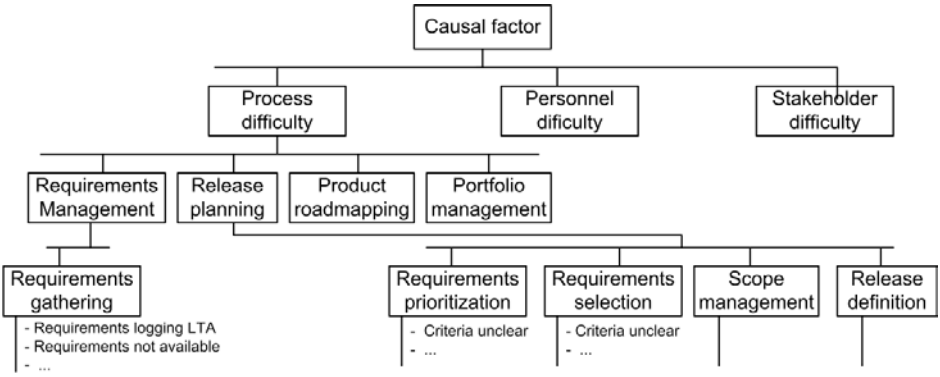


Figure 5-4. The explorative case root-cause map

During the interviews conducted in the HRM case study, two major process difficulties for requirements management were recognized:

- Customers do not see that their required features and software improvement wishes are implemented in new releases.
- The company finds its requirements gathering process for new features not productive.

When we apply RCA to these process difficulties, we identify a number of causal factors: To communicate a suggestion for improvement, a customer can contact the sales representative; in some cases the sales representative replies that suggestions should be posted to the helpdesk; in other cases the sales representative forwards the suggestion to the helpdesk; and some suggestions are not logged at all.

As for the second process difficulty, when a new release is defined, the helpdesk, the development manager and the software engineers are consulted. Rather arbitrary, but fitting a defined planning schedule, the development of a new release is triggered. Consequently, we identify three causal factors:

C1. Customers have difficulty in making their wishes known
C2. Customer requirements are not registered effectively
C3. Scoping of releases is rather arbitrary

The following root causes can be identified (indicated are the corresponding causal factors):

R1. Requirement logging is less than adequate (LTA) (root cause for C1 & C2)
R2. Requirements are not available (root cause for C3)
R3. Criteria for requirements prioritization are unclear (root cause for C1 & C3)
R4. Criteria for requirements selection are unclear (root cause for C3)

Van de Weerd, Versendaal and Brinkkemper (2007) describe a threefold solution for the two major process difficulties(A & B):

S1. Introduction of a separate activity for receiving and logging new requirements;
S2. Introduction of a wish list (requirements database) with wishes (requirements) containing a priority attribute;
S3. Introduction of a separate activity for prioritizing wishes.

When we map this process on the PSKI, this threefold solution would be described in a process advice, containing process descriptions, templates and examples. Note that RCA was not the basis for the solution finding at the HRM case study. However, if we do take into account the RCA and the resulting root causes we find that solution S1 addresses R1 and R2, solution S2 addresses R2 and partly R3, solution S3 addresses R3. Note that R4 has not been properly

addressed in the solution. We conclude that in the HRM case study, RCA was a useful approach for finding process improvements alternatives. This will be further validated in Section 5.4.4.

5.4 ERP case study

We carried out a case study at Infor Global Solutions (specifically the former Baan company business unit), a vendor of ERP (Enterprise Resource Planning) software (see for example Natt och Dag, Gervasi, Brinkkemper and Regnell (2004)). The goal of the ERP case study is to validate the increment types defined in Section 5.3.2 and the root-cause map in Section 5.3.3. In 1978, Baan was established as a book-keeping consulting company. Over the years, the company changed from a consultant company to a software developer for businesses. Baan was quoted on the Nasdaq stock exchange as an independent company from 1995 to 2000.

5.4.1 Case study design

Different sources are used to collect information. Firstly, several interviews have been conducted with six former employees of Baan. Two explorative 3-hour interviews were conducted with the Process Engineer of Baan. Based on this interview, the method evolution between 1997 and 2002 was modeled. This information was cross-checked by conducting 2-hour follow-up interviews with five other employees of Baan, consisting of two former (Senior) Product Managers, a Director ERP Development, a Manager ERP Product Ownership and a Software Engineering Process Group Manager for Baan Development. In these interviews, also the snapshots of 1994, 1996, 2003, 2004 and 2006 were identified and modeled.

Secondly, a document study was carried out. Documentation provided by the Process Engineer was used to complement and validate the results from the interviews. This documentation consisted of process descriptions, templates and examples of methods and work products used at Baan in the period 1997 until 2006. From the period before 1997 no documentation was available. We focused on the following case study questions, related to software product management:

- Which snapshots can you identify in the method evolution?
- Which methods were used per stage? Which activities can be distinguished?
- Which deliverables resulted from these methods?

– Which process difficulties arose in this stage? Why was an increment needed?

With the information gathered in the case study, we modeled 14 snapshots in PDDs, each representing a method that was used in a particular moment in time (van de Weerd, Brinkkemper & Versendaal, 2006).

5.4.2 Method snapshots

We analyzed 14 snapshots (listed in Table 5-1) of the evolution of the software development process at Baan, with emphasis on product management activities. The time period that is covered in the ERP case study ranges from 1994 to 2006.

Table 5-1. Overview of method increments at Baan

#	**Increment**	**Date**
0	Introduction requirements document	1994
1	Introduction design document	1996
2	Introduction version definition	1998, May
3	Introduction conceptual solution	1998, November
4	Introduction requirements database, division market and	1999, May
5	Introduction tracing sheet	1999, July
6	Introduction product definition	2000, March
7	Introduction customer commitment process	2000, April
8	Introduction enhancement request process	2000, May
9	Introduction roadmap process	2000, September
10	Introduction process metrics	2002, August
11	Removal of product families & customer commitment	2003, May
12	Introduction customer voting process	2004, November
13	Introduction master planning	2006, October

Note that, although some method increments entail the removal of a method fragment, we still describe them as increments, as described in Section 5.2.1. In the following section, one of these increments, namely the increment between snapshot 2 and 3, is further elaborated on. The other increments are described in van de Weerd, Brinkkemper and Versendaal (2006).

5.4.3 Increment example: Introduction of the conceptual solution

In Figure 5-5, increment # 2 of the ERP case study is visualized. Looking at the process-side of the diagram, we can distinguish one main activity, i.e. Requirements, and three sub-activities.

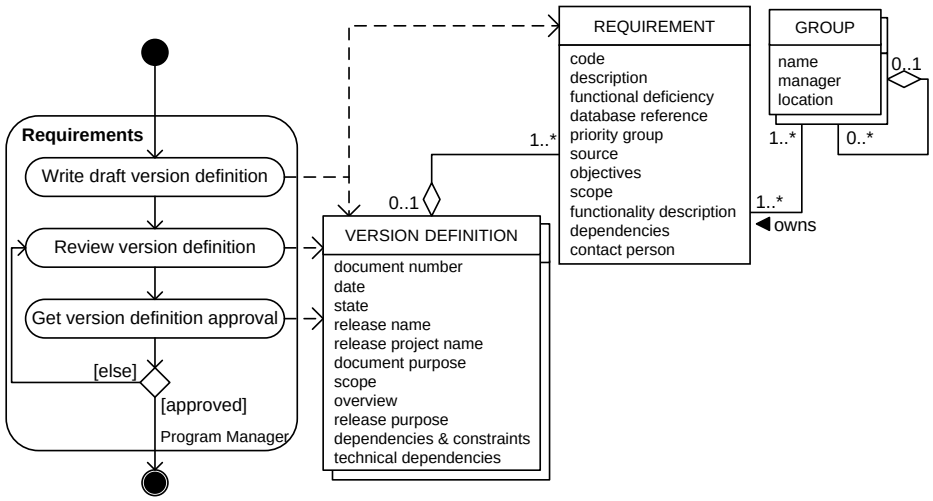


Figure 5-5. Snapshot of increment #2

The first sub-activity, Write draft version definition, results in the concepts VERSION DEFINITION and REQUIREMENT. The latter is connected to VERSION DEFINITION by means of aggregation. Both have a number of attributes, and finally, a REQUIREMENT is owned by a GROUP, that has the responsibility for this REQUIREMENT. The next sub-activity is to review the VERSION DEFINITION. If the approval is obtained, the next activity can be started; otherwise the VERSION DEFINITION has to be reviewed again.

In Figure 5-6, increment #3 is visualized. In this snapshot, one extra activity is included. Note, however, that this activity is open, i.e. this activity contains further sub activities that are elaborated elsewhere. Due to space limitations, the elaboration on this activity is not included in this paper.

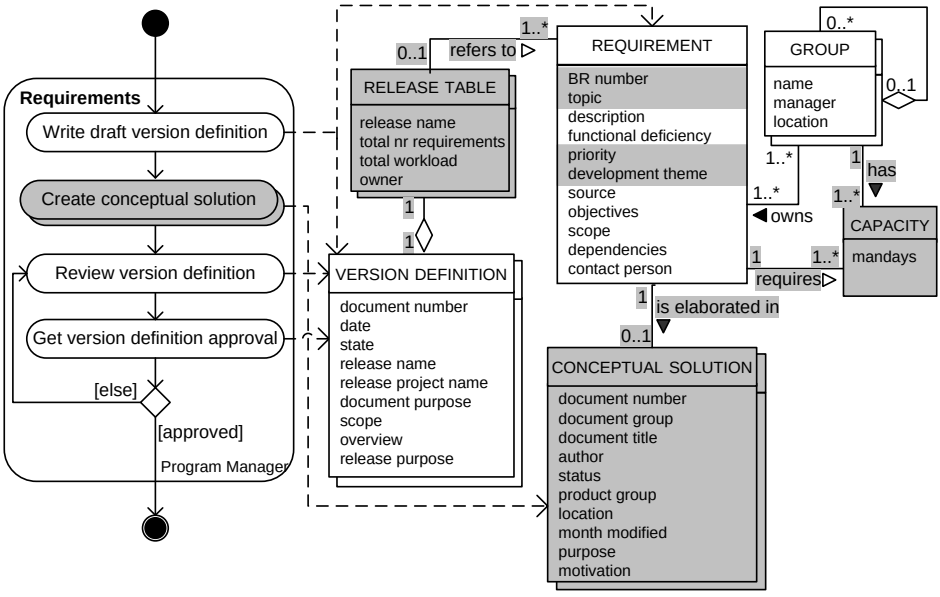


Figure 5-6. Snapshot of increment #3

5.4.4 Root cause analysis of method increments

In increment 3 (Figure 5-6) we distinguish the following increment types, based on the formalization in Section 5.3.2:

I1. Insertion of an activity node, i.e. Create conceptual solution
I2. Insertion of a concept, i.e. RELEASE TABLE, CONCEPTUAL SOLUTION and CAPACITY
I3. Insertion of a property, i.e. the properties added to REQUIREMENT
I4. Insertion of a relationship, i.e. the relationships connecting the introduced concepts to the existing concepts

Now we focus on RCA. The increments are included to solve one or more problems. Based on the interviews, several process needs were identified in the snapshot of increment #2. The most important ones were:

– Development managers find it hard if not impossible to determine a VERSION DEFINITION that is feasible with available resources, and consequently makes sub-optimal scoping decisions. In detail: signals from the market, as well as from internal stakeholders, indicate that a new release should be developed. The development managers ask the program managers and architects to establish the version definition for the different software

modules. The program managers and architects collect, with some difficulty, the features and requirement from different sources. They select a set of features to be developed according to their own opinions. The program managers and architects discuss the draft version definition with the development managers, and make changes to the selection of features.

– Software engineers find it hard to read the VERSION DEFINITION in order to build what is requested, and consequently do not build the precise features that were intended to be build. In detail: in the version definition each new software product REQUIREMENT is elaborated by the program manager and/or product architect. They describe dependencies with other REQUIREMENTS in the text associated with a requirement. The software engineers read the (often badly written) requirements, interpret requirement texts, possibly asking their program managers and architects for explanations. Subsequently, the requirements are built in the software product.

We identify the following causal factors:

C1. Requirement collection is difficult
C2. Text elaborations of requirements have different authors
C3. Requirements dependency descriptions are unstructured
C4. Interpretation of requirement is ambiguous

If we apply the earlier constructed root cause map for product software to this particular increment we choose to extend it accordingly in order to address all identified causal factors (see Figure 5-7). Note that, although we had to extend the root cause map with the (bold) root causes, it fits the constructed structure as derived in Section 5.3 very well.

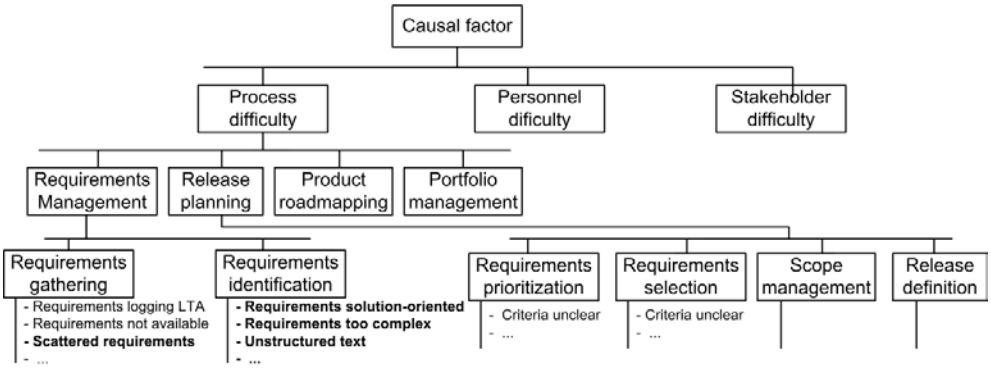


Figure 5-7. Baan increment extended root cause map for product software

The root causes of the four identified causal factors are fourfold:

R1. Requirements are scattered throughout the company in different documents (root cause for C1 & C2)
R2. Some requirements are written in a solution-oriented way (root cause for C2 & C4)
R3. Requirements are too complex (root cause for C4)
R4. Requirements are written in unstructured text (root cause for C3 & C4)

R2 and R3 led to the introduction of the CONCEPTUAL SOLUTION in increment #3. This document was used to write a solution on conceptual level for the particular REQUIREMENT. In this way, solution-oriented texts are also kept out the requirements themselves. R4 partly led to the decomposition of the VERSION DEFINITION and REQUIREMENTS. A RELEASE TABLE is used, in which information on the separate REQUIREMENTS is summarized. No (full) solution is implemented for R1 and R4. These are taken into account in the subsequent increment, which is described in van de Weerd, Brinkkemper and Versendaal (2006).

We note that the RCM for SPM has been extended on the lowest level, but that higher levels were untouched, indicating that for two different companies, the RCM for SPM is a useful tool. We conclude that RCA can be used in software development improvements (as in el Emam, Melo and Drouin (1997)) and more specifically software product management. The root causes showed in the RCM for SPM provide means for the PSKI to determine process improvement alternatives.

5.4.5 Validity threats

In exploratory research, three types of validity are important (Yin, 2003). Firstly, construct validity concerns the validity of the research method. We satisfy this type of validity by using multiple sources of data (interviewees and documents) and by maintaining a chain of evidence. Furthermore, we had key informants review the draft case study report. Secondly, the external validity concerns the domain to which the results can be generalized. We carried out the case study in the software product management domain in PS companies. The same protocol is followed as in earlier case studies in PS companies. Finally, to guarantee the reliability of the case study, all information should be recorded. This is done by maintaining a case study database which contains all relevant information used in the case study. This case study database consists of interview notes, documentation and process-data diagrams of all modelled methods.

5.5 Conclusion

By presenting a formal approach to incremental process improvement, we provided PS companies with an instrument to improve their software product management methods in an evolutionary way. Firstly, we formalized the method increments that occur during method evolution. Doing this provided insight in the evolution process, which can be used when assembling a method advice. Secondly, we presented an approach for the structural analysis of process needs, by using root cause analysis. By applying this analysis in a case study, we found that this approach and the corresponding root cause map can be of great value in the support of incremental method evolution.

Currently, we are working on the realization of the PSKI. We aim to further integrate the root cause analysis approach in the PSKI in order to map root causes to maturity capabilities and method fragments. The formalization of method increments is used to implement assembly rules. In the future, we plan to fill the method base with situational factors, method fragments and assembly rules. Finally, we plan to test the PSKI at PS companies of different sizes and in different sectors, in order to test the mapping between situational factors, maturity capabilities and method fragments

.

CHAPTER 6

A retrospective case study in incremental method evolution

Company growth in a global setting causes challenges in the adaptation and maintenance of an organization's methods. In this paper, we will analyze incremental method evolution in software product management in a global environment. We validate a method increment approach, based on method engineering principles, by applying it to a retrospective case study conducted at a large ERP vendor. The results show that the method increment types cover all increments that were found in the case study. Also, we identified the following lessons learned for company growth in a global software product management context: method increment drivers, such as the change of business strategy, vary during evolution; a shared infrastructure is critical for roll-out; small increments facilitate gradual process improvement; and global involvement is critical. We then claim that method increments enable software companies to accommodate evolutionary adaptations of development process in agreement with the overall company expansion.[1]

6.1 Introduction

The software market has made a shift from developing customized software to developing software as a standard product (Xu & Brinkkemper, 2007). Many of these product software companies face difficulties in their product development processes (Cusumano & Selby, 1995; Cusumano, 2004). An important solution to these problems is to implement a good software product management function. Ebert (2007) describes an empirical study with data from 178 industry projects that shows that time to market, schedule adherence and handover quality all improve with the strengthening of a coherent product management role.

Another trend that can be seen in many, especially globally operating, product software companies is company growth. Bryson (1997) argues that the

[1] This work is conditionally accepted for publication in the Journal of Information & Software Technology.

growth of a company is associated with the rising information intensiveness of production. A rapidly expanding company needs a reorganization of its structural systems (Kwestel, Preston & Plaster, 1998). This also holds for the processes and methods that are used. Von Krogh and Cusumano (2001) emphasize that to make a strategy for fast growth work, the structure and processes should be changed in a way that lets them acquire or create specific knowledge about new technologies, customers and industries. Also, strategic investments need to be made in a support infrastructure (Stalk, Evans & Shulman, 2001). Especially in globally operating organizations, it is important to focus on these issues, since many risks are associated with the assumed benefits (Eoin, Holmstrom, Ågerfalk & Fitzgerald, 2006), such as more overhead in communication, coordination and control, and a reduced collaborative time window due to the temporal difference.

To control company growth, organizations use process improvement methods, like SPICE (el Emam, Drouin & Melo, 1998) and CMM (Paulk et al., 1995), to implement new and better practices. These practices can be implemented in different ways and can be evolutionary or revolutionary in nature. In this research, we focus on an evolutionary approach instead of a revolutionary approach, for several reasons: a) it is a fundamental way to reduce risk on complex improvement projects (Krzanik & Simila, 1997); and b) we observe in practice that methods evolve (van de Weerd, Versendaal & Brinkkemper, 2006). This evolutionary approach has been subject of research in various scientific studies. Studies have been carried out to find the best approach to instigate a process improvement (Richardson & Ryan, 2001; Stelzer & Mellis, 1998) and research has been done to the key success factors that influence software process improvement (Rainer & Hall, 2002). However, in 2002, it was estimated that still 70% of software process improvement projects failed (Software Engineering Institute, 2002). Debou and Kuntzmann-Combelles (2000) underline the necessity to link any software process improvement program to business goals.

To summarize the problem: Product software companies need to implement software product management practices, but at the same time, need to improve their processes to stay in pace with the growth of their organization. Since many software process improvement projects fail, we want to get more insight in this process.

A retrospective case study at a large ERP (Enterprise Resource Planning) vendor is carried out to investigate the method increments that took place over a period of twelve years and eleven countries. We first analyze the general *method increments*, as introduced 0, which are collections of method fragments

that have been changed (inserted, modified, deleted) during a certain method adaptation. Secondly, we explore what the drivers for each method increment are, i.e. what circumstances or events in the organization triggered the method increments. This leads us to the following research question:

Which method increment types occur in incremental method evolution, and which general increment drivers can be identified?

In the next section, we first explain our research approach, including the meta-modeling technique and formalization of the method increments. In section 3, we describe the design of our retrospective case study, as well as the possible validity threats. Section 4 describes the results of the case study, as well as the analysis in which the method increment types are compared with the found method increments in the case study. Then, in section 5, we describe the lessons learned from the case study. Section 6 presents an overview of related work. Finally, in section 7, we describe our conclusions and future research.

6.2 Research approach

In this paper, we report about a retrospective case study. We chose this type of research for several reasons. Firstly, an exploratory case study makes it possible to get an in-depth understanding of a contemporary phenomenon where the investigator has little control over event (Yin, 2003). Secondly, the retrospective nature of the case study made it possible to investigate the changes in an organization over a period of twelve years; a time period which is almost impossible to investigate by a regular longitudinal research.

Our case study starts with retrospective interviews with several product managers and a document study on the actual deliverables of the used methods in the given time period. Examples of these deliverables are release plans, requirement documents and roadmaps. Based on the interviews and document study, we model the method increments in process-deliverable diagrams (PDDs), which are explained in Section 6.2.2, and analyze them. Finally, we identify how the method increments found in the case study correspond to our earlier identified elementary method increment types.

In the following sections, we will first describe our case study design. Then, in Section 6.2.2, we describe the meta-modeling technique we used to model the method increments. We conclude by describing the threats to validity.

6.2.1 Case study design

Different sources are used to collect information. Firstly, several interviews have been conducted with six former employees of the case company. The choice of these interviewees was based on their knowledge (they all played key roles in the Baan processes under investigation) and their availability. Two explorative 3-hour interviews were conducted with the Process Engineer, who was responsible for the process improvement roll-out in requirement management. Based on these interviews, the method evolution between 1994 and 2002 was modeled. This information was cross-checked by conducting 2-hour follow-up interviews with five other employees of Baan, consisting of two former (Senior) Product Managers, a Director ERP Development, a Manager ERP Product Ownership and a Software Engineering Process Group (SEPG) Manager. In these interviews, also the method snapshots of 1994, 1996, 2003, 2004 and 2006 were identified and modeled.

Secondly, a document study was carried out. Documentation provided by the Process Engineer was used to complement and validate the results from the interviews. This documentation consisted of process descriptions, templates and examples of methods and work products used in the period 1997 until 2006. In total, 26 documents have been analyzed. From the period before 1997 no documentation was available. We focused on the following case study questions, related to software product management:

- Which snapshots can you identify in the method evolution?
- Which methods were used per stage? Which activities can be distinguished?
- Which deliverables resulted from these methods?
- Which process difficulties arose in this stage?
- Why was an increment needed?

6.2.2 Modeling method increments

For the analysis of method increments, we use process-deliverable diagrams (PDDs), a meta-modeling technique that is based on UML activity diagrams and UML class diagrams (Saeki, 2003; van de Weerd, Brinkkemper, Souer & Versendaal, 2006). The resulting PDDs model the processes on the left-hand side and deliverables on the right-hand side (see Figure 6-1).

We follow standard UML (Object Management Group, 2004) conventions, but some minor adjustments have been made for modeling development processes. Firstly, deliverables can be simple or compound. Simple deliverables do not contain any sub-deliverables and are visualized with a rectangle.

Compound deliverables contain one or more sub-deliverables. Compound deliverables can be open, visualized with an open shadow, to indicate that it contains sub-deliverables. The sub-deliverables can be shown in the same diagram, by using aggregation, or in another diagram (for example for space saving). Closed compound deliverables, visualized with a closed shadow, indicate that sub-deliverables exist, but are not relevant in this context. Similarly, open and closed activities are used in the diagram. The dotted arrows indicate which deliverables result from the activities. More details on this modeling technique can be found in van de Weerd, Brinkkemper, Souer and Versendaal (2006).

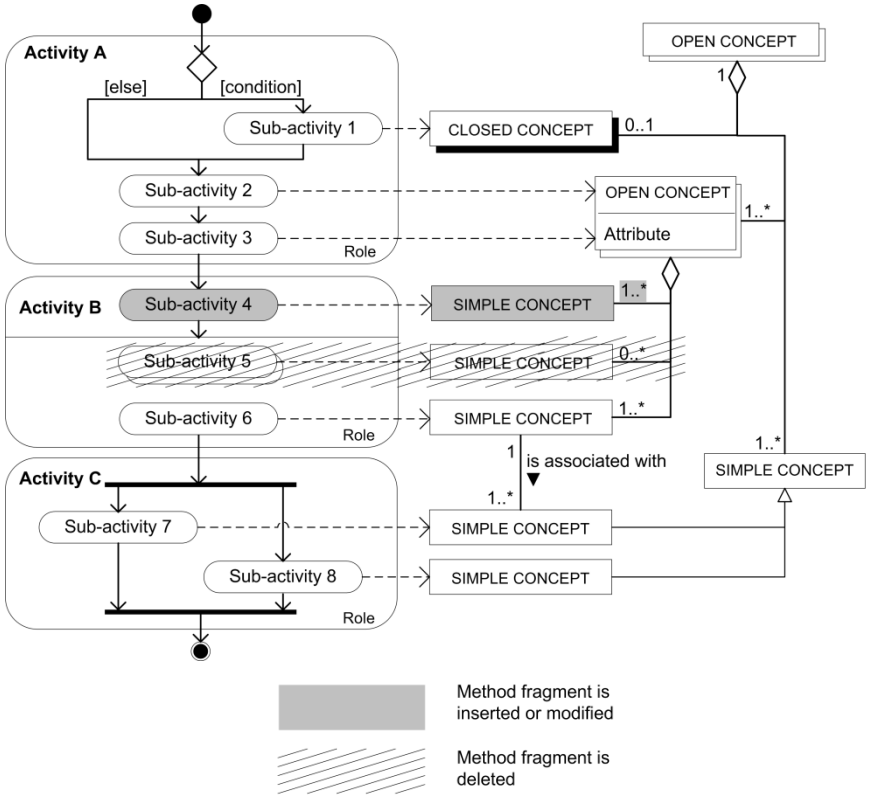


Figure 6-1. Process-deliverable diagram

In van de Weerd, Versendaal and Brinkkemper (2006) and van de Weerd, Brinkkemper and Versendaal (2007), we have defined the notions of method evolution, snapshot and method increment. The PDD, visualized in Figure 6-1, is called a *snapshot*, a model of the method as it was at a certain moment in

time. The evolution of a method over time exists of a number of these snapshots. By comparing snapshots, method increments can be analyzed. In Figure 6-1, we marked sub-activity 4 and its corresponding concept. We use this marking to show that this method fragment is inserted or modified compared to a snapshot earlier in time. Sub-activity 5 and its corresponding concept are also marked. This indicates that this method fragment is deleted, compared to the preceding snapshot.

Based on the meta-meta model of PDDs we identified a list of 18 elementary increment types:

- *insertion* of a concept, property, relationship, activity node, transition, role
- *modification* of a concept, property, relationship, activity node, transition, role
- *deletion* of a concept, property, relationship, activity node, transition, role
- These elementary method increment types are used to decompose and analyze the method increments that are found in the retrospective case study.

6.2.3 Threats to validity

In exploratory research, the quality of the methodology should be judged on the basis of three types of validity (Yin, 2003). *Construct validity* concerns the validity of the research method. We used multiple sources of data (interviewees and documents) to ensure that problems related to subjectivity and bias of data were avoided. Furthermore, we had key informants review the draft case study report.

Secondly, the *external validity* concerns the domain to which the results can be generalized. We carried out the case study in the software product management domain in a product software company. We cannot say whether our findings apply to *all* globally operating product software companies. However, we are convinced that this study is a first valuable exploration in the field of incremental method evolution in software product management.

Finally, the *reliability* of the case study is concerned with demonstrating that the results of the study can be replicated. This threat is mitigated by maintaining a case study database that contains all the relevant information used in the case study. This case study database consists of interview notes, documentation and process-deliverable diagrams of all modeled methods. In addition, we used a case study protocol that has been used in earlier case studies to method

increments in product software companies (van de Weerd, Versendaal & Brinkkemper, 2006).

A few remarks should be made on the retrospective nature of the case study. A problem relating to this type of study is hindsight bias, which refers to the ability of people to reconstruct prior probabilities for an event after it has occurred (Leary, 1982). In this study, it means that the interviewees tend to regard the drivers that lead to the method increments as having been fairly predictable ("I knew it from the beginning…"), although this was in reality not the case. We have tried to reduce this hindsight bias by using multiple interviewees and by backing their comments by documentation such as process descriptions and examples from the case study company.

6.3 Case study

We carried out a case study at Infor Global Solutions (specifically the former Baan company business unit, hereafter called 'Baan'), a vendor of ERP software. In 1978, Baan was established as a book-keeping consulting company. Over the years, the company changed from a consultant company to a software vendor for businesses. Baan was quoted on the Nasdaq stock exchange as an independent company from 1995 to 2000.The time period that is covered in the case study ranges from 1994 to 2006.

We analyzed the evolution of the software development process at Baan, with emphasis on product management activities. During this period, the development centers of Baan were located in eleven different countries in four continents, as is depicted in Figure 6-2.

All development centers used a shared infrastructure, consisting of a requirements database, online development method, and the document management systems for storing all development documents (Brinkkemper, 2000). Around the years 1998 – 2001 about 1500 engineers, managers and support staff were employed at Baan R&D. Later, the size was downscaled, which also implied that we had to include former employees in the case study.

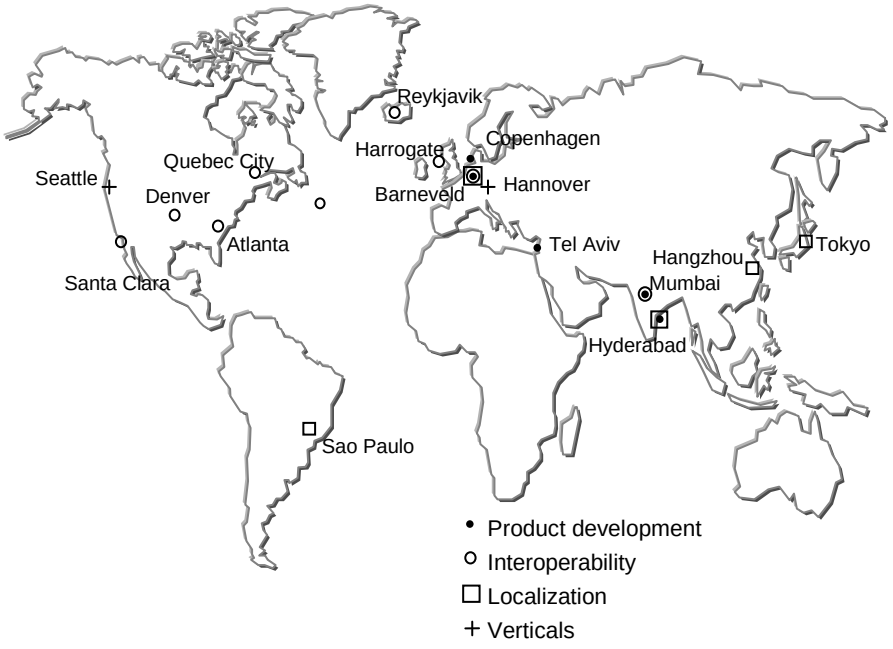


Figure 6-2. Global distribution of R&D locations (situation August 2000)

The different locations had different main functions: *product development*, for developing Baan's ERP product line and the already integrated products of acquired companies; *interoperability*, to handle the integration of products of acquired companies with the main ERP product; *localization*, for localizing the products to international markets; and *verticals*, which focused on a specific market segment, e.g. automotive and aerospace. Note that the acquired companies also perform independent product development next to the interoperability. This is omitted in Figure 6-2, as it is not relevant for the discussion. Also, sales and marketing, implementation services, and maintenance centers are not involved in this study, as these are not responsible for product management (van de Weerd, Brinkkemper, Nieuwenhuis, Versendaal & Bijlsma, 2006).

Table 6-1. Overview of method increments at Baan

#	**Increment**	**Date**	**Reason for increment**	**Initiating locations**
1	Introduction requirements document	1994	Development management	Netherlands

2	Introduction design document	1996	Development management	Netherlands
3	Introduction version definition	1998, May	Development management	Netherlands & US
4	Introduction conceptual solution	1998, Nov	Business management	Netherlands & India
5	Introduction requirements database, division market & business requirements, introduction of product families	1999, May	Development management	Netherlands & India
6	Introduction tracing sheet	1999, July	Certification	Netherlands, US & India
7	Introduction product definition	2000, Mar	Departmental interfacing	Netherlands
8	Introduction customer commitment process	2000, April	Business management	Netherlands & US
9	Introduction enhancement request process	2000, Sep	Departmental interfacing	Netherlands & India
10	Introduction roadmap process		Departmental interfacing	Netherlands
11	Introduction process metrics	2002, Aug	Certification	Netherlands & India
12	Removal of product families & customer commitment	2003, May	Business management	US
13	Introduction customer voting process	2004, Nov	Business management	US
14	Introduction master planning	2006, Oct	Development management	Netherlands

6.4 Description of method increments

In 1985, Baan developed its first standard product: a salary software package. However, no requirements or designs were used and no documentation was written. This caused much frustration in the development process; improvement was needed. Some people started working on a standard method and, finally, in 1988 the first version of the *Baan Development Method* was created. As a result, developers started to document their software. In the early nineties, Baan counted 400 employees and had offices in America and Canada. It was necessary to define clear processes and methods. A Software Process Improvement (SPI) project was started, based on the Capability Maturity Model (CMM). The CMM level was estimated on level 1, so improvement was

needed. In 1994, methods for software product management were defined for the first time. We use this as starting point for our analysis.

With the information gathered from the first interviews, we identified 14 method increments, as depicted in Table 6-1. For each method increment, we listed a description, date, and the countries that initiated the process change. Then, we analyzed the interviews and documents and modeled the 14 snapshots in PDDs, each representing a method that was used in a particular moment in time (van de Weerd, Brinkkemper & Versendaal, 2006). These snapshots were identified after a discussion with the interviewees. Based on the analysis, we also described the reason for every increment. In section 5, we will further elaborate on this.

In the following sections we will analyze 8 of the 14 method increments as depicted in Table 6-1. We choose these method increments in order to be able to show all increment types. The first six increments illustrate the principle of including new fragments, and the later ones illustrate the deletion and modification of fragments.

For each method increment, we first describe the circumstances that instigated the increment, as well as the actions that were taken. Secondly, we provide a snapshot of the increment (expressed in a PDD). Thirdly, we give an explanation of the method increment. For clarification, we describe activities between hyphens (e.g. Write requirements document) and deliverables in capitals (e.g. REQUIREMENTS DOCUMENT).

6.4.1 Increment 1: Documenting requirements

Before the snapshot depicted in Figure 6-3, the *Baan Development Method* was used as a standard method to develop and document software. However, this method was restricted to development and testing stages, and no attention was given to requirements engineering.

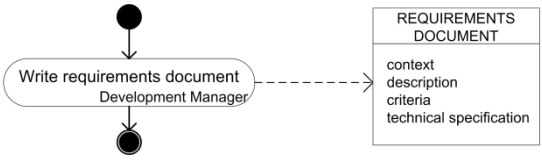


Figure 6-3. Snapshot of increment 1

In Figure 6-3, increment 1 of the Requirements stage at Baan is visualized. The requirements were described by the development manager in the

REQUIREMENTS DOCUMENT, in order to separate the requirements from the design. This REQUIREMENTS DOCUMENT typically consisted of two pages and described mainly functional requirements. Key questions that were answered in this document were: what's it about, what should it do, and what are the criteria? In addition, the Development Manager also described some high-level technical specification of the requirements of functionality.

6.4.2 Increment 2: Separating requirements and design

In increment 1, development managers were writing their technical specifications in the requirements document. Instead of describing the "*what*" question, the requirements document described the "*how*" question. This development driver caused the implementation of a method increment in which the requirements and design was being separated.

In Figure 6-4, increment 2 of the Requirements stage at Baan is visualized. We can divide this increment into two main adaptations. Firstly, the Technical specification is not described in the REQUIREMENTS DOCUMENT anymore. The technical specification is now described in another document, namely the DESIGN DOCUMENT. This is not modeled here, because we only focus on requirements management and release planning. The second change is the clear structure that has been introduced in the REQUIREMENTS DOCUMENT. In the REQUIREMENTS DOCUMENT, one or more REQUIREMENTS are described. Each REQUIREMENT has its own description and criteria.

Only one property in the concept REQUIREMENT DOCUMENT has been deleted. The two other properties have been modified; they are now properties of a newly inserted concept: REQUIREMENT. Finally, a relationship has been added: the aggregation between REQUIREMENT DOCUMENT and REQUIREMENT.

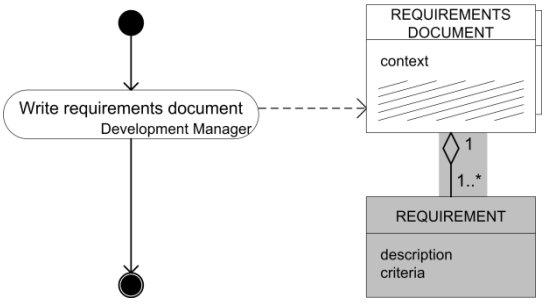


Figure 6-4. Snapshot of increment 2

6.4.3 Increment 3: Version definition

In increment 2, the requirements were documented in a requirements document. However, much information was missing in this document and it was not focused on the release of the new product version.

To solve these problems, a team of three developers in Santa Clara, US (cf. 6.5.3-3), developed the version definition, which contained more information concerning the release. Note that the term 'version definition' is adopted from Baan's organizational terminology. In most literature, the term 'release planning' is used. The requirements itself were also being documented more detailed, and, what's more, each requirement could now be assigned to a development group.

In Figure 6-5, increment 3 of the Requirements stage at Baan is visualized. Looking at the process-side of the diagram, we can distinguish two main activities, namely Requirements management and Version definition. The following method fragments have been altered: First, the role has been changed. Instead of the Development manager, the Program manager is now responsible for the requirements management and release planning. The activity Write requirements document is modified in Write version definition. Furthermore, four activity nodes have been added: three new activities: Gather requirements, Review version definition and Get version definition approval; and one branch. This branch comes after Get version definition approval. If this approval is obtained, the next activity can be started; otherwise the VERSION DEFINITION has to be reviewed again.

Along with these activity nodes, five new transitions are added to the method. At the deliverable side, several changes have been made as well: the concept REQUIREMENTS DOCUMENT is modified in VERSION DEFINITION, and multiple new properties have been added to this concept. Also, several new properties have been added to the existing concept REQUIREMENT. A new concept, GROUP, has been added along with two new constructions. One or more REQUIREMENTS are now owned by one or more GROUPS that have the responsibility for this REQUIREMENT. Finally, the relationship between REQUIREMENT and VERSION DEFINITION has been modified.

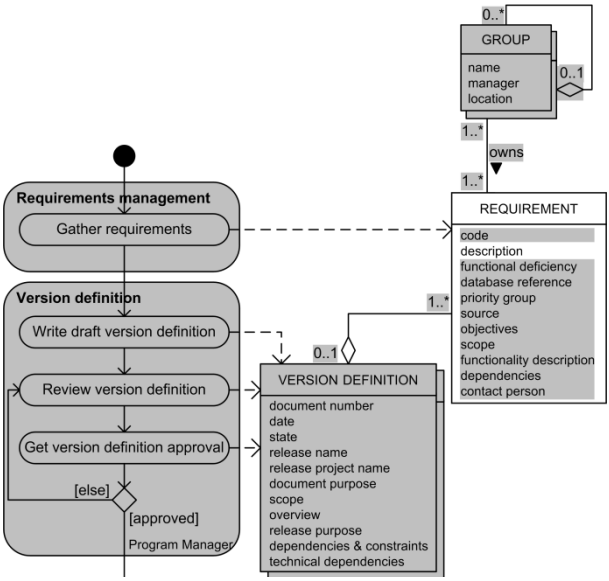


Figure 6-5. Snapshot of increment 3

Concurrently with this method increment, the development processes of Baan, called Baan Development Method (BDM), were documented and put on the corporate intranet to support the worldwide distributed R&D departments (Brinkkemper, 2000) (cf. 6.5.3-1). In 1998, the BDM website was built in standard HTML, containing instructions, examples and templates for deliverables.

6.4.4 Increment 4: Conceptual solution

Two main problems can be identified in method increment 3. Firstly, software engineers found it hard to read the version definition in order to build what is requested, and consequently did not build the precise features that were intended to be built. Also, the sales people found it hard to read the version definition and know exactly which new functionality was being built in the new product. Finally, it was difficult to store and trace requirements, since they were always part of a requirements document

Two main improvements were made to solve the problems described above. Firstly, the conceptual solution was developed in the development center of Hyderabad, India (cf. 6.5.3-3). This document was used for internal and external communication related to requirements. Towards the customer, the requirement

raiser, a conceptual solution provides guidance on the way in which requirements are elaborated and clarified in the perspective of the current Baan products. For the development department a conceptual solution provides input for design and realization of the concept.

After creating templates, instructions and examples, the method increment was first implemented in the main ERP development centers: Barneveld, Hyderabad and Mumbai. Later on, the acquired companies in Hannover, Quebec City, Tokyo and Santa Clara followed (cf. 6.5.3-4).

The second improvement concerned the separation of the requirements and the version definition. The version definition now only consisted of references to the requirements that are listed in the release table. In this way, requirements could exist without being part of a version definition and the version definition is readable again.

In method increment 4 (Figure 6-7), the following changes have been made. First, a new activity node has been inserted: the activity Create conceptual solution. This activity is open, i.e. it consists of several sub-activities. These sub-activities cannot be specified here due to space limitations. At the deliverable side, also several changes have been made. Three new concepts have been included. The RELEASE TABLE is as a reference table in the VERSION DEFINITION that refers to the REQUIREMENTS. In this way, the VERSION DEFINITION only has to include the references to the REQUIREMENTS instead of the entire descriptions. With the introduction of this concept, two relationships to VERSION DEFINITION and REQUIREMENT have further been inserted, and several attributes in REQUIREMENT have been modified. Secondly, the concept CAPACITY is inserted. With this concept the capacity of a group is being administrated. Along with this concept, two relationships to GROUP and REQUIREMENT have been inserted. Finally, the concept CONCEPTUAL SOLUTION is inserted, which elaborates on the REQUIREMENTS, indicated by an association.

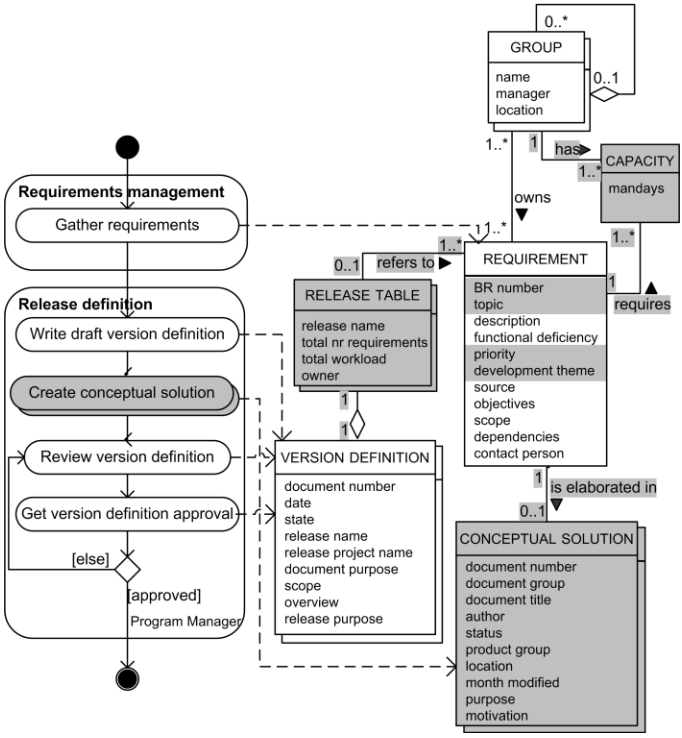


Figure 6-6. Snapshot of increment 4

6.4.5 Increment 5: Market and business requirements

In increment 4, no standard requirements gathering process was defined. External and internal stakeholders, such as customers, development, and consultants, sent their wishes to the Program manager, who added them to an Excel sheet as requirements. Secondly, the Program manager did not rewrite the requirements. Usually, they were described in the terminology of the customer, and often in one requirement multiple functionalities were described. At this time, every month hundreds of requirements were entering the company, so it was extremely difficult to keep the overview of the requirements database.

The solution to these problems that is implemented in increment 5 is twofold. The development center in the Netherlands developed a centralized requirements organization system in which a distinction was being made between market requirements and business requirements (cf. 6.5.3-1). Market requirements were requirements that are directly copied from the requirement raiser. In case a customer raised a certain requirement, it was documented as

market requirement in the terminology of the customer. Then, business requirements were written by the product manager, which could be a redefinition of one or multiple market requirements. More information in this process can be read in Natt och Dag, Gervasi, Brinkkemper and Regnell (2005). The second part of the solution consisted of the use of the common component framework. Since the acquisition rate was quite high in this period, and multiple products were being acquired, several products and product lines exist that shared functionality. By identifying common components, and relating that to the business requirements, the product managers tried to protect the company from double work.

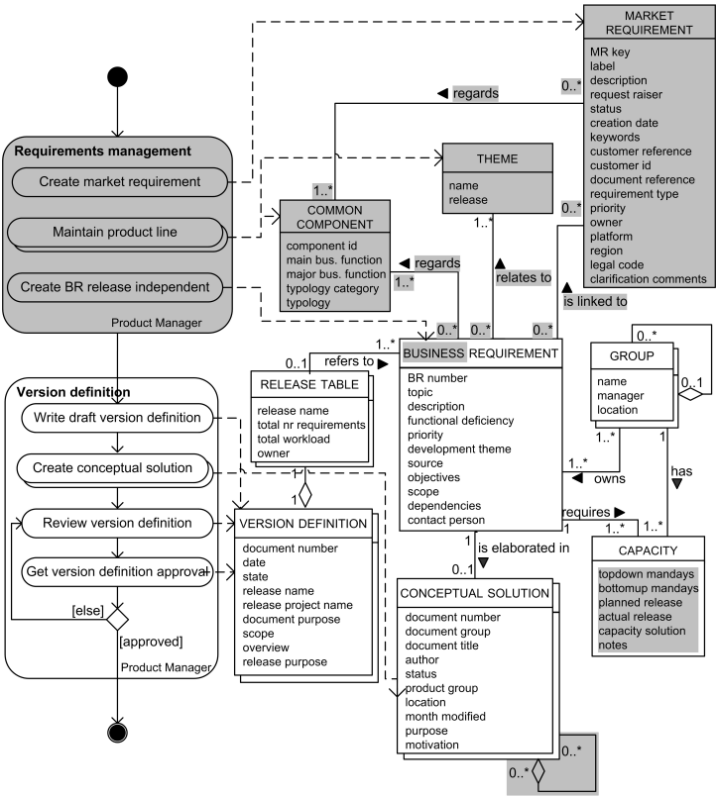


Figure 6-7. Snapshot of increment 5

As depicted in Figure 6-7, several method fragments have been added in method increment 5. First, three new activities have been added, grouped in the Requirements management main activity. The three activities are Create market requirement, in which a new MARKET REQUIREMENT, in the customer's

terminology, is added to the requirements database; Maintain product line, in which COMMON COMPONENTS, or core assets, are identified and managed; and Create BR release independent, in which BUSINESS REQUIREMENTS are created, based on the stored MARKET REQUIREMENTS. Together with the introduction of the three activities, also several things changed at the deliverable side. REQUIREMENT is modified in BUSINESS REQUIREMENT. The concept MARKET REQUIREMENT is inserted to distinguish customer wishes from actual business or product requirements. Zero or more MARKET REQUIREMENTS are related to zero or more BUSINESS REQUIREMENTS. COMMON COMPONENT was added, with relationships to MARKET REQUIREMENT and BUSINESS REQUIREMENT. Finally THEME is inserted, which is only related to BUSINESS REQUIREMENT. Two extra small changes have been made, namely the introduction of an aggregation relationship for CONCEPTUAL SOLUTION and several extra attributes in CAPACITY.

In Figure 6-8, a screenshot of the requirements database is illustrated. The picture shows a business requirement that is linked to several market requirements.

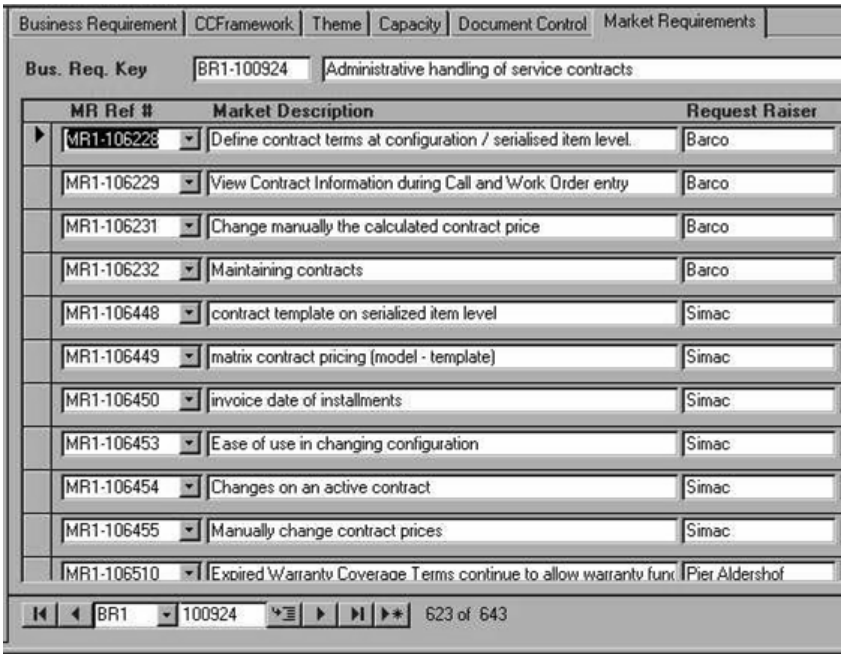


Figure 6-8. Baan requirements database screenshot

6.4.6 Increment 6: Tracing sheet

Increment 6 has a somewhat different driver for improvement than the previous method increments. This method increment was initiated in order to be CMMi certified. Several large customers demanded a certain CMMi level of Baan, so extra practices needed to be implemented to meet this demand.

In this case, the development center in the Netherlands decided to implement the tracing sheet, which purpose was to provide evidence of an association between a requirement and its source requirements, its realization, and its verification. Using the tracing sheet had several benefits: maintaining consistency between business requirements and conceptual solutions; checking the completeness of the work in progress; assisting in finding affected documents and components in case of a change request; and tracking the progress of the project.

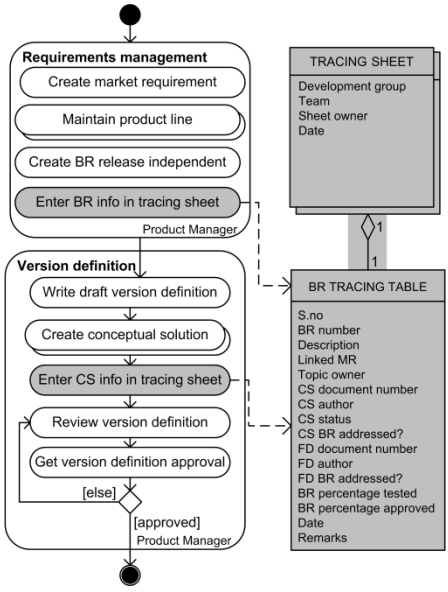


Figure 6-9. Snapshot of increment 6

In Figure 6-9, increment 6 is depicted. In this snapshot, a large part of the concepts is omitted because of space limitations. The concepts that are not depicted in this snapshot, but are depicted in the snapshot of increment 5, have not been altered. The method increment consists of the addition of several method fragments. First, the activities Enter BR info in tracing sheet and Enter CS info in tracing sheet have been added to the process side of the diagram. At the

deliverable side of the PDD, two concepts have been added: TRACING SHEET, with four attributes, and TRACING TABLE, with ten attributes. These concepts are connected by an aggregation.

In Figure 6-10, we illustrate a part of the tracing sheet. The example shows (among others) a sheet with three business requirements, with the description, linked market requirement, and some information concerning the conceptual solution in which the market requirement is detailed.

Note that increments 4, 5 and 6 were relatively small and introduced in a timeframe of nine months (see Table 6-1). These increments were deliberately kept small for the sake of easy implementation (cf. 6.5.3-2).

	Dev. Group		ERP			
	Team		Barneveld			
	Sheet Owner		Bovenkamp			
	Date		7-Apr-2000			
				Conceptual Solution		
S.no	BR no	Description	Linked MR	T.Owner	Doc. No.	Author
1	BR1-10075	Hours Accounting	MR1-100934	Johnson	DO994A US	Burnet
2	BR1-10092	Hours budgetting	MR1-100954	Johnson	DO996A US	Burnet
3	BR1-10072	User support	MR1-100824	Jansen	DO987B NL	Jansen

Figure 6-10. Part of the Tracing Sheet

6.4.7 Increment 12: Omitting the common component framework

For the sake of brevity, we have skipped several method increments. The other increments are described in van de Weerd, Brinkkemper and Versendaal (2006). Method increment 12 takes place in 2003. In 2000, Baan got in serious financial problems and is finally taken over by Invensys, which was in that time Europe's second largest software company, just behind Germany's SAP.

Several problems concerning the common component framework are identified. First of all, product managers found it difficult to make the distinction between common and market-specific components. Secondly, in practice the developers did not use the common components, because they found it too much work. Thirdly, because of the many acquisitions in this period, and consequently the long range of acquired products, the common component framework was too difficult to maintain.

Altogether, the common component framework could not be maintained and management decided to eliminate the common component framework and only focus on the linkage between business requirements and market requirements.

In Figure 6-11, the snapshot in increment 12 is depicted. One activity has been deleted, namely Maintain product line. Also, two concepts are deleted: COMMON COMPONENT and THEME, together with their attributes and their relationships to MARKET REQUIREMENT and BUSINESS REQUIREMENT.

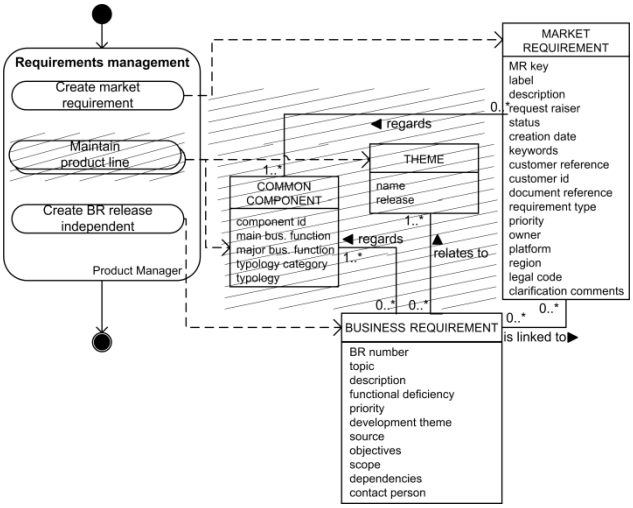


Figure 6-11. Snapshot of increment 12

6.4.8 Increment 13: Voting sheet

In June 2003, Invensys sold the Baan company to SSA Global, a large ERP vendor. A year later, method increment 13 took place. At that time, the requirements database had gotten so large that is was not useful anymore. At that moment several thousands of market requirements were stored and it was impossible to link them all properly to business requirements. The consequence was that customers complained because their demands were not met.

The solution was to leave the system of storing separate market and business requirements. Now, the market requirements are stored in the voting list. During a customer voting process, the ones that receive the most votes are stored in the requirements database as business requirement, the ones that score average are saved for the next voting round, and the ones that receive almost no votes are deleted.

In Figure 6-12, one extra activity is inserted, namely Carry out customer voting process. Another activity is modified, namely from Create BR release independent to Store selected MRs as BRs. At the deliverable side, one extra concept is added, the VOTING SHEET. This concept has several new attributes and an aggregation relationship to MARKET REQUIREMENTS. Several attributes have been deleted, modified or inserted in the existing concepts. Finally, the relationship between MARKET REQUIREMENT and BUSINESS REQUIREMENT is changed from aggregation to generalization.

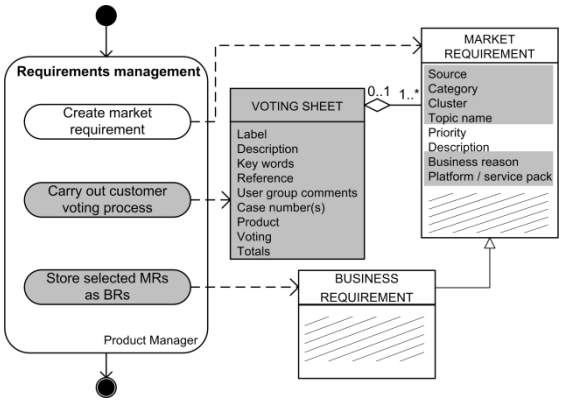


Figure 6-12. Snapshot of increment 13

6.5 Analysis and lessons learned

Reflecting on the retrospective case study that we carried out at Baan, we identified several lessons learned. First, we give an overview of our method increment analysis. Then, we describe the lessons learned on incremental method evolution. Finally, we reflect on four lessons learned at Baan, concerning software product management in a global setting.

6.5.1 Method increments analysis

Looking at the modeled method increments, we can conclude that all increments found in the retrospective case study can be modeled with the 18 elementary method increment types that are listed in Section 6.2.2.

By counting the occurrences of elementary method increments in the series of analyzed method increments, we have constructed Table 6-2. Several things attract attention. Firstly, properties are most often inserted, modified or deleted.

These kinds of method increments happen because of two reasons. First, it can be a side effect of another, more essential, change in the method. For example, in method increment 4, a release table is added. This leads to an inevitable change in the properties of the requirements concept, since this concept is now not part of the version definition anymore, but exists on its own, and is referred to in the release table. The other reason for an introduction, modification or change of attributes can be found in the experience of the users of the method. When a product manager or developer uses a certain template, it may become clear that one of the attributes is newly required or becomes redundant. These kinds of changes are easy to implement without changing the entire method rationale.

The second thing that attracts attention is that concepts and activity nodes are inserted regularly, but not often deleted. Consequently, the same holds for Relationships and Transitions. The reason for this is quite clear: during most of the years that are covered in the case study, Baan went through a large growth. Furthermore, during the period of downsizing after 2001, the method stays in place and is hardly scaled down in a similar proportion.

Role is the method fragment that changes the least. Two new roles were inserted: first the program manager and later the product manager. These roles kept on existing, as well as the development role. Of course, the number of employees in these roles changed over time. At a certain moment 60 product managers were employed and at other times 10. However, the role of the product manager was never omitted.

Table 6-2. Overview of counted method increment types

	Concept	**Property**	**Relationship**	**Activity node**	**Transition**	**Role**
Introduction	17	147	17	24	21	2
Modification	2	10	2	2	0	0
Deletion	3	38	5	2	1	0

6.5.2 Regarding incremental method evolution

Concerning incremental method evolution, we learned that increment drivers vary during evolution. The improvement of the requirements management processes related to the method increments were motivated by four different drivers:

1. *Development management*: the growth of the development effort requires that the role of the product manager is established, with expanding

responsibilities for requirements management and release planning. As shown in Table 6-1, this driver caused five of the fourteen method increments.

2. *Business management*: the change of corporate business strategy requires process changes. The business management driver was responsible for four method increments.
3. *Departmental interfacing*: other departments, such as marketing and customer services, require process adaptations for smooth interoperations and performance improvement. We found three method increments that were triggered by this driver.
4. *Certification*: the ambition to achieve higher maturity levels in CMM requires the implementation of certain process extensions, although those processes may not strictly speaking be critical to the business. Two of the fourteen method increments were caused by the certification driver.
5. Usually a process change is due to a combination of drivers, but always one main motivation could be identified. Table 6-1 shows the variety in the main motivation for each method increment.

6.5.3 Regarding global Software Product Management

In the years of process improvement in requirements management, Baan gained a variety of experiences. From these, the following lessons learned were distilled.

1. *Shared infrastructure is critical for rollout.* The company-wide availability of and access to tools supporting the method is a critical success factor for the proper adoption of new method increments. In the Baan case these were the central requirements database, the development method on the intranet, and the document management systems for storing all development documents (cf. Brinkkemper, 2000).
2. *Small increments facilitate gradual process improvement.* Most organizations are not able to adopt large method increments in one step. Simple method adaptations such as for example the introduction of a new Conceptual Solution document (increment 5) or a new Tracing Sheet (increment 6), can easily be communicated to the product managers. Small increments usually require some simple instructions without a formal training program, and personal communication means (telephone, email) support the rollout to all teams worldwide.
3. *Global involvement is critical.* A centralistic process improvement effort will not be successful. Baan had a Software Engineering Process Group

(SEPG) composed of representatives of all major development centers. The global rollout of method increments was facilitated by the local SEPG representatives. Furthermore, for some improvements a distributed project team was formed, the so-called Process Action Team (PAT). The PAT, consisting of experts in the improvement area, developed the method increment with templates, instructions and examples. In this way methodical content was accumulated with contributions from all global development locations. (cf. increments 3 and 4).

4. *Acquired companies come later*. Product software companies are known for their strong corporate cultures (cf. Google (Vise & Malseed, 2005) and Microsoft (Cusumano & Selby (1995)). This also implies that the companies have their own process cultures. After acquisition it is important to support the existing culture and method for a while, and motivate the evolutionary adaptations toward a uniform method.

We are convinced that these lessons learned can serve as guidance for globally operating product software vendors.

6.6 Related work

Several studies to evolution of methods and processes within the software domain have been carried out. To start with, Nejmeh and Riddle (2005) argue that a company's context is the major influence on the definition and sequencing of process change cycles. Since all companies operate in different contexts, there is no "one size fits all" process evolution approach suitable for all companies. To cope with this, Nejmeh and Riddle developed a Process Evolution Dynamics Framework, which helps companies to cope with their process evolution in response to several drivers. As examples of these drivers, they mention customer desires, marketplace structure, personnel availability and capability, business goals, and available technology.

Ahn, Ahn and Park (2003) use a more generic structure to categorize increment drivers or, as they call it, evolution drivers. In their research, they propose a mechanism that supports the customization of software processes, based on Case Based Reasoning and a knowledge-based technique. The evolution drivers that they use are *improvement drivers*, such as complying with a certain CMM level, and *domain-specific drivers*, which can be any context driver, such as the use of a new technology.

The improvement drivers that Ahn, Ahn and Park (2003) identified are also used by Nguyen and Conradi (1996). They developed a categorization

framework for process evolution which consists of six dimensions: where, why, what, when, how and by whom. The *why* dimension represents the major causes (or driver) and is classified in four categories: *correction*, *refinement*, *adjustment*, and *improvement*. In a case, study Nguyen and Conradi (1996) found that 35-40% of the recorded evolutions were requested or caused by the customer; 40% of recorded evolutions occurred because of resource underestimating; and 20% of the evolutions were a result of revising the plan document by changing initially planned activities.

Bandinelli, Fuggetta and Ghezzi (1993) use yet again another categorization mechanism. They divide evolution drivers within a software process into three categories of change: *incremental definition*, which is caused by the fact the it is impossible to define an entire software process model from the beginning; *changes in the environment or organization*, which may be caused by, for example, the introduction of new tools and changes in the strategy; *customization of the software process model*, which refers to the possibility of the process agents (e.g. the developers) to change the process when necessary.

Finally, Ocampo and Münch (2006) carried out an exploratory study to process evolution rationale. In this study they analyzed a database that was filled by process engineers with changes on software processes as well as justifications of the changes. They found a list of 11 explanations of changes. However, all explanations concerned procedural issues, such as ambiguous descriptions, non-compliant activities and improper sequences of activities.

In our study we found four main increment drivers: *development management*, *business management*, *departmental interfacing*, and *certification*. Several of these drivers are also mentioned in other studies. For example, Ahn, Ahn and Park (2003) mention improvement drivers for complying with a certain standard. This clearly maps with our certification driver. However, many drivers do not directly map because we have a different focus. For example, Nejmeh and Riddle (2005) describe customer desires as an increment driver. We do not describe these issues, because we only look at the Software Product Management function. Issues that are handled by other departments, such as marketing and sales, are categorized in the increment driver Departmental interfacing. In the same way, we can group e.g. Availability and Capability under the Development driver.

6.7 Conclusions

In this work, we carried out a research on incremental method evolution in a global software product management setting. We used a retrospective case study to analyze method increments. The results of this case study show that all increments found in the retrospective case study can be modeled with the 18 method increment types that we identified. However, in this case study, not all increments that we identified occurred.

In addition to the validation of the generic method increment types, we researched the general principles behind method increments. We found four increment drivers, in which we can place all method increments. These are development management, business management, departmental interfacing and certification. In addition, we described our lessons learned by reflecting on twelve years of implementing process improvements in the product management processes at Baan.

Situational method engineering theory has focused mostly on constructing or adapting a method for a certain project. However, the *evolution* of methods that takes place in most companies is neglected. In this research, we show the mechanism of incremental situational method engineering. Each method increment is caused by an increment driver that arises from a changed company situation. By using method increments for software process improvement, companies can implement small, local changes in their processes. Accordingly, they are not forced to radical changes. Our experience is that by properly embedding these increments in the existing infrastructure and communicating them to the right knowledge workers, process improvements can be implemented evolutionary and successfully.

Currently, we are working on the realization of the Product Software Knowledge Infrastructure (van de Weerd, Versendaal & Brinkkemper, 2006; van de Weerd, Brinkkemper & Versendaal, 2007). With this infrastructure, product software companies can obtain a custom-made advice that helps them to improve their processes. The method increment types that we validated in this case study are used to implement assembly rules in the infrastructure. In addition, research is being done to the situational factors that influence the choice of software product management methods (Bekkers, van de Weerd, Brinkkemper & Mahieu, 2008a). In the future, we plan to fill the method base with these situational factors, method fragments and assembly rules, and validate it at product software companies of different sizes and in different sectors.

CHAPTER 7

A method engineering approach to process improvement

Method Engineering and Requirements Engineering are two research fields that can benefit from another. To increase process maturity in systems development, we propose an approach for incremental method evolution that combines capability-based and problem-based methods. With this method, we can assemble new methods, based on the process need of an organization. We show how this approach can be implemented using Computer Aided Method Engineering (CAME) technology. In addition, we demonstrate the utility of the Product Software Knowledge Infrastructure by showing an example of the insertion of cost-value prioritization as a method increment in software product management. This shows how isolated innovations in the Requirements Engineering domain can be embedded in software development practices.[1]

7.1 Method engineering and requirements engineering

The research areas of Method Engineering and Requirements Engineering share a common interest, as they both aim at promoting process improvements in software and systems developments. Method Engineering works from the perspective of generic method descriptions, usually called meta-models and possibly supported by tooling, that allow for the roll-out of uniform high-quality methods in the perspective of full means for situational adaptation of the method to the circumstances at hand.

Requirements Engineering research focuses on all techniques for the proper description and handling of the specifications of a systems development process, or as Nuseibeh and Easterbrook (2000) formulate more formally, "the

[1] This work was originally published as:

Brinkkemper, S., Weerd, I. van de, Saeki, M., Versendaal, J. (2008). Process improvement in requirements management: a method engineering approach. *Proceedings of the 13th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'08)*, LNCS 5025, 6-22.

process of discovering the software system's purpose, by identifying stakeholders and their needs, and documenting these in a form that is amenable to analysis, communication, and subsequent implementation". In the scientific work of requirements engineering we see all kinds of innovative approaches being proposed related to the elicitation, modeling and analysis, communicating, validating and evolution of requirements.

This paper aims at establishing a cross-fertilization of the two perspectives by showing how requirements engineering techniques can be embedded into a systems development method supported by method engineering principles. We demonstrate this by inserting a cost-value requirements prioritization technique, developed by Karlsson and Ryan (1997), into the requirements management methods of a product software company (van de Weerd, Brinkkemper, Nieuwenhuis, Versendaal & Bijlsma, 2006).

7.1.1 Methods for product software development

Product software is a worldwide industry, yet this domain has not been subject of much scientific research. The last years, this is changing however. There have been several studies on all product software, focusing on product software as a research domain (Xu & Brinkkemper, 2007), product development (MacCormack, 2001; Hietala, Kontio, Jokinen & Pyysiainen, 2004) management of software products (van de Weerd, Brinkkemper, Nieuwenhuis, Versendaal & Bijlsma, 2006; Ebert, 2007), requirements management (Regnell, Höst, Natt och Dag, Beremark & Hjelm, 2001), release planning (van den Akker, Brinkkemper, van Diepen & Versendaal, 2005; Ruhe & Saliu, 2005), product line engineering (Pohl, Böckle & van der Linden, 2005; Kang, Lee & Donohoe, 2002), product delivery (Jansen, Ballintijn, Brinkkemper & van Nieuwland, 2006), and so on.

Xu and Brinkkemper (2007) summarize a number of specific characteristics of developing product software. An important difference is, for example, that the production costs do not depend on the number of copies sold. Therefore, product software companies that are selling millions of copies can have up to 99% gross profit margins for its product sales (Xu & Brinkkemper, 2007). On the other hand, the majority of the product-development project are late or over budget. Also, the requirements of the entire market must be held into account. This means that a software product should be developed so that it can run on different hardware and software platforms. All these characteristics make product software development a highly complex business, in which process failures have a huge impact on performance.

Furthermore, as is depicted in Figure 7-1, the success of a software product in the market has consequences for the internal functioning of the company. From a start-up creating a first release product by a relatively simple process, the growing company is shipping subsequent releases and product enhancements by utilizing a product development approach that should be incrementally adapted to the changing conditions.

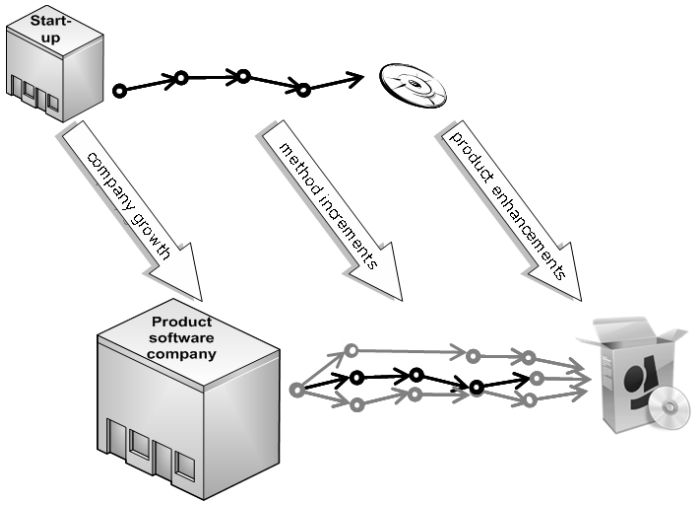


Figure 7-1. Incremental method evolution in a product software company

Several software process improvement approaches have been proposed to improve software development processes (Paulk, Weber, Curtis & Chrissis, 1995; el Emam, Drouin & Melo, 1998). These approaches are usually capability-based, i.e. based on the current capabilities of a company an advice is given which entails the implementation of capabilities on a higher maturity level. However, the increments in these approaches are often too large and general, instead of local and situational. For example, SEI has done a survey among 1,804 organizations, which indicates that the median time, to move from one CMM level to another, ranges from thirteen to twenty-four months (Process Maturity Profile, 2006).

In this research, we want to extend the capability-based process improvement with root-cause analysis, in order to give a more accurate analysis of the actual problem. We implement our approach in the Product Software Knowledge Infrastructure (PSKI) (van de Weerd, Versendaal & Brinkkemper, 2006; van de Weerd, Brinkkemper & Versendaal, 2007), which, when fully materialized, can help to increase the maturity of a company's processes. For

scoping reasons we limit our research to the software product management domain.

7.1.2 Research approach

This research project is carried out following the design research methodology for performing research in information systems as described by March and Smith (1995) and Hevner, March, Park and Ram (2004). They state that research in design science is done through the processes of *building* and *evaluating* artifacts. According to Hevner, March, Park and Ram (2004), the fundamental questions in design-science research are: *"What utility does the new artifact provide?"* and *"What demonstrates that utility?"* In addition, they provide seven guidelines on performing design-science that have been followed during this research. The first guideline Hevner at al. propose is that "design-science research must produce a viable artifact in the form of a construct, a model, a method, or an instantiation". The artifact in this research is the Product Software Knowledge Infrastructure (PSKI), or to be more specific, the functional architecture of the PSKI. The second guideline is *problem relevance*, which Hevner at al describe as "the objective of design-science research is to develop technology-based solutions to important and relevant business problems". The business problem lies in the fact that product software market is growing and that there is a need for methodical support, in order to increase the maturity of product software organizations. By developing the PSKI, we offer a technology-based solution to this problem. The other guidelines comprise: design evaluation, research contributions, research rigor, design as a search process, and communication of research. The page length of this paper limits us to describing each guideline in detail.

In earlier work (van de Weerd, Versendaal & Brinkkemper, 2006), we described our vision on this issue and introduced the PSKI, our main new artifact. Subsequently, in van de Weerd, Brinkkemper & Versendaal (2007), we identified and formalized general method increments that were found in an exploratory case study. In addition, we formalized common process needs, by developing a root-cause map for software product management and by identifying the root causes and process alternatives that are related to them. Finally, a first prototype of a method base for software product management is developed[1], based on the reference framework for software product management (van de Weerd, Brinkkemper, Nieuwenhuis, Versendaal & Bijlsma, 2006).

[1] http://www.softwareproductmanagement.org/

In this research we want to elaborate on the process improvement approach that will be implemented in the PSKI and its functional architecture of the PSKI. We evaluate this by using scenarios to demonstrate its utility. In Section 2, we will describe the realization of the PSKI, by elaborating on the requirements and functional architecture. Section 3 explains the technical realization of integrating the PSKI with a CAME tool. In section 4, we will give a scenario of a method increment, advised and assembled by the PSKI. Then, in Section 5, we give an overview of related literature. Finally, in section 6, we will describe the conclusions and further research.

7.2 Realization of the Product Software Knowledge Infrastructure

In this section we will first describe the rationale of the software improvement approach we use. Then we describe the functional architecture of the PSKI and show a typical scenario.

7.2.1 A combined process improvement approach

We propose the distinction between two types of process improvement approaches: the capability based and problem-based approach. The capability-based approach is based on the assumption that a company's capabilities should grow in maturity in order to increase performance. By assessing the organization's current capabilities, the maturity level can be determined and recommendations of implementing capabilities on a higher maturity level can be made. Examples of capability-based approaches are CMM (Paulk, Weber, Curtis & Chrissis, 1995) and SPICE (el Emam, Drouin & Melo, 1998). Secondly, the problem-based approach uses the mechanism of solving the underlying problems, or root causes, that cause a certain process to underperform. An example of a problem-based approach is RCA, which has been applied to process improvement and incident prevention in software and non-software industries; see for example Leszak, Perry & Stoll (2000).

In literature, some critique exists on capability-based approaches. For example, the organizational structure of small companies may not be suitable for a large process initiative (Demirors & Demirors, 1998). In addition, these kinds of process improvement approaches are often difficult to implement. Staples et al. (2007) found that CMMi is often not adopted by organizations because the following reasons: the organization was small; the services were too costly, and the organization had no time to implement the process

improvements. From our experience, we also found that capability-based approaches often are too superficial for the specific nature of product software companies. Moreover, we do not want to force companies to a company-wide process improvement program. On the other hand, following a complete problem-based approach would be too inefficient, due to the extensive analysis process that needs to be done. Therefore, we propose the *combined process improvement approach*, in which we complement the capability-based approach with problem-based aspects. When comparing this approach to the existing capability-based approaches such as SPICE and CMM, we envision the following advantages: 1) the maturity levels can be determined per process, which makes it possible to implement very small process improvements; 2) the capability-based approach is extended with a problem-based approach to be able to determine the more complex problems that underlay a unsatisfactory process. In Figure 7-2, we illustrate this approach.

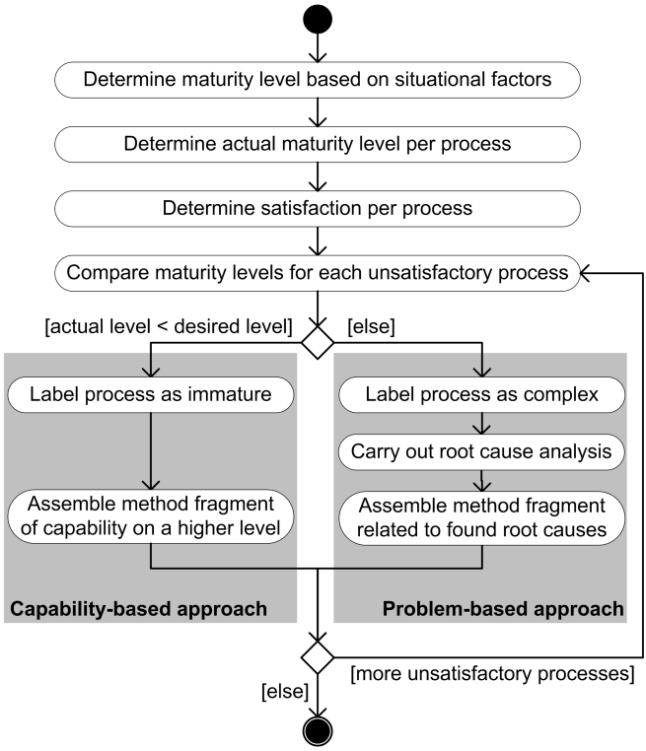


Figure 7-2. Process improvement approach

The process starts with determining the maturity level that the company *should have*, based on the situational factors of the company. For example, a company with 500 employees should be on a higher maturity level than a company with six employees. Secondly, the actual maturity levels per process are retrieved by performing a capability assessment. By inventorying which capabilities are mastered per process, the maturity level can be calculated. In addition, the user is asked whether the result of the concerned process is satisfactory or unsatisfactory. For each unsatisfactory process then, the maturity level as it *should be* (based on situational factors) and the actual maturity level are compared. If the actual maturity level is lower than the maturity level based on situational factors, then the process is labeled as immature and a process improvement is necessary. The process improvement is carried out by assembling a method fragment related to the capability on a higher maturity level. If the actual level is equal to or higher than the desired level, then the process is labeled as complex, and a root-cause analysis is carried out to find the underlying problems. The process improvement is carried out by assembling a method fragment, related to the found root causes.

7.2.2 Functional architecture

In Figure 7-3, the functional architecture of the PSKI is depicted. Starting from van de Weerd, Versendaal & Brinkkemper (2006), the following components can be identified: a *web-based interface* to communicate with the user; an *assessment base*, to store the assessment questions and answers; the *assessment administrator*, which can be used to add questions to the *assessment base*; and the *CAME too*l in which the method fragments are stored.

The main components of the PSKI are the assessment base and the CAME tool. In the assessment base, the PSKI stores assessment questions, answers, situational factors and capabilities. We distinguish two types of questions: *situation questions* that identify which situational factor apply to a company or product line and *capability questions* that assess which capabilities a company possesses. The second component is the CAME tool, which consists of a *method base*, in which method fragments their information are stored; a *PDD editor*, with which the method engineer can define the meta-modeling language that is used for administrating the methods; and the *method administrator* that is used by the method engineer to add methods to the method base. The method fragments that are stored in the method base consist of activities and deliverables. Each method fragment is labeled with a capability level and linked to zero or more values of situational factors.

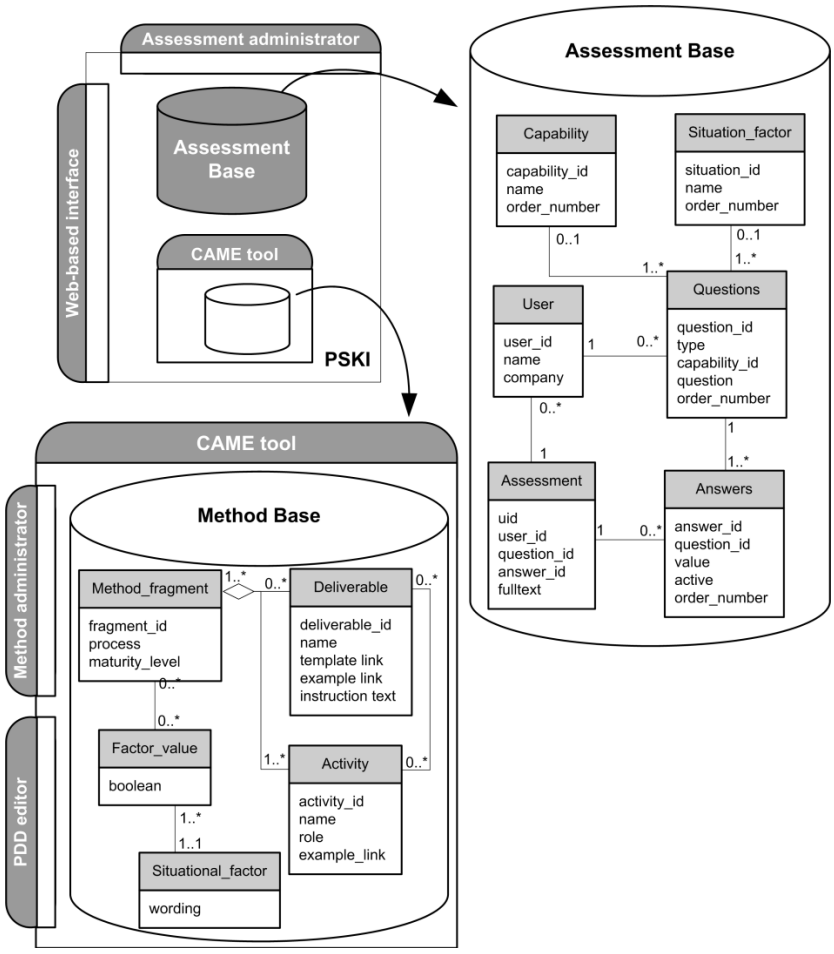


Figure 7-3. Functional architecture of the PSKI

7.2.3 Illustrative example: ERPComp

To illustrate the utility of the PSKI, we use a running example, which concerns an organization that develops ERP systems (ERPComp). ERPComp is 3 years old and currently has 50 employees. The user in this case is the product manager of the organization, who uses the PSKI because his organization has several problems: a) the releases are often not delivered in time, and b) the stakeholders are not satisfied with the implemented requirements.

Figure 7-4 illustrates the current requirements management process of ERPComp in PDD notation (van de Weerd & Brinkkemper, 2008).

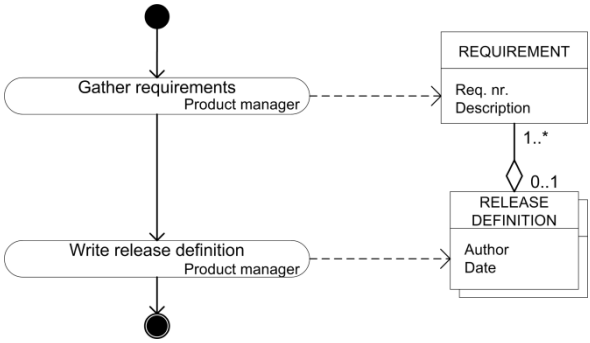


Figure 7-4. Snapshot of increment #0

The diagram shows a snapshot of the method at a certain time n, say method increment #0. It covers the requirements management activity of a company, and has two sub activities: Gather requirements, resulting in a REQUIREMENT and Write release definition, resulting in a RELEASE DEFINITION, which are both carried out by the product manager.

7.2.4 A typical scenario in the PSKI

In Figure 5, we show again the functional architecture of the PSKI enriched with the process that is followed when interacting with the PSKI. We will elaborate on this process by using the ERPComp example. Note that in this case, the capability-based approach is followed, as indicated in Figure 7-5. By following the solid arrows, the activities concerning the problem-based approach (*Present root cause map* and *Store root causes*) are skipped.

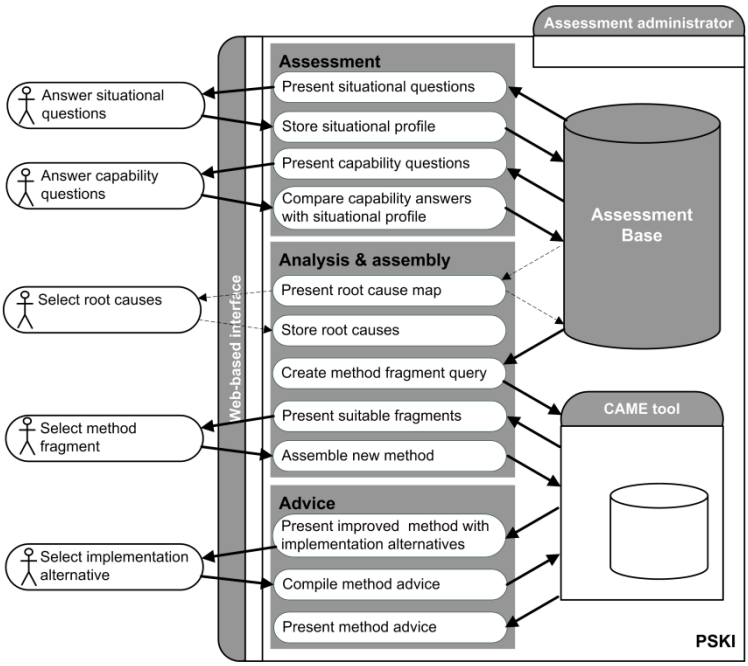


Figure 7-5. Capability-based PSKI scenario

The scenario depicted in Figure 5 describes a sequence of the following activities:

PSKI: Present situational questions

PSKI presents a form with a predefined set of situational assessment questions.

User: Answer situational questions

The product manager answers situational questions:

1. What is the age of your organization (in years)?	○ <1 ◉ 1-5 ○ 5-10 ○ >10
2. In which sector does your organization operate?	Large-sized enterprises ▾
3. What is the size of the development team?	◉ 1-4 ○ 5-9 ○ 10-20 ○ >20
…	
8. What is the number of product lines?	◉ 1 ○ 2-4 ○ 5-8 ○ >9
9. Which platform is used to develop your product on?	.NET ▾

PSKI: Store situational profile

PSKI stores the answers to the situational questions as a situational user profile in the assessment base. Also, based on this profile, the desired

maturity level is obtained. In this case, based on the age of the organization (3 years) and the sector in which the organization operates, the PSKI determines the maturity level at 4 (of 12, see Section 3.2).

PSKI: Present capability questions

Based on the answers to the situational questions, PSKI selects a subset from the capability questions, namely those questions that have the same type as indicated in by the user in his situational answers. This means that only questions that are applicable for large-sized organizations, with multiple products, developed on a .NET platform, are selected. Examples of these questions are:

1. Does your organization perform requirements prioritization per release?	○ yes	◉ no
2. Is there a prioritized requirements list?	○ yes	◉ no
3. Is the prioritized requirements list properly available for other stakeholders?	○ yes	◉ no
4. Is there a Product Manager responsible for the requirements prioritization per release?	○ yes	◉ no

User: Answer capability questions

The product manager answers the capability questions.

PSKI: Compare capability answers with situational profile

PSKI stores the answers to the capability assessment questions as a capability profile in the assessment base. When comparing the capability profile with the desired maturity level, the PSKI finds the following:

Process	*Right capabilities in place?*	*Result satisfactory?*
Requirements gathering	Yes	Yes
Requirements validation	No	No
Requirements prioritization	No	No

In case the product manager would have found the requirements gathering process unsatisfactory, although the process was at the right maturity level, the PSKI would present the root cause map of this process. However, due to limited space, we will not elaborate on such an example. The remaining steps are therefore:

PSKI: Create method fragment query

PSKI creates method fragment query, which retrieves those method fragments that are linked to the capabilities that should be implemented, in this case the level-3 capabilities of the *Requirements validation* and *Requirements prioritization* processes.

PSKI: Present suitable answers

PSKI displays all matching method fragments. Below, we depict an

example of two method fragments for the Requirements prioritization process.

1. **Requirements prioritization via a stakeholder voting round**
   Description: The product manager schedules a meeting in which each stakeholder gives his top x of requirements that need to be implemented in the next release. The requirements with the most votes will be implemented.
   Roles: Product manager, involved stakeholders
   Deliverables: REQUIREMENTS LIST with prioritized REQUIREMENTS
2. **Requirements prioritization via the cost-value approach**
   Description: In the cost-value approach, the relative costs and relative values of each requirement are estimated. Then, they are plotted on a cost-value diagram, which shows which requirements will generate the highest value and the lowest costs. Based on this diagram, the product manager prioritizes the requirements.
   Roles: Product manager, product group, customers, software engineer
   Deliverables: COST-VALUE DIAGRAM, prioritized REQUIREMENTS

User: Select method fragments
The product manager selects method fragments that are perceived as useful.

PSKI: Assemble new method
PSKI assembles the selected method fragment into the existing method fragments of the company.

PSKI: Present method with implementation alternatives
PSKI presents method accompanied by a number of different implementation alternatives.

User: Select implementation alternative
The product manager selects suitable implementation alternative

PSKI: Compile method advice
PSKI compiles method advice is compiled, according to the selected implementation alternative. In case the user has selected root causes, an advice is added on how to solve these.

PSKI: Present method advice
PSKI presents the method advice to the product manager.

7.3 Method improvement based on situational capability matching

In this section, we elaborate on the retrieving process of method fragments from the method base. Instead of building the method base ourselves, we use an existing tool, namely MetaEdit+. MetaEdit+ is an integrated modeling and meta-modeling environment for domain-specific languages (Kelly, Lyytinen & Rossi, 1996; Tolvanen, 2006). In MetaEdit+, we have realized our PDD notation as a meta-model. Now, it is possible to create, store and manipulate method fragments as PDDs. A screenshot of MetaEdit+ can be found at the end of this paper, in Figure 7-10.

7.3.1 Method fragment structure

A method fragment consists of a process fragment and a deliverable fragment. Method fragments can contain multiple activities and multiple deliverables. Also, constructs like branches, joining and forking of activities and aggregated deliverables can be modeled, as shown in Figure 7-8 and Figure 7-9 and described in van de Weerd and Brinkkemper (2008). The structure of a generic method fragment is depicted in Figure 7-6.

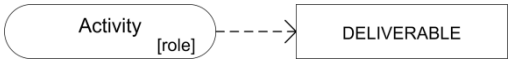


Figure 7-6. Generic method fragment structure

The name of an activity in a method fragment is a composition of one or more verbs, possibly an adjective and a noun, e.g. *Prioritize [verb] requirements [noun]*. Furthermore, an activity is carried out by a role, e.g. *Product Manager [role]*.

The structure of method fragments is used in the generation of capability questions for the capability assessment. We distinguish two types of capability questions: standard questions, which can be generated from the stored activities, deliverables and capabilities; and comprehensive questions, which are especially useful for assessing capabilities at a higher level. Based on the activities, deliverables and capabilities, we can derive the capability assessment questions. Each capability is related to three basic assessment questions, namely:

1. Does your organization perform the [capability]?
2. Is there a [deliverable]?
3. Is the [deliverable] properly available for other stakeholders?

4. Is there a [role] responsible for the [capability]?

In section 7.2.3, three capability assessment questions were listed. These were the assessment questions for capability A: Requirements prioritization per release, namely:

1. Does your organization perform requirements the prioritization per release?
2. Is there a prioritized requirements list?
3. Is the prioritized requirements list properly available for other stakeholders?
4. Is there a Product Manager responsible for the requirements prioritization per release?

In ERPComp, the product manager answers 'no' to all questions, since there is no requirements prioritization process in place.

7.3.2 Maturity matrix for Software Product Management

To assess the state of the SPM function in an organization, we developed the SPM maturity matrix. This maturity matrix is inspired by on the DYA architecture maturity model (Steenbergen, Brinkkemper & van den Berg, 2007) and the Test Process Improvement model (Koomen & Pol, 1999). We distinguish 16 SPM processes in the maturity matrix that originate from the reference framework for SPM (van de Weerd, Brinkkemper, Nieuwenhuis, Versendaal & Bijlsma, 2006) and 11 maturity levels. The number of maturity levels is determined by the implementation dependencies of the capabilities. In Table 7-1, we show an excerpt of the matrix, covering three processes. Each process has its own path to maturity, indicated by the letters A, B, C and D. Every letter represents a capability, which we define as the demonstrable ability and capacity to perform a certain process at a certain level. The position of the letters shows the preferred order in which the capabilities need to be implemented to reach a certain maturity level. 10 is the lowest maturity level and 12 is the highest maturity level. Suppose that a company should be on maturity level 4, based on its situational factors. This means that for Requirements prioritization, capabilities A and B should be implemented; for Requirements validation, capability A should be implemented; and for Requirements gathering, capabilities A and B should be implemented.

Table 7-1. Excerpt of the maturity matrix for Software Product Management

Process Maturity level	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Requirements prioritization			A		B		C			D		
Requirements validation				A		B		C			D	
Requirements gathering		A			B		C		D			

In ERPComp, the desired maturity level that is deducted from the situational user profile is level 3. We will elaborate on two processes in the SPM maturity matrix, namely requirements prioritization and requirements validation. Please note that although the capability structure of the requirements prioritization and requirements organizing processes are the same, this may vary in other processes.

In the requirements prioritization process we distinguish four capabilities:

1. Requirements prioritization per release
2. Requirements prioritization as an ongoing process
3. Requirements prioritization as an ongoing process, over multiple product lines
4. Requirements prioritization as a chain-wide process

Currently, no prioritization process is in place. Looking at the matrix, we see that level-3 companies should have capability A (Requirements prioritization per release) implemented.

For the requirements validation process, also four capabilities are distinguished:

1. Requirements validation per release
2. Requirements validation as an ongoing, automated process,
3. Requirements validation as an ongoing process, over multiple products
4. Requirements validation as a chain-wide process

Currently, there is no validation at all. A level-3 organization should master capability A: Requirements validation per release.

7.4 Method increment example

In this section, we illustrate a process improvement by a capability-based method increment. The snapshot of increment #0, that we showed in Figure 7-4, is created in MetaEdit+. This means that not only visual information is stored, but also extra information, depending on the variables that we added to the different concepts.

As described in section 7.3.1, the organization should implement two method increments. The first method increment concerns the capability ‘Requirements prioritization per release’. As described in section 2.4, two method fragments are related to this capability. In this case, the user chooses the method fragment ‘Requirements prioritization via the cost-value approach’, as is depicted in Figure 7-7. The cost-value approach is proposed in Karlsson and Ryan (1997) an evaluated in Lehtola and Kauppinen (2006) as a method for requirements prioritization in market-driven software product development.

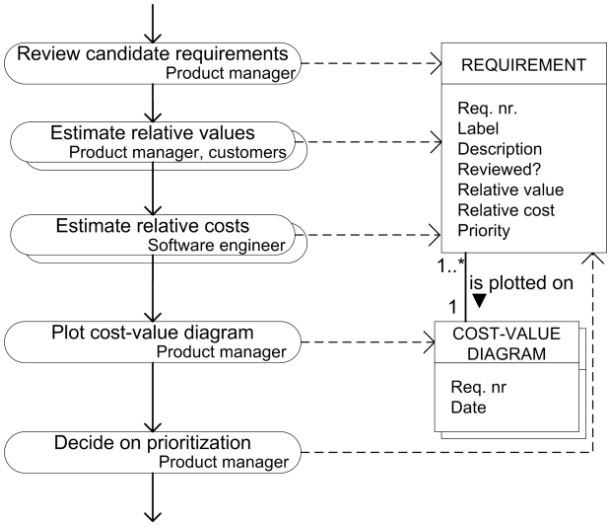


Figure 7-7. Method fragment linked to 'Requirements prioritization per release'

In Figure 7-8, we illustrate the method fragment related to the capability ‘Requirements validation per release’.

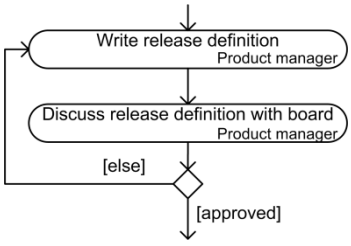


Figure 7-8. Method fragment linked to 'Requirements validation per release'

The fragment does not have the standard form of activity – deliverable, but the activity results in a decision, indicated by a branch. The Product manager

discusses the RELEASE DEFINITION with the board. If the board approves it, the release can be implemented. If not, the RELEASE DEFINITION has to be rewritten.

In Figure 7-9, we illustrate the snapshot of the improved method. It includes the method increments described in Figure 7-7 and Figure 7-8. The roles of the activities are filled in based on the situational information that was provided during the situational assessment.

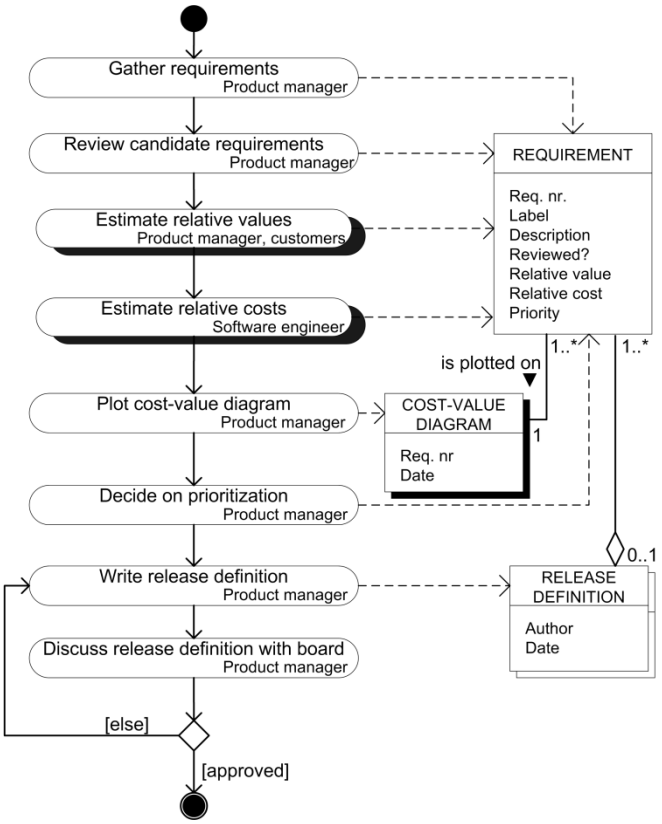


Figure 7-9. Snapshot of increment #1

Finally, we want to show how we created the method fragments we presented in this paper. In Figure 7-10 we show a screenshot of MetaEdit+, in which a new method is modeled. Looking at the scenario we explained in Section 7.2.4, we can position this activity, although it is not automated yet, in the step ‘Assemble new method’. In the future, this activity will be automated.

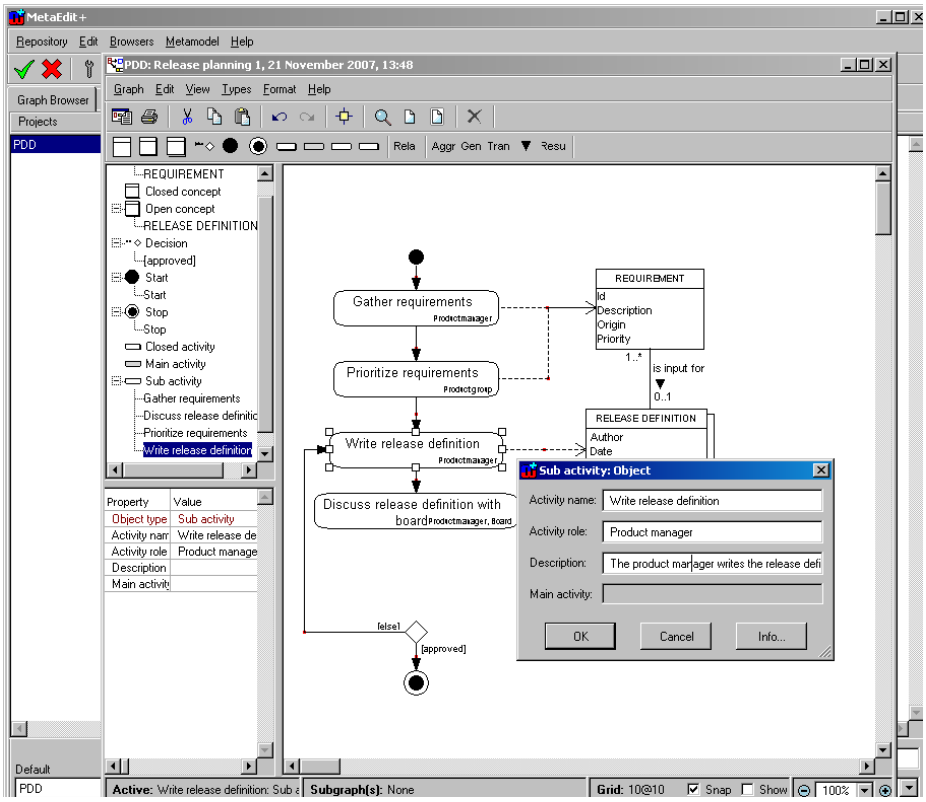


Figure 7-10. Assembly of the improved method in MetaEdit+

7.5 Related literature

Damian, Zowghi, Vaidyanathasamy and Pal (2004) state that there is a scarcity of requirements engineering-related software process improvement initiatives in the literature. In addition, Sawyer, Sommerville and Viller (1997) and Niazi (2002) state that existing software process improvement approaches leave a gap regarding requirements engineering. Therefore, they propose a practice-based approach to requirements engineering process improvement. Also other studies have been done to process improvement in requirements engineering. For example, Regnell, Beremark and Eklundh (1998) describe a requirements engineering process improvement programme, based on lessons learned from the implementation of a requirements engineering approach for packaged software. Beecham and Hall (2005) also point out that the requirements phase of software development is in need of further support. They propose the

Requirements Capability Maturity Model (R-CMM) as a first step in the solution to this problem.

Product software developers use methods and techniques in all phases of the development process, which are often supported by different software tools. These software tools range from simple text editors to complex tools for generating code from design specifications. Not only techniques on a low level can be automated, but also methods, which focus more on the high-level activities and deliverables of a process, can be automated. In the nineties of the previous century, this lead to a new research discipline, namely Method Engineering (Kumar & Welke, 1992; Brinkkemper, 1996; Rolland & Prakash, 1996), which comprises the design, construction and adaptation of methods, techniques and tools for the development of ISs. Tools were being designed to support the method engineering process, which are called computer-aided method engineering (CAME) tools (Kelly, Lyytinen & Rossi (1996). Many CAME tools have been developed in the last years, some for research purposes and some for commercial purposes. Their appliances vary from domain-specific modeling, to configuration management and situational method engineering.

7.6 Conclusions and further research

In this research, we proposed an approach for incremental method evolution that provides means by which innovative requirements engineering techniques can be inserted into systems development methods based on method engineering principles. This vision on process improvement combines a capability-based approach with problem-based aspects. We showed how this approach can be implemented in the PSKI by elaborating on the functional architecture. In addition, we explained the utility of the PSKI by giving an example of a method increment, i.e. cost-value requirements prioritization in software product management. This generic approach defines structure and relations of capabilities and method fragments, and generates capability questions automatically. Finally, we showed how method increments can be generated based on the situational and capability assessment answers.

We are currently working on the development of the PSKI and filling it with situational factors, capabilities and method fragments. In the future, we will use case studies to test the infrastructure at product software companies of different sizes and in different sectors, in order to test the mapping between situational factors, maturity capabilities and method fragments. We are confident that this paper shows how method engineering and requirements engineering research can benefit from each other.

CHAPTER 8

A maturity matrix for software product management

The quality of processes in Software Product Management (SPM) has a high impact on the success of a software product, as it improves product quality and prevents release delays. To improve the SPM practice, we propose the maturity matrix for SPM, a focus area oriented maturity model concentrating on the SPM functions Requirements Management, Product Roadmapping, Release Planning and Requirements Management. In this paper, we describe the development of the SPM maturity matrix, consisting of (a) identification and description of capabilities, (b) positioning the capabilities at the right levels in the maturity matrix and (c) validating the maturity matrix with expert validation and a survey among 32 product managers. The result is a validated maturity matrix that will guide further development of methodical support in SPM.

8.1 Maturity in SPM

Software product management (SPM) is a crucial area within many software companies. Good product management has a high impact on the success of a software product (Ebert, 2007). This requires a combination of technological, managerial and business skills, such as calculating optimal releases, setting out roadmaps, managing risks, and interacting with many internal and external stakeholders. If these activities do not get enough attention, the quality of a product decreases, release dates are not met, and managing customers' expectations become a large problem.

Although the product manager's function is highly important in the product software industry, little education exists in this area (van de Weerd, Brinkkemper, Nieuwenhuis, Versendaal & Bijlsma, 2006). Almost no education on SPM is being offered, except in the area of marketing and sales. Based on our experience in the market, especially in The Netherlands, Germany and Switzerland, we observe that most software product managers were earlier employed in functions such as development manager, project manager or sales manager. This causes a gap of knowledge that the product manager has to solve

by getting experienced in the area. Hence, lifting the quality of the product by improving the SPM processes is often difficult. Most existing software process improvement (SPI) models aim at a broad spectrum of SPI and the area of SPM is usually not the main area of attention.

In this research, we propose a maturity matrix for SPM that can be used to assess an organization's current SPM capabilities and offer local, incremental improvements to the product manager.

8.1.1 Background

In earlier research, we developed the Reference Framework for Software Product Management (van de Weerd, Brinkkemper, Nieuwenhuis, Versendaal & Bijlsma, 2006). Since its publication, various studies have been done to test the reference framework in product software companies (cf. Bekkers, van de Weerd, Brinkkemper & Mahieu, 2008a; van de Weerd, Brinkkemper & Versendaal, 2007). In this research, we use the reference framework as a foundation for our maturity matrix. Therefore, we will provide a brief explanation of the framework.

In Figure 8-1, the reference framework for software product management is depicted. The framework consists of *internal stakeholders* (product management, company board, sales & marketing, services, support, development and research & innovation) and *external stakeholders* (the market, partners and customers).

The most important internal stakeholder, Product management, consists of four business functions:

- *Portfolio management* concerns managing the different products that a company owns. Partnering, product lifecycle management and product line identification are part of this function.
- *Product roadmapping* handles with the development of the product roadmap, in which future releases are planned based on themes and core assets.
- *Requirements management* contains the activities of requirements gathering, identification and organizing; all ongoing activities within the product management domain.
- *Release planning* deals with prioritizing and selecting requirements in order to define the new release. Also the activities release validation, launch preparation and scope change management are part of this function

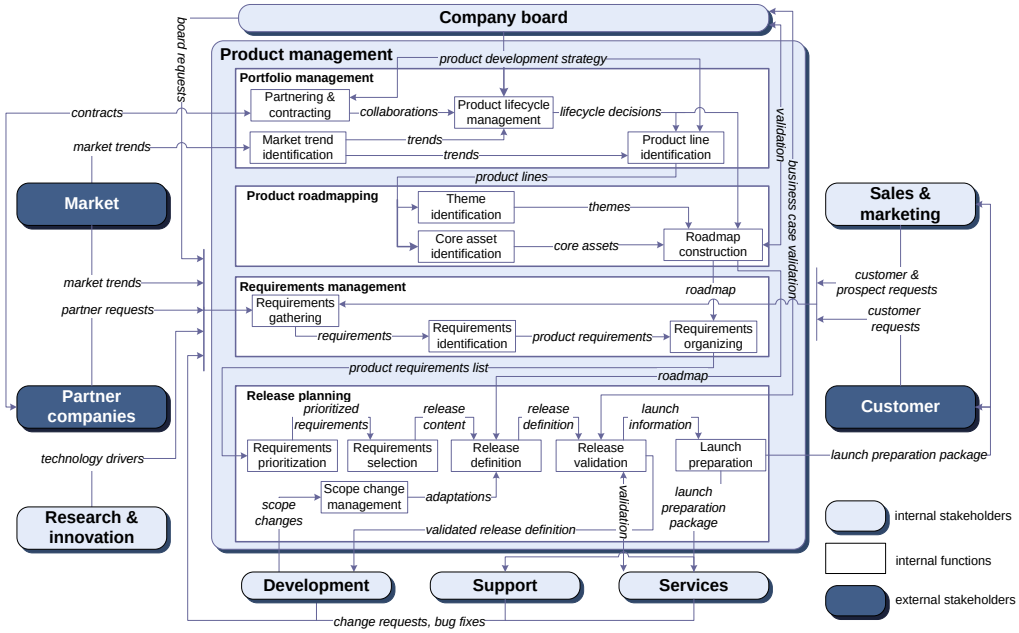


Figure 8-1. Reference framework for Software Product Management

8.1.2 Paper outline

This paper is organized as follows. Section 2 describes our research method. In Section 3, we describe the process of developing the maturity matrix. Then, in Section 4, we describe the empirical validation of the matrix. In Section 5, we describe the implications and outlook. Finally, in Section 6, we provide the conclusions of our research.

8.2 Research design

This study follows the design science methodology, in which research is done through the processes of building and evaluating artifacts (Hevner, March, Park & Ram, 2004). The artifact in this research is the maturity matrix for SPM. During our research, we follow the 5 process steps of the design cycle (Vaishnavi & Kuechler, 2007). This design cycle consists of several steps that follow an iterative process; knowledge produced in the process by constructing and evaluating the artifact is used as an input for a better awareness of the problem. The 5 process steps are:

1. *Awareness of the problem*. In Section 1, we described the problem and its context.
2. *Suggestion*. The suggestion for a solution to the problem identified in step 1 is developed in this step. In Section 2, we describe our approach in tackling the problem and the research methods that we use.
3. *Development*. The development of the artifact, in this case the maturity matrix is described in Section 3.
4. *Evaluation*. This step comprises the evaluation of the method. We used a survey to validate the method, as is described in Section 4. The results of this survey lead to a higher level of problem awareness and suggestions for solutions. We elaborate on these suggestions in Section 5.
5. *Conclusion*. Finally, in Section 6, conclusions and areas for further research are covered.

8.2.1 Artifacts

In Figure 8-2, we depicted the structures of the two artifacts in this research. First, the Reference framework is depicted, consisting of key processes that are grouped into business functions. Secondly, the Maturity matrix consists of key processes and SPM capabilities. Each SPM capability contributes to a key process and it indicates which maturity level this process has.

In addition to the artifact structure, the research methods used during the development of both artifacts are provided. At the left, the research methods that were used for developing the Reference framework are listed and at the right the method for developing the Maturity matrix are listed.

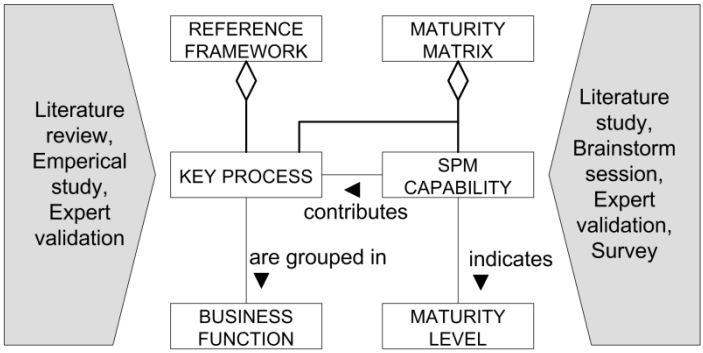


Figure 8-2. Artifacts and research methods

8.2.2 Data collection methods

This research was conducted with the following data collection methods:

Literature study. One of the sources for the capabilities, which are defined for each of the processes in the reference framework for SPM, is a literature study. This literature study was based on a multitude of papers describing specific processes within the field of SPM, e.g. Abramovici and Soeg (2002) and Clements and Northrop (2001).

Brainstorm session. A brainstorm session was conducted with experts from the scientific community to create the model. The session consists of two parts: 1) the capabilities themselves were determined; 2) the positions of the capabilities in the SPM maturity matrix were determined. The literature study was used as a basis for the brainstorm session.

Expert validation. An expert from practice validated the results of the brainstorm session: the capabilities themselves and their position within the SPM Maturity Matrix.

Survey. A final validation was conducted based on a survey with SPM experts from practice from all over the world. The goal of this survey was to validate the order of the capabilities relative to each other in the SPM Maturity Matrix.

8.3 Developing a maturity matrix

In this section, we first describe our choice for the type of maturity matrix we use, and then we describe its structure and the development process.

8.3.1 Variance of maturity models

Van Steenbergen, Brinkkemper and van den Berg (2007) recognize three variants of maturity models: 1) *staged 5-level models*, which distinguish five levels of maturity, which in turn have a number of focus areas that are defined specific to that level; 2) *continuous 5-level models*, which contain a number of focus areas, in each area the 5 levels are distinguished; and 3) *focus area oriented models*, in which each focus area has its own number of specific maturity level.

Most well-known maturity models are staged or continuous 5-level models, such as the Capability Maturity Model (Paulk, Curtis, Chrissis & Weber, 1993),

and its follow-up CMMI (CMMI Product Team, 2002). Earlier research into the improvement of SPM shows some shortcomings in these methods. CMMI for example, has been found too heavy to use by several organizations (Cusumano, 2004). And there are others who say that extensive software process improvement (SPI) frameworks, such as CMMI and ISO/IEC 15504 (ISO/IEC 15504, 1996) are too large to implement, or even comprehend (Kuilboer & Ashrafi , 2000; Reifer (2000). For example, a typical CMM SPI cycle can take between one and a half and two years to complete. It also requires large resources and long term commitment (Zahran, 1997), which can be a problem for small and medium companies. Another problem is that small and medium software companies often not only lack the funds required to implement many of the practices from CMM but also have to base their SPI initiatives on practices that do not apply to them (Brodman, & Johnson, 1994).

For the reasons above, we choose to develop a focus area oriented model, in order to make local analysis and incremental improvement possible. Similar model have been used for the testing domain (Koomen & Baarda, 2005) and the architecture domain (van Steenbergen, Brinkkemper & van den Berg, 2007).

8.3.2 Structure of the maturity matrix

In Table 8-1, the SPM maturity matrix is depicted.

Tabel 8-1. Maturity matrix structure

	0	1	2	3	4	5	6	7	8	9	10	11	12
RM1. Requirements gathering		A		B	←	→	C			D			
RM2. Requirements identification			A			B			C		↘		
RM3. Requirements organizing			A				B				C		
RP1. Requirements prioritization				A		B	C			D			
RP2. Requirements selection				A		B		C			D		
RP3. Release definition			A			B		C					
RP4. Release validation				A			B			C			D
RP5. Launch preparation		A				B					C		
RP6. Scope change management				A		B				C			
PR1. Theme identification			A		B		C						
PR2. Core asset identification						A			B				C
PR3. Roadmap construction			A			B				C			

PM1. Market trend identification			A		B				C				
PM2. Partnering & contracting			A				B		C				
PM3. Product lifecycle management			A			B			C				
PM4. Product line identification				A			B			C			

The matrix consists of columns and rows, which represent the two dimensions of the model. The columns 0 to 12 represent the different maturity levels for the model, where 0 is the lowest level of maturity and 12 the highest. The SPM key processes are represented by the rows and are divided into four groups (the business functions; 'Requirements management', 'Release planning', 'Product roadmapping', 'Portfolio management'). When a process is carried out at a certain maturity level it is called a capability. In Table 1, we can for example identify the capability Requirements gathering A, which is located on level 1. This capability is coded as RM1A and described as: "Ad hoc requirements gathering. Requirements are being gathered and registered."

Finally, two more concepts need to be introduced:

Intra-process capability dependency – This is the dependency of one capability within a certain key process to another capability in the same key process. In Table 1 this type of dependency is depicted with an arrow between RM1B and RM1C.

Inter-process capability dependency – Intra-process refers to the dependency of a capability in a certain key process to a capability in another key process. In Table 1 this is depicted with an arrow between RM1D and RM3C.

8.4 Developing a maturity matrix for SPM

We followed three main steps during the development of the maturity matrix:

1. Identification and description of capabilities.
A brainstorm session with four SPM experts was held to identify the capabilities. Two of them had extensive professional experience in SPM and the other two were researchers to SPM. During the session, it appeared that the maturity levels in the SPM processes follow two types of sequences.

The first sequence follows the natural hierarchy of software products (portfolio > product > release > requirement). The capability naming therefore is: (A) ad hoc, (B) release-based, (C) roadmap-based, and (D) portfolio-based. However, this hierarchy is not for all processes suitable. Therefore, we use a

second sequence, which can consist of (A) ad hoc, (B) organized, (C) integrated / externally oriented, (D) optimized. Please note that a sequence may consist of 3 or 4 capabilities. Short descriptions of the capabilities are included in Appendix A, based on Bekkers, van de Weerd, Brinkkemper and Mahieu (2008b).

2. Positioning the capabilities in the maturity matrix
The next step concerned positioning the capabilities in the matrix. By analyzing the inter- and intra-process capability dependencies, we decided the order of the capabilities in the matrix. For example, for the first capability in the process in Requirements identification (RM2A), it makes sense to have gathered and registered requirements, which is capability RM1A. Therefore, RM2A must be placed at least 1 level after RM1A. This activity resulted in a matrix of 13 levels (0-12). During the validation, we will find out whether this is the right size.

3. Validating the maturity matrix
The third and last step is the empirical validation of the matrix. In Section 4, we describe how we use a survey to validate the positions of the capabilities.

8.5 Empirical validation

8.5.1 Survey structure

Our survey consists of three parts: Introduction questions, general questions and capability questions. It starts with two introduction questions:

- Which SPM areas are you familiar with?
- How are you related to SPM?

The answers to these two questions determine which questions will be presented in the remainder of the survey. First, the respondent can choose which of the four SPM areas will be included in the survey. Only SPM areas of which the checkboxes are ticked, will be included in the survey. The second question is used to find out whether the respondent is a software product manager or another SPM professional. After the introduction question, some general questions are posed concerning company size, experience, etc.

The main structure of our survey is based on the four business functions that are defined in the reference framework for SPM: requirements management, release planning, product roadmapping, and portfolio management. For each function, we asked how our identified capabilities should be implemented in an organization. If we would ask to fill in a whole matrix per area, we would get

very large matrices. For example, the release planning area has 21 capabilities and 12 rows. During the first pilots it appeared this would cause a cognitive workload that was too high. Therefore, we decided to use another approach and divided the matrix in three sub matrices. In the first matrix, ranging from 1 to 6, capabilities A and B are covered. In the second matrix, ranging from 4 to 9, capabilities B and C are covered. Finally, if necessary, a third matrix is used to cover capabilities C and D. In Table 8-2, part of the matrix is depicted, showing the division in three separate matrices.

Tabel 8-2. Matrix division

	0	1	2	3	4	5	6	7	8	9	10	11	12
RM1. Requirements gathering													
RM2. Requirements identification													
RM3. Requirements organizing													

In Figure 8-3, we depict part of the survey in which all capabilities A and B of the Requirements Management process area are listed. The capabilities are listed on the left and the levels (1-6) in the middle. The last columns can be used to indicate whether the respondent has implemented the capability. The N/A option is for non-product managers that execute the survey.

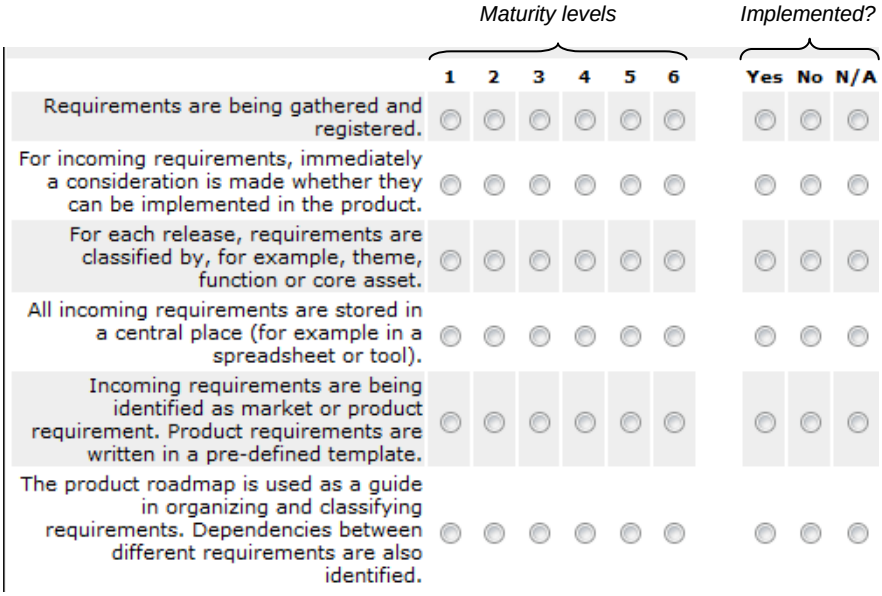


Figure 8-3. Survey questions Requirements Management

8.5.2 Data collection

In order to find respondents to the survey, we posted a message to several mailing lists. Below, you can find an overview of the member of these mailing lists:

- Netherlands product software mailing list: 175
- Netherlands SPM community: 68
- International SPM community: 176

In addition, during a national meeting among 20 Dutch product management professionals, the survey was promoted. Finally, the url of the survey was posted on the SPM community website[1]. Please note that the members of the different mailing lists, visitors to the meeting, and visitors of the website overlap.

The number of respondent to the survey is 61. Of these 61 only 32 were useful, i.e. they did not quit the survey after the first two pages. Not all questions have the same amount of responses. The respondents can make a choice in the beginning of the survey, for which they can indicate on which SPM area they can answer questions. Per SPM area, we have the following amounts of respondents:

- Requirements management: 31
- Release planning: 21
- Product roadmapping: 20
- Portfolio management: 16

The respondents originate mostly from the Netherlands (25). Other individuals originate from India, South Africa, Sweden, Switzerland and the USA. Ten of the respondents followed a course or study in SPM and one is certificated in SPM. In Table 8-3, some characteristics concerting the respondents function, company size and experience are provided.

Tabel 8-3. Respondents' characteristics

Function		**Company size**		**Experience**	
Product manager	23	Less than 50	10	0-2 years	8
Consultant	2	50 - 250	4	3-5 years	8
Researcher	2	More than 250	9	6-10 years	9
Other	4			More than 10 years	7

[1] http://www.softwareproductmanagement.org/

8.5.3 Data analysis

In this section, we will analyze the results from the survey. For each business function, we will describe the results of the survey and compare this with our initial maturity matrix. In principle, we will use the survey results to update our maturity matrix. However, we make one exception. Dependencies between capabilities in the different business functions were not part of the survey. Therefore, in case a capability is placed on a certain maturity level because of a dependency to a capability in another business function, we keep to the original sequence.

We use boxplots to give a graphical overview of the distribution of each capability, as is illustrated in Figure 8-4. These boxplots show the median, smallest and highest observations and the distribution over the quartiles. In addition, the outliers (if present) are shown. We use the medians as an indication of the maturity level of a capability, because we want to know which level was chosen most for a certain capability.

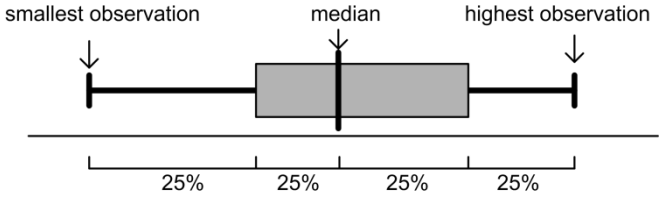


Figure 8-4. Boxplot

We list all capabilities within one business function in a boxplot. At the Y-axis, the different capabilities are listed and at the X-axis, the maturity levels are indicated. Please note that some means are not integers, but rational numbers. E.g. RM1B has a mean of 2,5. The reason for this is that the respondents had to enter these maturity levels twice (cf. Section 4.1). Sometimes this resulted in two different answers, of which we calculated the mean.

Requirements management
In Figure 8-5, the results of the Requirements management survey part are depicted in a box plot. In Table 8-4, the matrix is illustrated with the original results, and the deviations that were found when comparing it with the boxplot. We also indicate the status of a capability. "A" means that the result of the survey was the same as the original position, "A" indicates the result of the survey that was different from the original capability, and "A" shows the old position of the capability. Finally, in Table 8-4, we also give the 95%

confidence intervals. For example, the upper and lower bounds for the 95% confidence interval of RM1B are 2,5 and 3,5.

We use the medians as indicated in the boxplots to deduct the sequence of the capabilities. The confidence intervals are used to draw conclusions about the agreement of the respondents. For example, the chance that a respondent places capability RM1A on maturity level 1 is 95%. For capability RM1B, there is a 95% chance that a capability is placed between level 2,5 and level 3,5. The smaller the interval is, the higher the agreement of the respondents.

Several issues are noteworthy. Firstly, the intra-process capability dependencies are the same; all capabilities are positioned in the sequence A, B, C, (D). For the inter-process capability dependencies we see some differences that we will implement in the final matrix. The confidence intervals of the capabilities are relatively small.

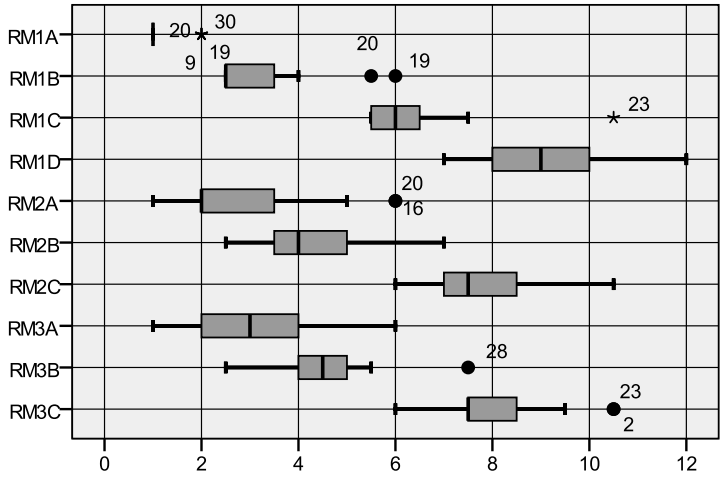


Figure 8-5. Boxplot Requirements management

Tabel 8-4. Requirements management matrix

RM1		A 1-1		B 2,5-3,5			C	C 5,5-6,5		D 8-10			
RM2			A 2-3			B 3,5-4,5			C 7-8,5				
RM3			A		A 3-4		B 4-5		C 7,5-8		C		

Release planning

In Figure 8-6 and Table 8-5, the results of the Release planning part of the survey are illustrated and compared with the original matrix. Again, the intra-process capability dependencies are the same. However, in the inter-process capability dependencies we see many small deviations. Most of which have been incorporated into the matrix without comments. One issue stands out: The prioritizing and selection capabilities (RP1A and RP2A) are implemented earlier than all other capabilities. Apparently, the respondents consider these activities as the minimum that a product manager should implement. Another thing that is noticeable is that the C and D capabilities have larger confidential intervals than the A and B capabilities.

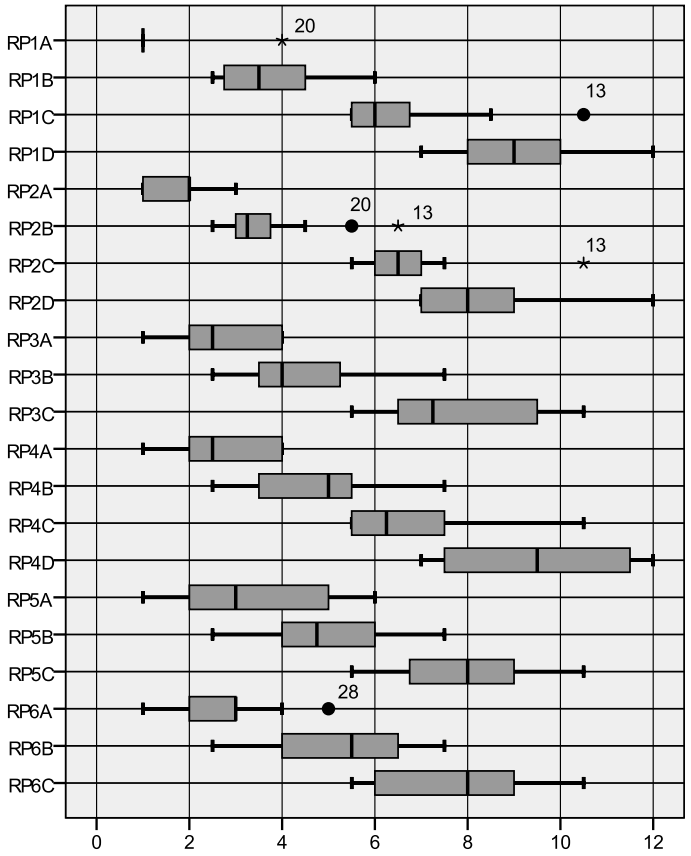


Figure 8-6. Boxplot Release planning

Tabel 8-5. Release planning matrix

RP1		A 1-1		A		B 3-4,5	C	C 5,5-6,5		D		D 8-10	
RP2			A 1-2	A		B 3-3,5		C	C 6-7		D 7-9		
RP3				A 2-4		B	B 3,5-5	C		C 6,5-9,5			
RP4				A 2-4			B 3,5-5,5		C 5,5-7,5	C		D	D 8-11
RP5		A			A 2-5	B		B 4-6			C 7-9		
RP6					A 2-3	B			B 4-6,5	C	C 6-9		

Product roadmapping

The results of the Product roadmapping part of the survey are illustrated in Figure 8-7 and compared with the original matrix in Table 8-6. Also for Product roadmapping, the intra-process capability dependencies are analogous to the original matrix. In the inter-process capability dependencies we see some deviations that will be included in the matrix. However, the position of the B-capabilities will not be changed. According to the boxplot, all B-capabilities should be placed on the same maturity level. However, these capabilities are highly dependent of capabilities in other business functions. For example, PR2B ("External sources are investigated based on ROI in the search for core asset acquisition") can only be done if the right Product line management capability of the Portfolio management function is implemented. Therefore, we decided not to change the position of these capabilities.

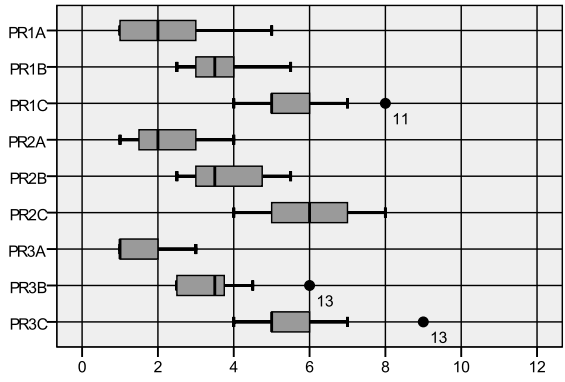


Figure 8-7. Boxplot Product roadmapping

Tabel 8-6. Product roadmapping matrix

PR1			A	A 1-3	B 3-4		C			C 5-6			
PR2				A 1-3		A			B 3-5				C 5-7
PR3			A 1-2			B 2,5-4				C 5-6			

Portfolio management

Finally, in Figure 8-8 and Table 8-7 the results of the Portfolio management part of the survey are illustrated and compared with the original matrix.

Again, the intra-process capability dependencies are the same as in the original matrix. In the inter-process capability dependencies we see a few small deviations, which we will include in the matrix. One of these deviations, however, will not be included: The survey results show that the respondents position the C-capability of Product line identification (PM4C) at the same level as PM3C. However, this capability is dependent of the B-capability in the Product roadmapping function. Therefore, we will keep the C-capability in its place. Finally, we observe that the confidence intervals of the portfolio management capability are larger compared to the other matrices.

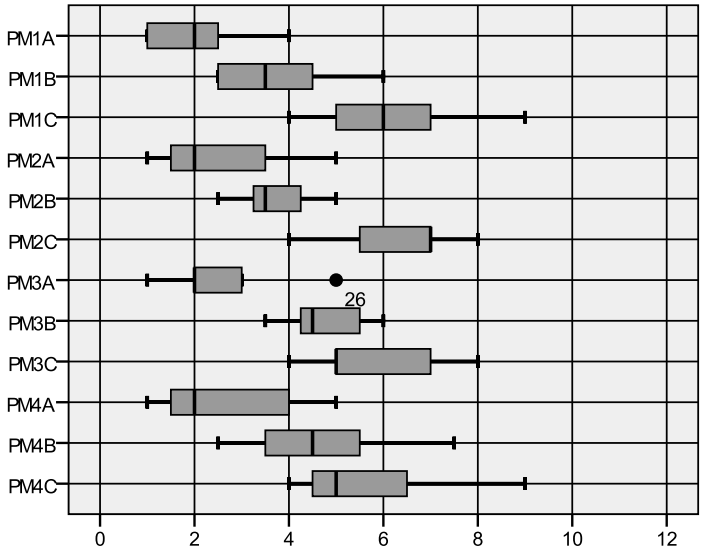


Figure 8-8. Boxplot Portfolio management

Tabel 8-7. Portfolio management matrix

PM1		A 1-3		B 2,5-4,5				C 5-7				
PM2		A 1-4		B 3-4,5		B		C	C 5-7			
PM3		A 2-3			B 4-5,5		C 5-7	C				
PM4		A 1-4	A		B 3,5-6	B			C 4-7			

8.5.4 Results

In Table 8-8, the final maturity matrix for SPM is presented. As can be seen in the table, the number of levels stayed the same. We also show the score for one of the respondents in the benchmark questions. This respondent works for a company with less than 50 employees from the Netherlands. He did not have any education or certification in SPM. The overall SPM maturity level is determined by checking for which maturity level all capabilities (and the preceding capabilities) are implemented. As can be seen in the table, the maturity level in this case is 3. To advance to a higher maturity level, say level 4, work needs to be done in PM1 and PM2.

Tabel 8-8. Maturity matrix for SPM

	0	1	2	3	4	5	6	7	8	9	10	11	12
RM1. Requirements gathering		A		B				C		D			
RM2. Requirements identification			A			B			C				
RM3. Requirements organizing					A		B		C				
RP1. Requirements prioritization		A				B		C				D	
RP2. Requirements selection			A			B			C		D		
RP3. Release definition				A			B			C			
RP4. Release validation				A			B		C				D
RP5. Launch preparation					A			B			C		
RP6. Scope change management					A				B		C		
PR1. Theme identification				A	B					C			
PR2. Core asset identification				A					B				C
PR3. Roadmap construction			A			B				C			
PM1. Market trend identification			A		B				C				
PM2. Partnering & contracting			A		B					C			

PM3. Product lifecycle management			A			B		C					
PM4. Product line identification			A			B		C					

8.5.5 Threats to validity

Validity refers to the degree to which a response measures what we intend for it to measure (Wohlin et al, 1999). We identified three types of validity threats that are important to discuss, namely threats to conclusion validity, construct validity, and external validity. Internal validity is less important since this study is not about establishing a causal relationship. It is a descriptive study, in which we describe the mostly used implementation sequence of SPM capabilities.

Firstly, conclusion validity is the degree to which conclusions we reach about relationships in the data are reasonable. Important is the composition of participants and the statistical analysis. At the beginning of the survey, respondents were asked to indicate with which of the SPM they were familiar. The respondents only got to answer questions about these familiar areas. Therefore, we believe that the respondents were experienced enough to answer the questions they selected. Most of the respondents are not anonymously, since an email address is needed to send them the benchmark report. However, there is no reason to believe that they have not answered according to their best knowledge, since there is no social pressure to answer in a particular way. Therefore, we consider the conclusion validity not to be critical.

Construct validity concerns whether we measure the construct that we believe we measure. On the introduction page of the survey, we have provided all definitions on all important concepts used in the questionnaire. Hence, we minimized the threat that participants interpret concepts differently. However, it is not possible to completely eliminate this threat.

External validity concerns generalization of the results to other groups and contexts than the one studied. The survey was sent to different groups of SPM professionals, as is described in Section 4.2. The respondents are mainly product managers with varying experience, working for companies of varying sizes. Most of the respondents are from the Netherlands. Given the diverse background of the respondents concerning company size and experience, we believe that the validity is high within the product manager population in the Netherlands. Since we believe SPM is not particularly influenced by culture, we believe that we can also generalize the results to other countries. However, to be sure, more research is needed.

8.6 Implications and outlook

The SPM Maturity Model can be used to determine the maturity level, but also be as a means to achieve process improvement by using it in the Product Software Knowledge Infrastructure (PSKI) (van de Weerd, Versendaal & Brinkkemper, 2006). The process improvement method consists of four steps followed by a feedback loop. Firstly the current maturity level of an organization needs to be determined by filling in the SPM maturity matrix. Secondly, the optimal maturity level for the organization is determined; this is achieved by studying the situational factors (Bekkers, van de Weerd, Brinkkemper & Mahieu, 2008a) for the organization and its environment. Thirdly, the differences between the current and the optimal situation are analyzed and better suiting method fragments, selected from a knowledge base containing SPM method fragments, are suggested for the suboptimal processes. An incremental order to improve the processes is then suggested as an improvement guide for the organization. Finally a feedback loop incorporates the knowledge gained in this process into the knowledge base of method fragments.

8.7 Conclusion and future research

In this paper, we described the development of a maturity framework for SPM that can be used to assess an organization's current SPM capabilities and offer local, incremental improvements to the product manager. After developing the matrix, we used a survey in which 32 SPM professionals reported by indicating the right order in which SPM capabilities should be implemented in an organization. The results of this survey supported our initial positioning of the SPM capabilities, at least, when looking at the inter-process capability dependencies. Concerning the intra-process capability dependencies, several deviations were found, of which most of them have been used to improve the matrix. The maturity matrix can be more refined by carrying out more research on especially the capabilities with higher confidence intervals.

We believe that the maturity matrix for SPM is a valuable tool for process assessment and improvement in SPM. In future research, we will carry out case studies at different product software companies to further validate the matrix in SPM capability improvement.

CHAPTER 9

Using a knowledge infrastructure for incremental process improvement

The success of a software product depends on the quality of its Software Product Management (SPM) processes. In this research, a knowledge infrastructure is presented that supports incremental process improvement in SPM. By assessing a company's SPM processes in the areas of Requirements Management, Product Roadmapping, Release Planning and Requirements Management, we create a maturity profile that serves as a basis for process improvement. In this paper, we describe a comparative case study of three product software companies. For each company, a maturity profile is created, as well as a method advice. The results indicate that the knowledge infrastructure is able to create a useful method advice.

9.1 Introduction

More and more software companies have shifted their focus from developing customized software to developing software products as a standard product. This shift implies that these companies have to adapt their processes to the new situation. They have to turn their standard development processes into *Software Product Management* (SPM) processes. There is some scientific literature available on SPM, e.g. Gorchels (2000), Condon (2002), Ebert & de Man (2002), and Dver (2003). However, there is no extensive body of knowledge in this area, as can be found in, for example, software engineering. Due to a lack of education and documentation, acquiring SPM capabilities may cause difficulties.

In earlier research, we proposed the Product Software Knowledge Infrastructure (PSKI) in order to support companies in their growth in SPM maturity. In this research, we will use the PSKI to demonstrate three case studies in which, based on a company´s process need and situational factors, a

method advice is given. Furthermore, we will investigate whether the process advice that is produced is a useful advice.

In the following section, we will describe our research approach. Then, in section 3, we describe the knowledge infrastructure for process improvement in SPM. Section 4 presents the three cases in which three companies are assessed. Finally, in section 5, we present our conclusions and discussion.

9.2 Research approach

9.2.1 Research question and propositions

As explained in Section 1, we will use the PSKI in three case studies to research whether the PSKI can provide a useful method advice. This leads us to the following research question:

> *To what extent can a knowledge infrastructure, in which method engineering principles and the SPM maturity matrix are integrated, provide a useful method advice for product managers?*

The answer to this question is not a simple one. The 'extent' of success is difficult to measure, especially since we cannot analyze the effect of the implementation of the method advice. What we can measure is whether we interpreted the problems correctly and whether the product managers find the method advice useful. Hence, we can state the following propositions in this study:

> *Using the knowledge infrastructure for assessing a company's need results in a correct interpretation of a company's problem(s).*
>
> *Using the knowledge infrastructure for matching method fragments with maturity levels and situational factors results in a proper method advice.*

If the results indicate that both propositions are correct, we consider the method advice useful.

9.2.2 Case study design

The research question is addressed by carrying out a comparative case study. A comparative single case study is "a study in which (a) a small number of cases in their real life context are selected and (b) scores obtained from these cases are analyzed in a quantitative manner" (Dul & Hak, 2008).

We chose for a comparative case study for several reasons. First, a case study is a very useful research method when in-depth investigation is needed. Our research question cannot be solved by carrying out a survey because the question is too complex. On the other hand, setting up an experiment would be unrealistic in view of time and costs. Second, by incorporating comparison in the research design, we can increase the internal validity of the research (Hollifield & Coffey, 2006). In addition, multiple case studies are preferable compared to single case studies (Yin, 2004; Dul & Hak, 2008), since these lead to a higher validity and thereby generalizeability. Finally, by carrying out a comparative case study, patterns might be revealed that are difficult to spot in single case studies (Bryman, 1989).

The research approach that we follow in each case study is as follows

1. *Assessment*
   For each of the three cases, an assessment is carried out. During this assessment, an interview is done and product documentation is studied.
2. *Creating maturity profiles*
   Based on the assessment, we create three *maturity profiles*. A maturity profile illustrates the assessment score, as depicted in the maturity matrix. In addition, we provide the average maturity level, the minimum, maximum and variation.
3. *Create method advice*
   For each company, we identify the needed method increments and include this in an advice report for the companies.
4. *Advice response*
   The interviewees are asked to respond on the advice report.

9.2.3 Validity threats

In performing a case study, we have to take several validity threats into account (Yin, 2003). First, *construct validity* is realized by using multiple interviewees per case to ensure that problems related to subjectivity and bias of data are avoided.

The second threat is to the *external validity*, which is the extent to which the results of a study can be generalized to others. By using multiple case studies, we increased the generelizeability of the study and mitigated this threat.

Finally, the *reliability* of the case study is concerned with demonstrating that the results of the study can be replicated. We ensure this by maintaining a case study database that contains all the relevant information used in the case study. In addition, in all three case studies, we used the same case study protocol.

9.3 A knowledge infrastructure for process improvement in SPM

In this section, the construction of the knowledge infrastructure for process improvement in SPM is explained. First, we summarize our earlier research concerning the Product Software Knowledge Infrastructure (van de Weerd, Versendaal & Brinkkemper, 2006). Then, we give an overview of the maturity matrix as proposed in (van de Weerd, Bekkers & Brinkkemper, forthcoming). Thirdly, we present the maturity assessment that is used to obtain a maturity profile. Finally, we show how the three constructs are integrated.

9.3.1 The Product Software Knowledge Infrastructure

In Van de Weerd, Versendaal and Brinkkemper (2006), we proposed the Product Software Knowledge Infrastructure (PSKI). With this infrastructure, product software companies can obtain a custom-made advice that helps them to improve their processes. An overview of PSKI and its environment is presented in Figure 9-1. We illustrated the PS company at the left-hand side and the PSKI at the right-hand side.

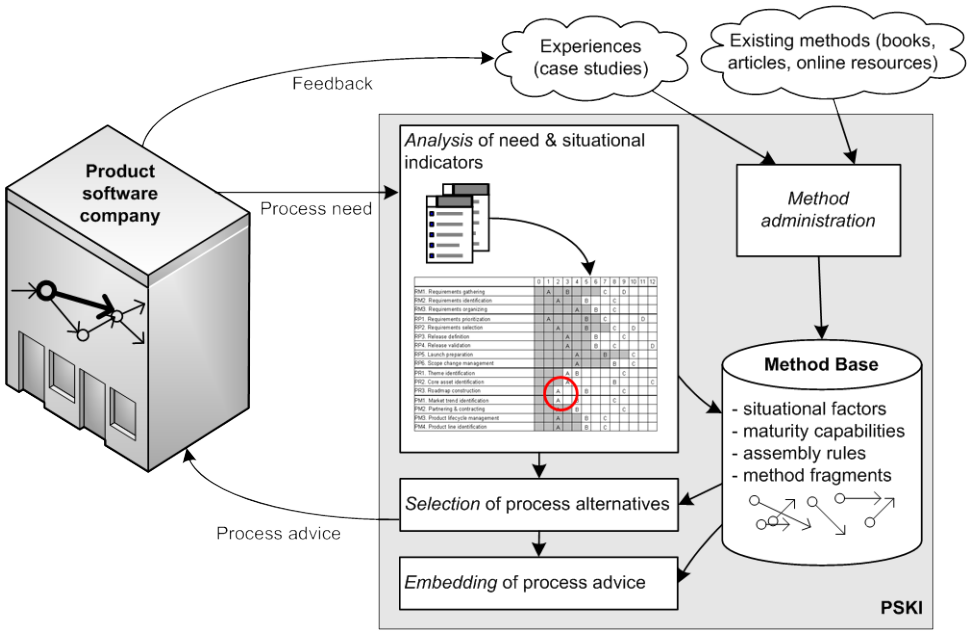


Figure 9-1. The Product Software Knowledge Infrastructure

The PSKI contains:

- method fragments, which are coherent parts of a method that contain part of the process and output of a method and the relations between them (Brinkkemper, 1996);
- situational factors that are relevant for the company and its context (Brinkkemper, 1996);
- capabilities that a company possesses, each related to its own maturity level (van de Weerd, Versendaal & Brinkkemper, 2006);
- assembly rules, which describe the way in which method fragments should be assembled to a new method (Brinkkemper, Saeki & Harmsen, 1999).

The process starts with the analysis of the company's need and its situational factors. Based on the results of this step, process alternatives from the method base are selected. These process alternatives have the form of method fragments. Finally, the selected method fragment is embedded in the organization by constructing a process advice for the company.

The first step, Analysis of need & situational indicators, is further detailed in the figure. In this first step of the method improvement process, a maturity profile is created, based on an assessment. The circle indicates which process has the lowest maturity level. In the next sections, we will explain both the maturity matrix and the assessment.

9.3.2 Maturity matrix

In Van de Weerd, Bekkers and Brinkkemper (forthcoming), we proposed the maturity matrix for SPM. This maturity matrix can be used to assess an organization's current SPM capabilities and to offer local, incremental improvements to the product manager. The structure of the maturity matrix for SPM is based on the reference framework for SPM (van de Weerd, Brinkkemper, Nieuwenhuis, Versendaal and Brinkkemper, 2006).

The matrix consists of columns and rows, which represent the two dimensions of the model. The columns 0 to 12 represent the different maturity levels for the model, where 0 is the lowest level of maturity and 12 the highest. The SPM key processes are represented by the rows and are divided into four groups: Requirements management, Release planning, Product roadmapping, and Portfolio management. When a process is carried out at a certain maturity level it is called a capability. In Table 1, we can for example identify the capability Requirements gathering A, which is located on level 1. This capability is coded as RM1A and described as: "Ad hoc requirements gathering. Requirements are being gathered and registered."

Table 9-1. Maturity matrix for Software Product Management

	0	1	2	3	4	5	6	7	8	9	10	11	12
RM1. Requirements gathering		A		B				C		D			
RM2. Requirements identification			A			B			C				
RM3. Requirements organizing					A		B		C				
RP1. Requirements prioritization		A				B		C				D	
RP2. Requirements selection			A			B			C		D		
RP3. Release definition				A			B			C			
RP4. Release validation				A			B		C				D
RP5. Launch preparation					A			B			C		
RP6. Scope change management					A				B		C		
PR1. Theme identification				A	B					C			
PR2. Core asset identification				A					B				C
PR3. Roadmap construction			A			B				C			
PM1. Market trend identification			A		B				C				
PM2. Partnering & contracting			A		B					C			
PM3. Product lifecycle management			A			B		C					
PM4. Product line identification			A			B		C					

9.3.3 Assessment instrument

To be able to fill in a company's maturity profile in the maturity matrix, we developed an assessment instrument. This instrument is a semi-structured interview containing the following parts:

Introduction. In the first part of the interview, the product managers are asked the following question: Where in the SPM domain are the problems or challenges in your organization? We present this question in order to find out whether the assessment itself instigates a change in the way how the interviewees consider their own SPM processes. After posing the question, the reference framework for SPM was presented, in order to make the product managers acquainted with the terminology that was used later in the interview.

General questions. The second part of the interview started with asking whether the presentation of the reference framework for SPM had changed anything about their thoughts on the problems and challenges their company

faced in SPM. In addition, several questions were asked concerning their experience with SPM.

Situational factors. In this part of the interview, the product managers were asked to give the values for a list of situational factors, divided over five categories: business unit characteristics, customer characteristics, market characteristics, product characteristics, and stakeholder involvement. These situational factors have been developed in earlier research (Bekkers, van de Weerd, Brinkkemper & Mahieu, 2008a) and give information about the company and its context. The situational factors of all three companies are listed in Appendix B.

*Maturity assessment***.** The last part of the interview contains the actual assessment. For each process in the reference framework for SPM, capabilities have been defined (van de Weerd, Bekkers & Brinkkemper, forthcoming), as is presented in Appendix A. For each capability, the product manager had to answer whether or not this capability was implemented in the company. We can further explain this by showing an example of the SPM-process 'Requirements gathering'. This process has the following capabilities:

A. Ad hoc requirements gathering. Requirements are being gathered and registered.
B. Organized requirements gathering. All incoming requirements are stored in a central place (for example in a spreadsheet).
C. Integrated requirements gathering. Internal and external stakeholders use various channels (e.g. website, helpdesk) to submit requirements, which are automatically stored in a central database.
D. Optimized requirements gathering. Requirements are being gathered and stored automatically, for example by data mining in emails, setting out competitor spiders, etc.

The other capabilities can be found in Van de Weerd, Bekkers and Brinkkemper (forthcoming).

9.4 Cases

In this section, we will present the three cases that we used to illustrate the PSKI. All three case companies are from The Netherlands. For privacy reasons, the names are coded. For each case, we will describe the challenges that were indicated by the interviewees; the situational factors that characterize the company and its environment; and the maturity profile that was created based on the assessment.

9.4.1 Case 1: CashComp

CashComp produces products for electronic payment traffic. It is the largest of the three case companies; currently over 350 employees are working at CashComp. Three product managers of CashComp were interviewed in one session of two hours. The interview session took place in May, 2009.

The following challenges concerning SPM were mentioned:

C1.1. Interactivity of the requirements management phase. Requirements are too often sent back and forth between product managers and other department.
C1.2. Decision-making on stopping the support of old product releases.
C1.3. Making clear who is responsible for which process.
C1.4. Changing from a company that developed customized software to a company that develops product software.

Important situational factors are:

- Size of business unit team: 50
- Development philosophy: waterfall
- Product tolerance for defects: low
- New requirements rate: 150 per year

Especially in product toleranc CashComp is quite extreme compared to other companies. For electronic payment traffic software it is crucial that no defects are found in the software after it is implemented. Therefore, the company has implemented an extensive internal testing procedure. In addition, they have to certify each product at an external party.

Based on the assessment, we created a maturity profile for, as is shown in Table 9-2. The maturity level of CashComp is slightly higher than both other case companies. Looking at CashComp's situational factors, this is not surprising. CashComp works with large business teams, which makes it necessary to have formal SPM processes in place. In addition, the fault tolerance of the product is very low. When their electronic payment products are implemented at the customers, there should be no more defects in the products.

Table 9-2. SPM maturity profile CashComp

	0	1	2	3	4	5	6	7	8	9	10	11	12
RM1. Requirements gathering		A		B				C		D			
RM2. Requirements identification			A			B			C				
RM3. Requirements organizing					A		B		C				
RP1. Requirements prioritization		A				B		C				D	
RP2. Requirements selection			A			B			C		D		
RP3. Release definition				A			B			C			
RP4. Release validation				A			B		C				D
RP5. Launch preparation					A			B			C		
RP6. Scope change management					A				B		C		
PR1. Theme identification				A	B					C			
PR2. Core asset identification				A					B				C
PR3. Roadmap construction			A			B				C			
PM1. Market trend identification			A		B				C				
PM2. Partnering & contracting			A		B					C			
PM3. Product lifecycle management			A			B		C					
PM4. Product line identification			A			B		C					

For a few processes, CashComp has a low score, namely Requirements identification, Core asset identification, and Partnering & contracting. For the latter, we have to make a few reservations. Partnering & contracting is not carried out by the product managers, but by another department. The interviewees were not entirely sure about the capabilities on this process. This may be a reason for the relatively low score. Therefore, we do not include this process in our recommendations.

Based on the challenges that were indicated by the three product managers, the situational factors and the maturity matrix, we provided CashComp with two recommendations:

- Advance the product roadmapping process
  - Affecting SPM processes PR2, PR3 and RM3
  - Related to challenges C1.4
- Distinguish between market and product requirements
  - Affecting SPM process RM2
  - Related to challenges C1.1 and C1.4

9.4.2 Case 2: ProcessComp

ProcessComp is a company with 15 employees that develop an internet application that makes it possible to deduct information from a company's information systems, in order to analyze, manage and support the company's processes. For the assessment, the owner and a former product manager (current position is solution consultant) were interviewed concurrently in a session of two hours. The interview session took place in May 2009.

The following challenges concerning SPM were mentioned by the interviewees:

C2.1. Managing various channels that are used to submit bugs and requirements.

C2.2. Finding a balance between the various types of change requests: bugs, requirements, opportunities and SLA requirements.

C2.3. Handling scope creep in a release and in particularly making a well-informed choice between delaying the release, cutting functionality, or hiring more developers.

C2.4. Preparing the product launch. There is a need for a structured process in which trainings and documentation are provided.

Important situational factors are:

- Size of business unit team: 15 fte
- New requirements rate: 100 per year

Based on the assessment, we created a maturity profile for ProcessComp, depicted in Table 9-3. ProcessComp has a varying maturity profile. Concerning requirements management, the company scores relatively high on requirements gathering, in contrast to the lower scores on requirements identification and organization. In the release planning area, the release definition process scores low, but the requirements prioritization and requirements selection activities have the highest score. Product roadmapping was a difficult area to assess at ProcessComp. Themes are being registered and used in a tool to group requirements. In addition, several sources are being used to obtain core assets (capability B). However, no central list is being kept that contains an overview of these core assets (capability A). Therefore, we set the score on Core asset identification on level 2 and product roadmapping on level 4. Finally, Portfolio management has a high score on Partnering & contracting and Product line identification. Market trend identification needs more attention. Product lifecycle management is less important, since ProcessComp has one product.

Table 9-3. SPM maturity profile ProcessComp

	0	1	2	3	4	5	6	7	8	9	10	11	12
RM1. Requirements gathering		A		B				C		D			
RM2. Requirements identification			A			B			C				
RM3. Requirements organizing					A		B		C				
RP1. Requirements prioritization		A				B		C				D	
RP2. Requirements selection			A			B			C		D		
RP3. Release definition				A			B			C			
RP4. Release validation				A			B		C				D
RP5. Launch preparation					A			B			C		
RP6. Scope change management					A				B		C		
PR1. Theme identification				A	B					C			
PR2. Core asset identification				A					B				C
PR3. Roadmap construction			A			B				C			
PM1. Market trend identification			A		B				C				
PM2. Partnering & contracting			A		B					C			
PM3. Product lifecycle management			A			B		C					
PM4. Product line identification			A			B		C					

Based on the challenges that were indicated by ProcessComp' employees, the situational factors and the maturity matrix, we provided them with two recommendations:

- Distinguish between market and product requirements
  - Affecting SPM process RM2
  - Related to challenges C.2.1 and C.2.2
- Advance the product roadmapping process
  - Affecting SPM processes PR2, PR3 and RM3
  - Related to challenges C.2.2 and C.2.3

9.4.3 Case 3: SupplyComp

SupplyComp is a vendor in supply chain systems. Their main product is an application for order, transport and crossdock management.

Three employees of SupplyComp were interviewed: the CEO, a solution engineer and a solution consultant. All three were interviewed concurrently in one session of two hours. The interview session took place in April, 2009.

During the first part of the interview, the following challenges concerning SPM were mentioned:

C3.1. Properly registering requirements
C3.2. Up-to-date product and technical documentation
C3.3. Linking requirements to software artifacts
C3.4. Ensuring requirements traceability
C3.5. Changing from a company that developed customized software to a company that develops product software.
C3.6. Overview of which deliverables have to be produced in which part of the SPM process.

The second part of the interview concerned the situational factors. SupplyComp is located in the Netherlands and has approximately 15 employees. The company operates in a growing market and its customers are mainly large companies. The product's age is 4 years and the expected product lifetime is over 15 years. SupplyComp receives about 250 new requirements and 40 defects per year from different stakeholders.

Finally, the assessment was carried out. Based on the assessment, we created a maturity profile, as is shown in Table 9-4. SupplyComp is a young company with 15 employees. The calculated maturity profile corresponds to their situation. They recently shifted from developing customized software to developing product software. They have one main product, which makes most portfolio management processes less urgent. On the other hand, SupplyComp is operating in an emerging market. In order to be competitive and grow, it is necessary to improve certain SPM processes. Especially the requirements identification process and the product roadmapping area needs attention.

Based on the challenges that we indicated by SupplyComp's employees, the situational factors and the maturity matrix, we provided them with three recommendations:

- Distinguish between market and product requirements
  - Affecting SPM process RM2
  - Related to challenges C3.1 and C3.5
- Improve requirements documentation and traceability
  - Affecting SPM process RM3
  - Related to challenges C3.1, C3.3, C3.4 and C3.6
- Implement a product roadmapping process
  - Affecting SPM processes PR3 and RM3
  - Related to challenges C3.2, C3.3, C3.5 and C3.6

Table 9-4. SPM maturity profile SupplyComp

	0	1	2	3	4	5	6	7	8	9	10	11	12
RM1. Requirements gathering		A		B				C		D			
RM2. Requirements identification			A			B			C				
RM3. Requirements organizing					A		B		C				
RP1. Requirements prioritization		A				B		C				D	
RP2. Requirements selection			A			B			C		D		
RP3. Release definition				A			B			C			
RP4. Release validation				A			B		C				D
RP5. Launch preparation					A			B			C		
RP6. Scope change management					A				B		C		
PR1. Theme identification				A	B					C			
PR2. Core asset identification				A					B				C
PR3. Roadmap construction			A			B				C			
PM1. Market trend identification			A		B				C				
PM2. Partnering & contracting			A		B					C			
PM3. Product lifecycle management			A			B		C					
PM4. Product line identification			A			B		C					

9.4.4 Elaboration on the recommendations

In this section, we will elaborate on the proposed recommendations. Two of the recommendations will be explained here. The other two are too large to explain in detail. A short summary and an overview of the method in the form of a process-deliverable diagram will be provided. In addition, we include a link to the web page where the method is explained.

Distinguish between market and product requirements
All three companies score low in the requirements identification process. This may lead to problems in other processes, such as requirements organizing and prioritization. We propose a categorization into *market requirements* and *product requirements*. This process is described by Natt och Dag, Gervasi, Brinkkemper and Regnell (2004). Market requirements are requirements that are directly copied from the requirement raiser. In case a customer raised a certain requirement, it is documented as market requirement in the terminology of the customer. In addition to customers, also the other internal and external stakeholders can submit requirements. Product requirements, on the other hand, are written by the product manager. These requirements can be a redefinition of

one or multiple market requirements. Product requirements should have clear descriptions and a comparable estimated development workload.

In Figure 9-2, we depicted the method fragment related to the recommendation. At the left-hand side the activities, including roles, are shown. At the right-hand side, the deliverables are illustrated. Observe that market and product requirements have a zero to many relation. Some market requirements will not be linked to product requirements, and it is possible to create product requirements without linking them to market requirements.

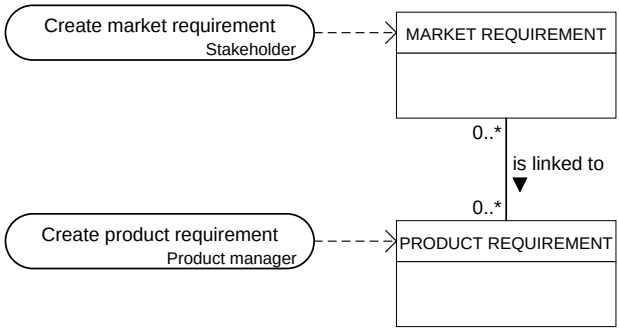


Figure 9-2. Method fragment: categorize requirements in market and product requirements

Improve requirements documentation and traceability
SupplyComp scored low on requirements identification and organization. In addition they indicated that they had difficulties with tracing requirement throughout the lifecycle of a product release. In order to improve the requirements documentation, product requirements need to be described via a predefined format. A process-deliverable diagram of this method fragment is illustrated in Figure 9-3.

The fields specification, dependencies, reference, and test realize traceability of the product requirements. These properties are created by later processes, such as ‘Update product requirements status’. By filling in and regularly updating these fields, a better overview of the requirements and their statuses is obtained.

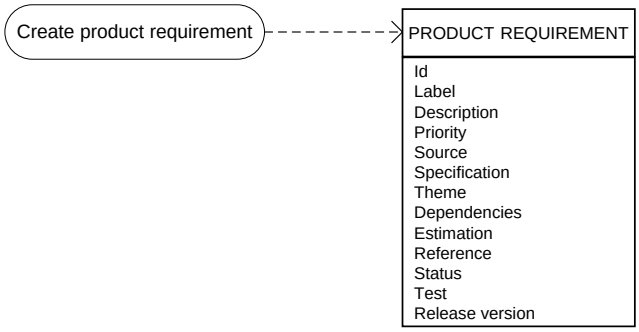


Figure 9-3. Method fragment: Create product requirement

Implement a product roadmapping process

SupplyComp started as a company that developed software for a few large customers. At this stage of the software business, it is very clear what needs to be accomplished with the product. However, as ideas and ambitions grow, and the product is developed for an entire market instead of a few customers, a roadmap is needed to manage expectations, plans, themes and core assets of a product. The roadmap provides a layout of the product releases to come over the following year(s).

Vähäniitty, Lassenius and Rautiainen (2002) describe a roadmapping process for small software companies. In Figure 9-5, this process and its deliverables are visualized. In the main activity Roadmapping, two roadmap development methods are expressed: the time-critical development method and the function-critical development method. Due to space limitations, we refer for more information to our SPM knowledge infrastructure[1].

[1] http://www.cs.uu.nl/wiki/bin/view/Spm/SmallSoftwareBusinesses

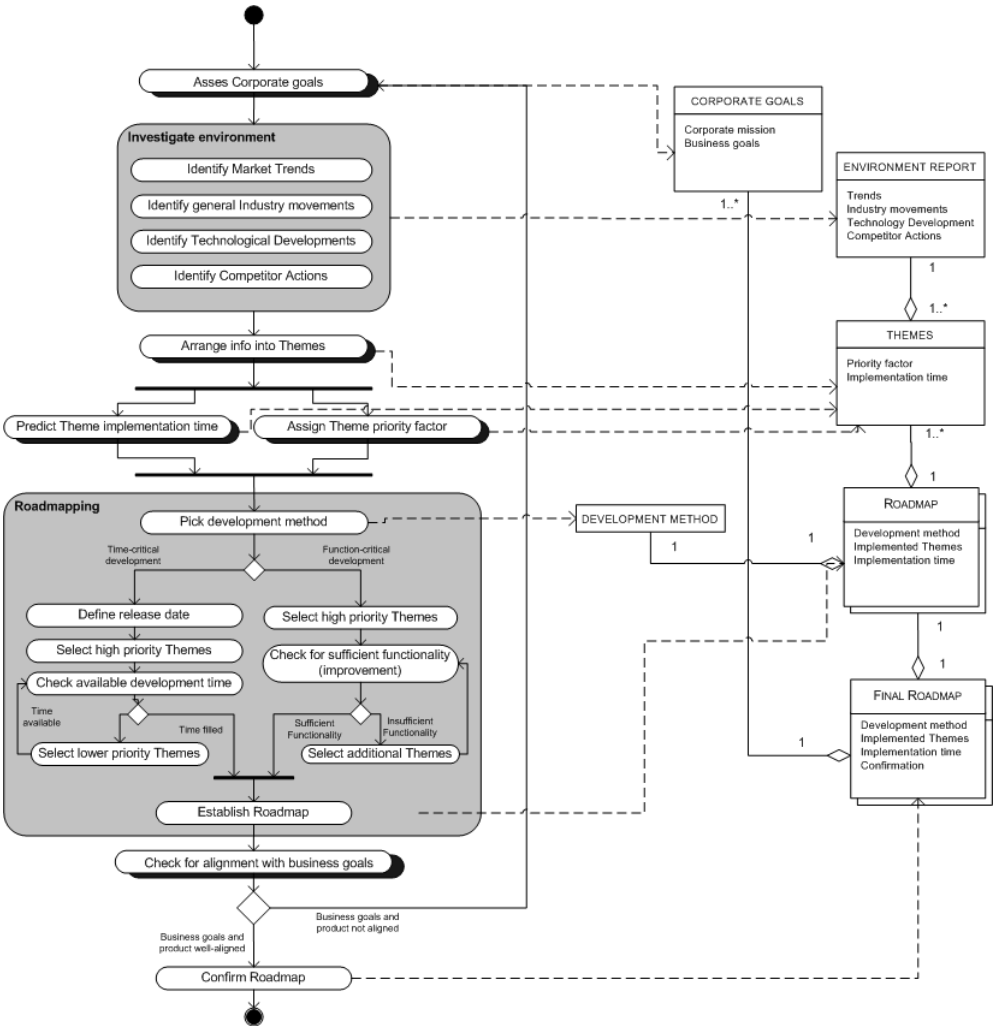


Figure 9-4. Process-deliverable diagram of product roadmapping in small software businesses

Advance the product roadmapping process

CashComp and ProcessComp both have a product roadmapping process in place. Multiple internal stakeholders are involved in the roadmapping process. What misses in both companies is managing the core assets in the products. Therefore, we propose to make the roadmapping process more structured and include the core asset identification activity in it. This will also bring the requirements organizing activity to a higher level.

We propose to implement the T-Plan 'fast start' method (Phaal, Farrukh, Mitchell & Probert, 2003), as depicted in Figure 9-5.

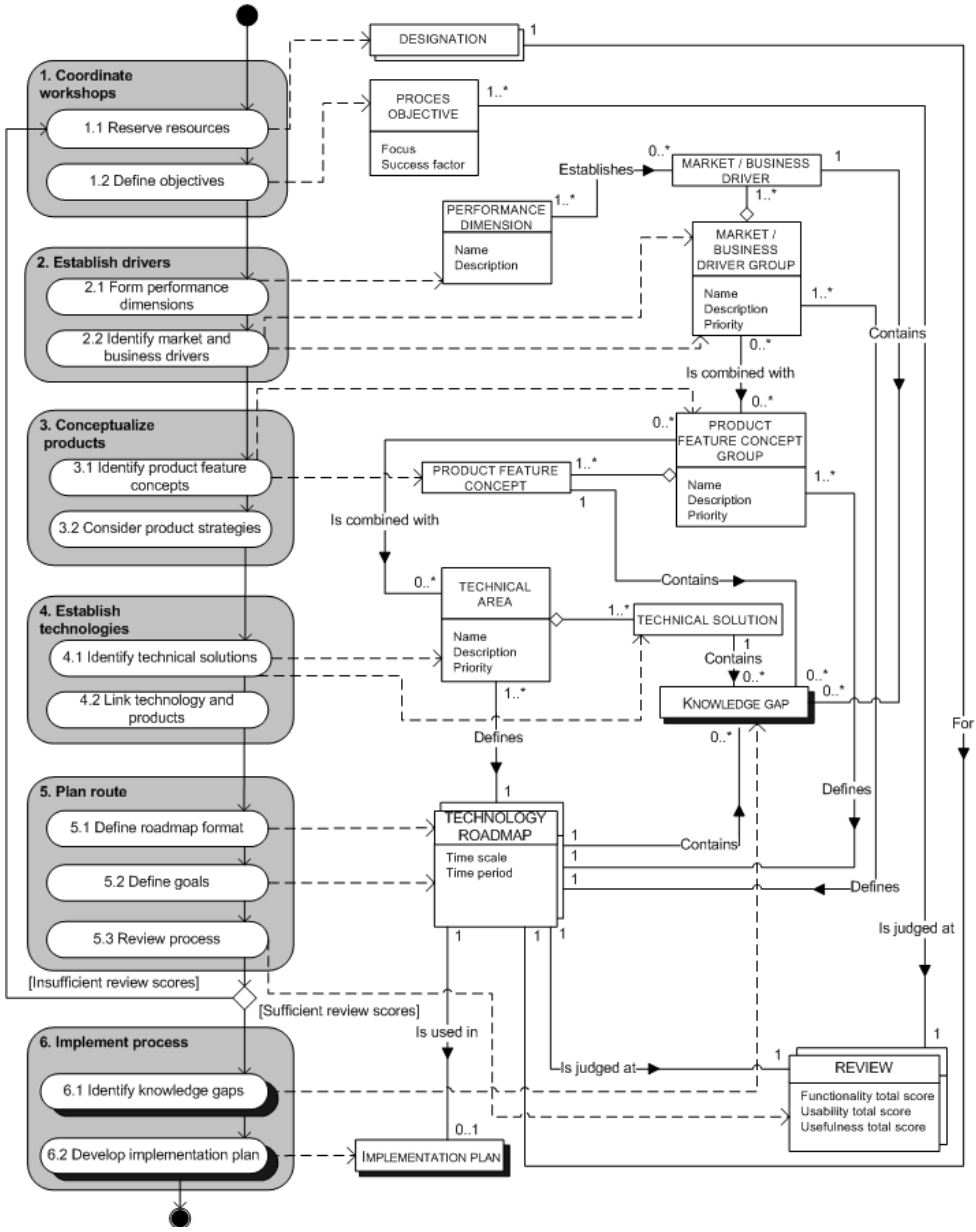


Figure 9-5. Process-deliverable diagram T-Plan roadmapping method

The T-Plan method consists of four workshops: identifying market and business drivers, conceptualizing the product(s), identifying current and future technologies and constructing the roadmap. In Figure 9-5, the four workshops can be recognized, and also two extra phases: the coordination phase and the implementation phase. The main deliverable of the method is the technology roadmap. For more details on the T-Plan method, we refer to our SPM knowledge infrastructure[1].

9.5 Results

9.5.1 Maturity profiles

In this section, we will compare the maturity profiles of the cases described in the previous section. We calculated the mean and the standard deviation of the achieved scores in the SPM processes, as can be viewed in Table 9-5. CashComp has the highest mean, followed by ProcessComp and SupplyComp. The standard deviations of CashComp and ProcessComp are similar. SupplyComp´s standard deviation is clearly lower, which indicates that the different scores tend to be closer to each other. This shows that SupplyComp's processes are more aligned with each other than the other the processes of the other two companies.

Table 9-5. Maturity profile statistics

	CashComp	**ProcessComp**	**SupplyComp**
Mean	8.31	6.75	4.44
Standard deviation	3.32	3.36	2.28
Lowest score	2	2	2
Highest scrore	12	12	9

In Figure 9-6, an overview is provided of the scores of the three companies on the four process areas. The figure shows that CashComp and PorcessComp follow the same pattern: Release Planning (RP) scores highest, followed by Portfolio Management (PM), Requirements Management (RM) and Product Roadmapping (PR). For SupplyComp this order is different. In this company, there is a clear difference between Release Planning and Requirements Management that both score above level 5, and Portfolio Management and Product Roadmapping that both score below level 3. This last maturity profile is

[1] http://www.cs.uu.nl/wiki/bin/view/Spm/FastStart

typical for young product software companies: on the more operational level the maturity is higher than on the more strategic level.

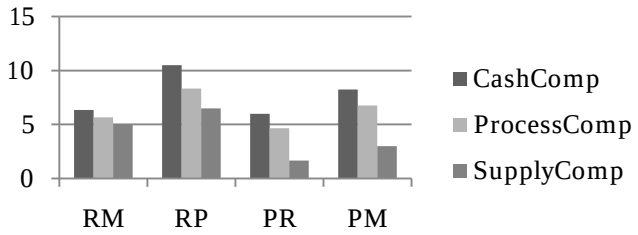


Figure 9-6. Maturity profile per process area

9.5.2 Advice response

After sending the advice report to the case companies, we asked them the following questions:

1. Do you recognize your company in the maturity profile, with respect to a) requirements management, b) release planning, c) product roadmapping, and d) portfolio management?
2. Did you have any plans to improve your SPM processes? Did the assessment provoke a change in these plans with respect to process choice or time path?
3. What are your plans with the recommendations that we provided?

CashComp answered positively on the first question. The product managers recognized the maturity profile as depicted in the maturity matrix for all process areas. Furthermore, they indicated that they were already focusing on improving their requirements management processes, and that, also after the assessment, this will be their first priority. In addition, several employees are going to follow a course on SPM. This will improve the communication about SPM. After the course they will pay attention to the other recommendations.

The second company that responded to the method advice is ProcessComp. Concerning Requirements Management, ProcessComp scored lower than they had expected. They stated that a categorization procedure for requirements was in use, which should lead to a higher score. However, this procedure (using the categories Need to have, Nice to have, Wish to have and Opportunity driven queues) is a prioritization method. The process requiremenrs prioritization (part of Release planning) received the maximum score. Hence, this misfit between maturity profile and company expectation seems to be based on a

misunderstanding of maturity matrix. For the Release Planning area, ProcessComp recognized the profile except on the processes of Release definition (they expected higher) and Launch preparation (they expected lower). Concerning the Release definition process, again the requirements categorization procedure was mentioned. However, this is also not part of the release definition procedure, so, again, we hold the misunderstanding of the maturity matrix accountable for this. For Product Roadmapping, ProcessComp recognized the maturity profile. Finally, for Portfolio Management, the maturity profile was also as expected, except on the Market trend identification, where they expected to score higher. Concerning the second question, ProcessComp indicated that "the recommendations clearly show where in the SPM area additional gains can be reached". They were planning to focus on the launch preparation. However, the method advice shows that there are other areas where more advantage can be gained, such as the distinction between market and product requirements. They indicated that they planned to implement this in their organization. Finally, the interviewees from ProcessComp were exploring opportunities to incorporate the T-Plan roadmapping process in their SPM practice.

Finally, SupplyComp responded to the advice. They answered positively on the first question. The interviewees recognized the maturity profile as depicted in the maturity matrix. SupplyComp was already working on their SPM processes before they were assessed. Two years ago they made a change from developing custom-made software to developing product software. In order to cope with this, they were working on the areas requirements management and release planning. SupplyComp planned to focus on release planning the following year, but they indicated that, based on the method advice, they will shift their focus to requirements management and product roadmapping. As a concluding remark, they state: "We will use the recommendations to guide our SPM improvement process in 2009".

9.6 Conclusion and discussion

9.6.1 Conclusion

In this paper, we presented three case companies that were willing to take our SPM assessment. For each company, we created a maturity profile that can be used to clarify the less mature processes within the SPM practice. For all companies, we created a recommendation, based on their maturity profile and situational factors.

In section 2, we stated the following research question:

To what extent can a knowledge infrastructure, in which method engineering principles and the SPM maturity matrix are integrated, provide a useful method advice for product managers?

We defined two propositions that are used to answer the research question:

Using the knowledge infrastructure for assessing a company's need results in a correct interpretation of a company's problem(s).

All three companies recognized their company in the maturity profile, except ProcessComp that indicated a few small differences. However, this was largely caused by a misunderstanding of the matrix. Therefore, we hold this proposition to be true.

Using the knowledge infrastructure for matching method fragments with maturity levels and situational factors results in a proper method advice.

All three companies indicated that they would use the method advice to improve their SPM processes. Two of them even indicated that the recommendation caused a shift of focus concerning the SPM processes that need attention. Hence, we can also say that the second proposition is true.

Based on the propositions, the answer to the research question is that the knowledge infrastructure, as explained in this research, does provide a useful method advice for product managers.

9.6.2 Discussion and further research

A few remarks should be made. Firstly, we would like to point out the learning effect that arose during the assessment. During the interviews, we asked the interviewees before and after the assessment what they thought were the weak spots in their SPM practice. In two out of the three cases the interviewees answered differently after taking the assessment. Apparently, the questions and statement about the different SPM processes created some form of awareness.

Another interesting finding in the study is the comparison of the maturity profiles. After comparing the means of the different process areas, it appeared that two of them were similar, and the third was different. We have a strong indication that young companies have the lowest score in the Product roadmapping and Portfolio management areas, when compared to the other two areas. We believe this occurs because young companies focus more on the operational SPM processes than the strategic SPM processes. However, further research is needed to find out which factors influence this maturity profile.

Concerning the validity of the research, we found that some of the interviewees had the urge to get the highest score. To prevent this, the interviewer asked them to elaborate further on the concerning capabilities. In the future, the assessment can be improved by asking the interviewees for evidence. For example, when the interviewee indicates that they have a formal scope change management process in place, we can ask whether a document with a description of the scope change process can be provided.

Currently, the PSKI is not automated. It is still a concept, of which several sub concepts, such as the assessment, maturity matrix and SPM knowledge infrastructure have (partly) been carried out. Especially in the area of method storage and assembly, further research is needed.

CHAPTER 10

Conclusion and outlook

The main research question in this dissertation is:

> *How can product software companies improve the maturity of their product management processes?*

Our answer to this research question is by developing the vision of a so-called Product Software Knowledge Infrastructure (PSKI). In the different chapters, the concept of the PSKI was presented, a reference framework to structure method knowledge in the SPM domain was developed and validated, and an instrument to assess a company's current maturity was introduced, validated and tested. In the following sections, the findings on the research question will be further explained. Then, the implications in a broader perspective will be described. Finally, the limitations of this research are discussed, resulting in future research.

10.1 Research questions

The research in this dissertation is positioned around three research questions. In this section, the findings of each research question and its sub-questions are presented.

RQ1. *Which main functions and stakeholders can be identified in the Software Product Management domain?*

In answer to this research question, the reference framework for Software Product Management was developed. By performing field interviews and discussions with product managers and by doing a literature review on (software) product management, we were able to identify the stakeholders and activities that are most important in SPM. Furthermore, we provided an overview of contemporary literature on software product management. By carrying out a case study, we found confirmation of the validity of the identified context, processes and relations in the reference framework for software product

management. Since its publication, the reference framework for SPM has inspired several other authors, e.g. Berander (2007), Kittlaus and Clough (2009). In addition, the framework is used for educational and consultancy purposes. More on this is described in the Implications section.

RQ2. *How can incremental method engineering be implemented in a knowledge infrastructure?*

Incremental method engineering is a suitable technique to use for process improvement in knowledge infrastructures. By using a meta-modeling technique, method fragments can be analyzed and stored in a method base. In addition, formalizing method increments makes it possible to reuse them later when assembling a process advice. RQ2 is subdivided into three sub questions:

RQ2.1. How can we model activities and deliverables in order to reuse them for situational method engineering purposes? The development of the meta-modeling technique process-deliverable diagrams (PDDs) provided the answer to this research question. After evaluating several approaches for meta-modeling, PDDs were found to be the most adequate modeling technique. PDDs consist of activities and deliverables and can be used for different purposes. In chapter 4, we showed two cases, in which the PDDs are used. The first one is method evolution analysis and the second one is assembly-based method engineering. For both appliances, PDDs have proven to be effective means for the meta-modeling of methods. In addition, PDDs have been used in method comparison and for educational purposes.

RQ2.2. How can product software companies improve their software product management methods in an evolutionary way, using method fragment increments? In answer to this research question, a formal approach to incremental process improvement was presented. This approach can serve as an instrument to improve a company's SPM processes in an evolutionary way. The approach makes use of formalized method increments, which can be used in assembling a method advice. In addition, the root-cause analysis technique was presented as a means to structurally analyze a company's process need. By applying this analysis in a case study, it was found that the approach and the corresponding root-cause map can be of great value in the support of incremental method evolution.

RQ2.3. Which method increment types occur in incremental method evolution, and which general increment drivers can be identified? In Chapter 6, a retrospective case study was carried out on incremental method evolution in a global SPM setting. The results of this case study show that all increments

found in the retrospective case study can be adequately modeled with the 18 method increment types that we identified in Chapter 6. In addition to the validation of the generic method increment types, increment drivers, which are the general principles behind method increments, were studied. Four increment drivers were found: development management, business management, departmental interfacing, and certification.

RQ3. How can we support process improvement in a knowledge infrastructure?

By analyzing a company's process need and mapping this with the method fragments in the knowledge infrastructure, a method advice can be assembled. By using a capability-based approach, i.e. assessing a company's capabilities in order to find the areas that can be improved, the method increments can be small and local. RQ3 consists of three sub questions.

RQ3.1. How can we use a knowledge infrastructure to support incremental method evolution? The answer to this research question is the presentation of an approach for incremental method evolution. This approach provides the means by which SPM methods can be inserted into existing processes, based on method engineering principles. This vision on process improvement combines a capability-based approach with problem-based aspects. In Chapter 7, it is explained how this approach can be implemented in the PSKI. In addition, it is demonstrated how method increments can be generated, based on the situational and capability assessment answers.

RQ3.2. How can we design a maturity matrix for software product management? In the last part of this dissertation, the maturity framework for SPM is presented. This maturity matrix can be used to assess a company's current SPM capabilities and offers local, incremental improvements to the product manager. After developing the matrix, it was validated in a survey in which 32 SPM professionals participated. The product managers were asked to indicate the right order in which SPM capabilities should be implemented. The results were then compared with the initial matrix. The findings of the survey supported the initial positioning of the SPM capabilities for the most part. The differences were used to improve the matrix.

RQ3.3. To what extent can a knowledge infrastructure, in which method engineering principles and the SPM maturity matrix are integrated, provide a useful method advice for product managers? In answering this research question, we found three case companies that were suitable to take our SPM assessment. For each company, we created a maturity profile that can be used to clarify the less mature processes within the SPM practice. Furthermore, we

created a method advice for each company, based on their maturity profile and situational factors. After reading the method advice, all three companies indicated that they would use the method advice to improve their SPM processes. Two of them even indicated that the recommendation caused a shift in focus concerning the SPM processes that need attention. Therefore, we concluded that the answer to the research question is that the knowledge infrastructure does provide a useful method advice for product managers.

10.2 Implications

10.2.1 Maturing the software product management domain

The research carried out in this dissertation is an important step in positioning the industrial domain of SPM in scientific research. By describing the relation between the different functions and stakeholders concerning SPM, and by positioning them in a reference framework, we contributed to the body of knowledge on SPM. Since the publication of the framework (van de Weerd, Brinkkemper, Nieuwenhuis, Versendaal & Bijlsma, 2006), several other researchers have used the framework to position their research (e.g. Berander, 2007; Kittlaus & Clough, 2009; Bekkers, van de Weerd, Brinkkemper & Mahieu, 2008a).

The reference framework for SPM has contributed to an increase in the availability of SPM knowledge for product software companies. By creating a knowledge infrastructure in the form of a website[1], SPM practitioners can browse, read and download method descriptions, white papers, examples and templates for SPM deliverables. This knowledge infrastructure is a first step to maturing the SPM practice.

Another result of our work on SPM is the education program we set up. In 2005, at the start of this dissertation research, virtually no education programs or working groups on SPM existed in The Netherlands and the rest of Europe. In 2006, a workgroup on Requirements Management was set up in Utrecht, in which researchers and practitioners were involved in meetings and discussions concerning requirements management in a product software setting. In 2007, this workgroup was changed into a course in SPM, where the reference framework is used to structure the course program. Currently, two regular courses and three in-company courses including consultancy sessions have been

[1] http://www.softwareproductmanagement.org/

held, and a new edition of the SPM course is scheduled next autumn. The regular courses are typically followed by 14 product managers and consist of 10 meetings of four hours, in which all aspects of SPM are covered. In addition, some other initiatives concerning SPM have been set up. A software product managers' network has been initiated, of which the members regularly set up meetings to exchange knowledge. Furthermore, a joint initiative from the Netherlands, Germany, Switzerland, and Sweden is working on a certification program for product managers in software companies.

10.2.2 Requirements management versus software product management

In the presented framework for SPM, requirements management is identified as one of the four main process areas. In a strict sense, requirements management only entails the activities of gathering, identifying and organizing requirements. However, in most research, requirements management is used in a broader sense. It also involves, among others, prioritization, validation, and scope change activities. In this perspective, we can say that SPM is a more elaborate version of traditional requirements management. As more and more companies are shifting from developing customized software to developing product software, their requirements management activities also change. A product roadmap is needed to plan the releases of the following years, requirements have to be traced throughout the release lifecycle, and make-or-buy decisions have to be made on the different components of the product. We reserve the term requirements management strictly for gathering, identifying and organizing requirements. Hence, the entire picture, including also the release planning, product roadmapping and portfolio management processes, is referred to as SPM.

SPM does not always comply with the classical view of requirements management. The main conferences and journals in the area mainly focus on requirements management for customized software. However, in order to mature the research community around requirements management, we contend that SPM should be adopted as a legitimate scientific research area.

10.2.3 Method engineering

By proposing a meta-modeling technique based on UML class diagrams and UML activity diagrams we were able to support the method engineering studies we carried out in this dissertation. This technique, in which Process-Deliverable

Diagrams (PDDs) are created, has proven to be an effective means of modeling methods. PDDs have since been applied in various research papers on method engineering, especially in the creation of new methods assembled from fragments of other methods (cf. Luinenburg, Jansen, Souer, Brinkkemper & van de Weerd, 2008; van de Weerd, de Weerd & Brinkkemper, 2007; Levantakis, Helms & Spruit, 2008), but also in other research domains, such as in information systems research methodology and business planning (cf. Jansen, 2008; van Ee, van Duinkerken).

10.3 Limitations and future research

In the last section of this chapter, the limitations of this research are described. In addition, directions for future research are provided.

10.3.1 Software product management

One of the main contributions of this dissertation research is the reference framework for SPM. Currently, the framework has been validated via expert reviews and various case studies. All of the case study companies were from the Netherlands. In addition, we were invited by German and Swiss practitioners to present the reference framework at their requirements management workgroup. The framework was perceived as very useful and applicable. However, we are not sure whether the results can be generalized to product software companies in all countries. The United States, for example, have a more marketing-oriented perspective on SPM (see e.g. www.productmarketing.com/). In future research, it would be interesting to test the reference framework in multiple countries.

10.3.2 Product software knowledge infrastructure

The PSKI, as presented in Chapter 2, and further explained in Chapters 7, 8 and 9, is still a limitedly worked-out concept. Several parts o the infrastructure have been detailed, such as the assessment and the maturity matrix. However, it is not yet fully automated. Future research in this area touches three main topics. First, the linking between the situational factors, maturity levels and method fragments need to be further investigated. Situational factors can be used to determine a company's optimal maturity level and to select those method fragments that fit in with the rest of the process. A start has been made in this research by Bekkers, van de Weerd, Brinkkemper and Mahieu (2008b).

Secondly, the storage of method fragments needs more attention. In Chapter 4, a formalization of method fragments and method increments was proposed. Based on this, the method fragment structure can be implemented in a CAME tool, for example MetaEdit+. By adding extra information to the stored method fragments, they can easily be retrieved after assessing the company's need.

Finally, some research challenges lie in the assembly of the method advice. When, based on the assessment, a method fragment is selected, it has to be implemented in the company's existing processes. In addition, an implementation plan should be provided. To be able to automate this process, more research is needed.

References

Abramovici, M., & Soeg, O.C. (2002). *Status and Development Trends of Product Lifecycle Management Systems*, Ruhr-University Bochum, Chair of IT in Mechanical Engineering (ITM), Germany.

Abran, A., Moore, J. W., Bourque, P., Dupuis, R., & Tripp, L. L. (2004). *SWEBOK: Guide to the Software Engineering Body of Knowledge*. Los Alamitos, CA: IEEE Computer Society.

Ahn, Y. W., Ahn, H. J, & Park, S. J. (2003). Knowledge and case-based reasoning for customization of software processes - A hybrid approach. *International Journal of Software Engineering and Knowledge Engineering, 13*(3), 293-312.

Akker, M. van den, Brinkkemper, S., Diepen, G. van, & Versendaal, J. (2005). Flexible release planning using integer linear programming. *Proceedings of the 11th International Workshop on Requirements Engineering for Software Quality,* 13-14 June 2005, Porto, Portugal, Essener Informatik Beitrage, Band 10.

Alur, D., Crupi, J., & Malks, D. (2001). *Core J2EE Patterns*, Upper Saddle River, NJ, Prentice Hall PTR.

Ardis, M., Daley, N., Hoffman, D. M., Siy, H., & Weiss, D. (2000). Software product lines: a case study. *Software - Practice and Experience, 30*(7), 825–847.

Aydin, M.N., & Harmsen, F. (2002). Making a method work for a project situation in the context of CMM. *Proceedings of the 14th International Conference on Product Focused Software Process Improvement*, Rovaniemi, Finland, 158-171.

Bandinelli, S. C., Fuggetta, A., & Ghezzi, C. (1993). Software process model evolution in the SPADE environment. *IEEE Transaction on Software Engineering, 19*(12), 1128-1144.

Bareisa, E., Karciauskas, E., & Blazauskas, T.: (2005) Development of case tools for software process improvement. *Information Technology and Control, 34,* 181-187.

Beecham, S., Hall, T., & Rainer, A. (2005). Defining a requirements process improvement model. *Software Quality Journal, 13*, 247–279

Bekkers, W., Weerd, I. van de, Brinkkemper, S., & Mahieu, A. (2008a). The influence of situational factors in software product management: an empirical study. *Proceedings of the 21st International Workshop on Product Management,* Barcelona, Spain, 41-48.

Bekkers, W., Weerd, I. van de, Brinkkemper, S., & Mahieu, A. (2008b). *The Relevance of Situational Factors in Software Product Management.* Technical report UU-CS-2008-016, Utrecht, The Netherlands: Universiteit Utrecht.

Bensabat, Goldstein, & Mead (1987). The case research strategy in studies of information systems. *MIS Quarterly 11*, 369-386.

Berander, P., & Andrews, A. (2005). Requirements Prioritization. In: A. Aurum and C. Wohlin (Eds.), *Engineering and Managing Software Requirements* (pp. 69-94). Heidelberg, Germany: Springer-Verlag.

Berander, P. (2007). *Evolving Prioritization for Software Product Management.* Doctoral Dissertation Series No. 2007:07, Blekinge Institute of Technology, 2007.

Berardi, D., Cali, A., Calvanese, D., De Giacomo, G. (2005). Reasoning on UML class diagrams. *Artificial Intelligence, 168,* 70 - 118.

Bodoff, S., Green, D., Haase, K., Jendrock, E., Pawlan, M., & Stearns, B. (2002). *The J2EE Tutorial*. Boston: Addison Wesley Professional.

Bonaccorsi, A., & Lipparini, A. (1994). Strategic partnerships in new product development: an Italian case study. *Journal of Production Innovation Management, 11*(2), 134–145.

Booch, G., Rumbaugh, J., Jacobson, I. (1999). The *Unified Modeling Language User Guide.* Redwood City, CA: Addison Wesley Longman Publishing.

Bosch, J. (1999). Product-line architectures in industry: a case study. *Proceedings of the 21st International Conference on Software engineering,* Los Angeles, CA, USA, 544-554.

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

Brinkkemper, S. (2000). Method engineering with web-enabled methods. In: S. Brinkkemper, E. Lindencrona and A. Sølvberg (Eds.), *Information Systems Engineering: State of the Art and Research Themes* (pp. 123-133). Heidelberg, Germany: Springer-Verlag.

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

Bryman, A. (1989). *Research Methods and Organization Studies*. London: Unwin Hyman..

Bourque, P., & Dupuis, R. (Eds.) (2004). *Guide to the Software Engineering Body of Knowledge.* Los Alamitos, CA: IEEE Computer Society.

Brodman, J. G., & Johnson, D. L. (1994). What small businesses and small organization say about the CMM. *Proceedings of the 16th International Conference on Software Engineering*, Sorrento, Italy, 247-262.

Brownsword, L., & Clements, P. (1996). *A Case Study in Successful Product Line Development,* Technical Report CMU/SEI-96-TR-016. Pittsburgh, KS: Carnegie Mellon.

Bryson, J. R. (1997). Business service firms, service space and the management of change. *Entrepreneurship & Regional Development, 9*(2), 93-112.

Carlshamre, P. (2002). Release planning in market-driven software product development provoking and understanding. *Requirements Engineering, 7*(3), 139-151.

Carlshamre, P., & Regnell, B. (2000). Requirements lifecycle management and release planning in market-driven requirements engineering processes. *Proceedings of the 11th International Workshop on Database and Expert Systems Applications*, Greenwich, London, 961-965.

Carmel, E. (1995). Time-to-completion factors in packaged software development. *Information & Software Technology, 37*, 515-520.

Carmel, E. (1999). *Global Software Teams,* Upper Saddle River, NK: Prentice Hall.

Clark, T., Evans, A., & Kent, S. (2001). The metamodelling language calculus: foundation semantics for UML. *Proceedings of the 4th International*

Conference on Fundamental Approaches to Software Engineering, FASE 2001, LNCS, 2029, 17 - 31.

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

CMMI Product Team (2002). Capability *Maturity Model Integration (CMMI), Version 1.1*. CMU/SEI-2002-TR-012.

Coleman, F. Hayes, F., & Bear, S. (1992). Introducing objectcharts or how to use statecharts on object-oriented design. *IEEE Transactions on Software Engineering 18*(1), 9-18.

Condon, D. (2002). *Software Product Management - Managing Software Development from Idea to Product to Marketing to Sales*. Boston: Aspatore Books.

Conradi, R., Fernström, C., & Fuggetta, A. (1993). A conceptual framework for evolving software processes. *ACM SIGSOFT Software Eng. Notes, 18*(4), 26-35.

Cooper, R. G., Edgett, S.J., & Kleinschmidt, E. J. (2001). Portfolio management for new product development: results of an industry practices study. *R&D Management 31*(4), 361-380.

Cronholm, S., & Ågerfalk, P.J. (1999). On the concept of method in information systems development. *Proceedings of the 22nd Information Systems Research Seminar in Scandinavia*, Keuruu, Finland, 229-236.

Cusumano, M. A. (2004). *The Business of Software: What Every Manager, Programmer, and Entrepreneur Must Know to Thrive and Survive in Good Times and Bad*. New York: Free Press.

Cusumano, M. A., & Selby, R. W. (1995). *Microsoft Secrets: How the World's Most Powerful Software Company Creates Technology, Shapes Markets, and Manages People.* New York: Free Press.

Damian, D., Zowghi, D., Vaidyanathasamy, L., & Pal. Y. (2004). An industrial case study of immediate benefits of requirements engineering process improvement at the Australian center for Unisys Software. *Empirical Software Engineering, 9*, 45-75.

Darke, P., Shanks, G, & Broadbent, M. (1998). Successfully completing case study research: combining rigour, relevance and pragmatism. *Information Systems Journal 8*, 273-289.

Debou, C., & Kuntzmann-Combelles, A. (2000). Linking software process improvement to business strategies: experiences from industry. *Software Process: Improvement and Practice 5*(1), 55-64.

Demirors , O., & Demirors , E. (1998). Software process improvement in a small organization: Difficulties and suggestions. *Proceedings of the 6th European Workshop Software Process Technology,* LNCS 1487, pp. 1-12.

Dolstra, E. (2003). Integrating software construction and software deployment. In B. Westfechtel and A. van der Hoek (Eds.), *11th International Workshop on Software Configuration Management (SCM-11)*, LNCS 2649, 102–117.

Duebendorfer , T., & Frei, S. (2009). *Why Silent Updates Boost Security.* ETH Tech Report 302. TIK, ETH Zurich.

Dver, A. S. (2003). *Software Product Management Essentials*. Tampa, FL: Anclote Press.

Ebert, C. (2006). Understanding the product lifecycle: four key requirements engineering techniques. *IEEE Software 23*(3), 19-25.

Ebert, C. (2007). The impacts of software product management. *Journal of Systems & Software 80*(6), 850-861.

Ebert, C., & Man, J. de (2002). e-R&D - Effectively managing process diversity. *Annals of Software Engineering 14*(1), 73 -91.

Ebert, C., & Smouts, M. (2003). Tricks and traps of initiating a product line concept in existing products. *Proceedings of the 25th International Conference on Software Engineering* Portland, OR, USA, 520-525.

Eisenhardt, K. M. (1989). Building theories from case study research. *The Academy of Management Review, 14*(4), 532-550.

Emam, K. el, Drouin, J. N., & Melo, W. (1998). *SPICE: The Theory and Practice of Software Process Improvement and Capability Determination*. Los Alamitos, CA: IEEE Computer Society.

Eoin, O. C., Holmstrom, H., Ågerfalk, P. J., & Fitzgerald, B. (2006). Exploring the assumed benefits of global software development. *Proceedings of the International Conference on Global Software Engineering*, Costão do Santinho, Florianópolis, Brazil, 159-168.

Firesmith, D., Henderson-Sellers, B. H., Graham, I., & Page-Jones, M. (1998). Open Modeling Language (OML) – Reference Manual. *SIGS reference library series*. Cambridge, UK: Cambridge University Press.

Fowler, M. (2003). *Patterns of Enterprise Application Architecture*. Boston: Addison-Wesley.

Fricker, S., Gorschek, T., & Glinz, M. (2008). Goal-oriented requirements communication in new product development. *Proceedings of the 2nd International Workshop on Software Product Management (IWSPM'08)*, Barcelona, Spain.

Glass, R. L., Vessey, I. & Ramesh, V. (2002). Research in software engineering: an analysis of the literature. *Information and Software Technology*, *44*, 491–506.

Gorchels, L. (2000). *The Product Manager's Handbook -The Complete Product Management Resource (2nd edition)*. Lincolnwood, IL: NTC Business Books.

Harmsen, F., Brinkkemper, S., & Oei, J. L. H. (1994). Situational method engineering for informational system project approaches. *Proceedings of the IFIP WG8.1 Working Conference on Methods and Associated Tools for the IS Life Cycle*, Amsterdam, The Netherlands,169-194.

Helms, R.W., Brinkkemper, S., Oosterom, J. van, & Nijs, F. de (2005). Knowledge entry maps: structuring of method knowledge in the industry. In: Kiyoki, Y., Kangasslo, H., Jaakkola, H., Henno, J. (eds.): *Proceedings of the 15th European Japanese Conference on Information Modelling and Knowledge Bases*, Tallinn, Estonia, 12-25.

Helms, R.W., & Reijsen, J. van (2008). Impact of Knowledge Network Structure on Group Performance of Knowledge Workers in a Product Software Company. In D. Harorimana & D. Watkins (Eds.), *Proceedings of the 9th European Conference on Knowledge Management* (pp. 289-296). UK: Academic Conferences.

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. (2003). Method engineering for OO systems development. *Communications of the ACM* 46, 73-78.

Herbsleb, J.D., & Goldenson, D.R. (1996). A Systematic Survey of CMM Experience and Results. *Proceedings of the 18th International Conference on Software Engineering*, Berlin, Germany, 323 – 330.

Hevner, A.R., March, S.T., Park, J., & Ram, S. (2004). Design science in information systems research. *MIS Quarterly, 28*, 75-105.

Hietala, J., Kontio, J., Jokinen, J.P., & Pyysiainen, J. (2004). Challenges of software product companies: results of a national survey in Finland.P*roceedings of the 10th IEEE International Symposium on Software Metrics*, Chicago, IL, USA, 232-243.

Hoek, A. van der (1997). Software release management. *Proceedings of the 6th European Software Engineering Conference together with the 5th ACM SIGSOFT Symposium on the Foundations of Software Engineering*, Zurich, Switzerland, 159–175.

Hollifield, C. A., & Coffey, A.J. (2005). Qualitative research in media management and economics. In A. B. Albarran, S. M. Chan-Olmsted, M. O. Wirth (Eds.), *Handbook of Media Management and Economics* (pp. 573-600). New York: Routledge.

Hunter, R., & McCallum, C. (1998). A visual approach to software process improvement. *Software Process Improvement '98*, Monte Carlo, Monaco.

Information Technology - Open Systems Interconnection (1994). *Basic Reference Model: The Basic Model*. International Standard, ISO/IEC 7498-1. 2nd ed. Geneva, Switzerland: ISO.

ISO 9001 (1997). *Quality systems - Model for Quality Assurance in Design, Development, Production, Installation and Servicing*, Geneva, Switzerland: ISO.

ISO/IEC-15504. *Information Technology - Software Process Assessment*. Technical Report - Type 2, Geneva, Switzerland: ISO.

Jacobson I, Booch G, & Rumbaugh J. (1999). *The Unified Software Development Process*. Redwood City, CA: Addison Wesley Longman Publishing Co., Inc.

Jansen, S. (2007). *Customer Configuration Updating in a Software Supply Network*. PhD Thesis. Utrecht University, The Netherlands.

Jansen, S. (2008). Applied Multi-Case Research in a Mixed-Method Research Project: Customer Configuration Updating Improvement. In A.C. Steel & L.A. Hakim (Eds.), *Information Systems Research Methods, Epistemology and Applications*.

Jansen, S., Ballintijn, G., Brinkkemper, S., & Nieuwland, A. Van (2006). Integrated development and maintenance for the release, delivery, deployment, and customization of product software: a case study in mass-market ERP software. *Journal of Software Maintenance and Evaluation: Research and Practice, 18*, 133-151.

Jeusfeld, M. A., Jarke, M., & Mylopoulos, J. (2009). *Metamodeling for Method Engineering*. Cambridge, MA: MIT Press.

Kahn, K. B. (2005). The *PDMA Handbook of New Product Development, Second Edition*. Hoboken, NJ: John Wiley & Sons.

Kang, K.C., Lee, J. & Donohoe, P. (2002). Feature-oriented product line engineering. *IEEE Software, 19,* 58-65.

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*, Skövde, Sweden.

Karlsson, J., & Ryan, K. (1997). A cost-value approach for prioritizing requirements. *IEEE Software 14*(5), 67-74.

Kelly, S., Lyytinen, K. & Rossi, M. (1996). MetaEdit+: a fully configurable multi-user and multi-tool CASE and CAME environment. *Proceedings of the 18th International Conference on Advanced Information Systems, LNCS 1080,* 1-21.

Kilpi, T. (1997). Product management challenge to software change process: preliminary results from three SMEs experiments. *Software Process - Improvement and Practice 3*(3), 165-175.

Kittlaus, H.-B., & Clough, P. N. (2009). *Software Product Management and Pricing: Key Success Factors for Software Organizations.* Berlin, Germany: Springer.

Koomen, T., & Baarda, R. (2005). *TMap Test Topics*, Den Bosch, The Netherlands: Tutein Nolthenius,

Koomen, T., & Pol, M. (1999): *Test Process Improvement: A Step-by-step Guide to Structured Testing.* Boston: Addison-Wesley Longman Publishing.

Krogh, G. von, & Cusumano, M. A. (2001). Three strategies for managing fast growth. *Sloan Management Review 42*(2), 53–61.

Krishnan, M. S. (1997). *Cost and Quality Considerations in Software Product Management*. PhD Dissertation Thesis, Carnegie Mellon University, Pennsylvania, USA.

Krzanik, L., & Simila, J. (1997). Is my software process improvement suitable for incremental deployment? *Proceedings of the 8th International*

Workshop on Software Technology and Engineering Practice. London, UK, 76-87.

Kuilboer, J. P., & Ashrafi, N. (2000). Software Process and Product Improvement: An Empirical Assessment", *Information and Software Technology, 42*(1), 27-34.

Kumar, K. & Welke, R. J. (1992). Methodology Engineering: a proposal for situation-specific methodology construction. In: W. W. Cottermand & W. A. Senn (Eds.), *Challenges and strategies for research in systems development* (pp. 257-269). New York: John Wiley & Sons, Inc.

Kwestel, M., Preston, M., & Plaster, G. (1998). *The Road to Success: How to Manage Growth*. New York: John Wiley & Sons.

Laqua, R. (2002). Concepts for a product line knowledge base & variability. *Proceedings of NetObjectDays 2002,* Erfurt, Germany, 30-39.

Lauesen, S. (2004). COTS tenders and integration requirements. *Proceedings of the 12th IEEE International Requirements Engineering Conference*, Kyoto, Japan, 166-175.

Leary, M. R. (1982). Hindsight distortion and the 1980 presidential election. *Personality and Social Psychology 8*(2), 257–263.

Lehtola, L., Kauppinen, M. (2006). Suitability of requirements prioritization methods for market-driven software product development. *Software Process: Improvement and Practice, 11*, 7-19.

Lehtola, L., Kauppinen, M.,& Kujala, S. (2005). Linking the business view to requirements engineering: long-term product planning by roadmapping. *Proceedings of the 13th IEEE international Conference on Requirements Engineering*, Paris, France, 439-446.

Leszak, M., Perry, D.E., Stoll, D. (2000). A case study in root cause defect analysis. *Proceedings of the 22nd International Conference on Software Engineering*, Limerick, Ireland, 428 -437.

Levantakis, T., Helms, R. W. & Spruit, M. R. (2008). Developing a reference method for knowledge auditing. In T. Yamagchi (Ed.), *Proceedings of the 7th Conference of Practical Aspects on Knowledge Management, 5345*. Lecture Notes in Artificial Intelligence (pp. 147-159). Berlin Heidelberg: Springer-Verlag.

Lok, R. H., & Walker, A. J. (1997). Automated tool support for an emerging international software process assessment standard. *Proceedings of the 3rd*

International Software Engineering Standards Symposium, Montreal, Canada, 25-35.

Luinenburg, L., Jansen, S., Souer, J., Brinkkemper, S. & Weerd, I. van de, (2008). An Approach to Creating Product Software Design Methods: The Case of Web Information Systems. Accepted for the WISM 2008 workshop in conjunction with ER 2008, Barcelona, Spain.

MacCormack, A. (2001). Product-development practices that work: how internet companies build software. *MIT Sloan Management Review, 42,* 75-84.

Man, J. de, & Ebert, C. (2003). A common product life cycle in global software development. *Proceedings of the 11th Annual International Workshop on Software Technology and Engineering Practice*, Amsterdam, The Netherlands, 16-21.

March, S. T., & Smith, G. F. (1995). Design and natural science research on information technology. *Decision Support Systems, 15,* 251-266.

Moon, M.,& Yeom, K. (2004). An approach to develop requirement as a core asset in product line, *Proceedings of the 8th International Conference, LNCS 3107,* 23 – 34.

Natt och Dag, J., Gervasi, V., Brinkkemper, S., Regnell, B. (2004). Speeding up requirements management in a product software company: linking customer wishes to product requirements through linguistic engineering. *Proceedings of the 12th IEEE International Requirements Engineering Conference*, Kyoto, Japan, 283-294.

Natt och Dag, J., Gervasi, V., Brinkkemper, S., & Regnell, B. (2005). A linguistic-engineering approach to large-scale requirements management. *IEEE Software 22*(1), 32-39.

Nawrocki, J. R., Walter, B., & Wojciechowski, A. (2002). Comparison of CMM level 2 and eXtreme programming. *Proceedings of the 7th European Conference on Software Quality, LNCS 2349,* 288-297.

Nejmeh, B. A., & Riddle, W. E. (2005). A framework for coping with process evolution. *Proceedings of the 1st International Software Process Workshop, LNCS 3340,* 302-316.

Nguyen, M. N., & Conradi, R., Towards a rigorous approach for managing process evolution. *Proceedings of the 5th European Workshop on Software Process Technology, LNCS 1149,* 18-35.

Niazi, M. K. (2002). Software Process Improvement: A Road to Success. *Proceedings of the 7th Australian Workshop on Requirements Engineering*, Melbourne, Australia, 125-139.

Niazi, M., Wilson, D., & Zowghi, D. (2005). A maturity model for the implementation of software process improvement: an empirical study. *Journal of Systems and Software, 7*, 155-172.

Nuseibeh, B., & Easterbrook, S. (2000). Requirements engineering: a roadmap. In A. C. W. Finkelstein (Ed.), *The Future of Software Engineering* (pp. 37-46). New York: ACM Press.

Object Management Group (2004). *UML 2.0 Superstructure Specification*. Document reference ptc/04-10-02.

Ocampo, A., & Münch, J. (2006). Process evolution supported by rationale: An empirical investigation of process changes. *Proceedings of the International Software Process Workshop and International Workshop on Software Process Simulation and Modeling, LNCS 3966*, 334-341.

Paulk, M. C. (1995). How ISO 9001 compares with the CMM. *IEEE Software, 12*, 74-83.

Paulk, M. C., Curtis, B., Chrissis, M. B., Weber, C.V. (1993). *Capability Maturity Model for Software (Version 1.1)* (SEI/CMU-93-TR-24, ADA263403). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University.

Paulk, M. C., Weber, C. V., Curtis, B., & Chrissis, M. B. (1995). *The Capability Maturity Model: Guidelines for Improving the Software Process*. Boston: Addison-Wesley Longman Publishing Co.

PMBOK, A Guide to the Project Management Body of Knowledge, first ed. (2000). Project Management Institute, Pennsylvania, USA.

Pohl, K., Böckle, G., Linden, F. van der (2005). *Software Product Line Engineering*. Heidelberg, Germany: Springer-Verlag.

Potts, C. (1995). Invented requirements and imagined customers: requirements engineering for off-the-shelf software. *Proceedings of the 2nd IEEE International Symposium on Requirements Engineering*, York, UK, 128-131.

Process Maturity Profile Software CMM 2005 End-Year Update (2006). Retrieved at March 22, 2008, from: http://www.sei.cmu.edu/appraisal-program/profile/pdf/ SW-CMM/2006marSwCMM.pdf.

Rainer, A., & Hall, T. (2002). Key success factors for implementing software process improvement: a maturity-based analysis. *Journal of Systems & Software 62*(2), 71-84.

Ralyté, J., Deneckère, R. & Rolland, C. (2003). Towards a generic model for situational method engineering. *Proceedings of 15th International Conference on Advanced. Information Systems Engineering, LNCS 2681*, 95-110.

Ralyté, J., Rolland, C.(2001). An assembly process model for method engineering. *Proceedings of the 17th International Conference on Advanced Information Systems Engineering, LNCS 2068*, 267–283.

Regnell, B., Beremark, P., & Eklundh, O. (1998). A market-driven requirements engineering process: results from an industrial process improvement programme. *Requirements Engineering, 3*, 121-129.

Regnell B., & Brinkkemper, S. (2005). Market-driven requirements engineering for software products. In: A. Aurum and C. Wohlin (Eds.), *Engineering and Managing Software Requirements* (pp. 287-308). Heidelberg, Germany: Springer-Verlag.

Regnell, B., Höst, M., Natt och Dag, J.N., Beremark, P., & Hjelm, T. (2001). An industrial case study on distributed prioritisation in market-driven requirements engineering for packaged software. *Requirements Engineering, 6*, 51-62.

Reifer, D. J. (2000). The CMMi: It's Formidable. *Journal of Systems and Software, 50*(2), 97-98.

Richardson, I., & Ryan, K. (2001). Software process improvements in a very small company. *Software Quality Professional 3*(2), 23-35.

Rolland, C., Prakash, N. (1996). A proposal for context-specific method engineering. *of the IFIP TC8, WG8. 1/8.2 working conference on method engineering on Method engineering: Principles of method construction and tool support: principles of method construction and tool support*, Atlanta, GA, USA, 191-208.

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

Root Cause Analysis Handbook: A Guide to Effective Incident Investigation (1999). Houston, TX: ABS Group Consulting, Inc.

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.

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

Saeki, M. (2000). Towards formal semantics of meta models. *Proceedings of the International Workshop on Model Engineering*, Nice, France,

Saeki, M. (2003). Embedding metrics into information systems development methods: an application of method engineering technique. *Proceedings of the 15th International Conference on Advanced Information Systems Engineering, LNCS 2681*, 374-389.

Saliu, M. O., & Ruhe, G. (2005). Supporting software release planning decisions for evolving systems. *Proceedings of the 29th Annual IEEE/NASA Software Engineering Workshop*, Greenbelt, MD, USA, 14-26.

Sawyer, P., Sommerville, I., & Viller, S. (1997). Requirements process improvement through the phased introduction of good practice. *Software Process Improvement and Practice, 3*, 19-34.

Schultz, G. (2007, May 18). The Windows Vista timeline. *TechRepublic*. Retrieved May 15, 2009, from http://articles.techrepublic.com.com/5100-10878_11-6073291.html.

Schwaber, K., & Beedle, M. (2002). *Agile Software Development with Scrum*, Upper Saddle River, NJ: Prentice-Hall.

Software Engineering Institute (2002). *Process Maturity Profile of the Software Community*. Carnegie Mellon University.

Sommerville, I. (2007). *Software Engineering.* Boston: Addison-Wesley.

Souer, J., Weerd, I. van de, Versendaal, J., & Brinkkemper, S. (2007). Situational requirements engineering for the development of content management system-based web applications. *International Journal of Web Engineering and Technology 3*(4), 420-440.

Stalk, G., Evans, P., & Shulman, L. (2001). Competing on capabilities.In: D. Barnes (Ed.), *Understanding Business: Processes*, New York: Routledge, 42-49.

Staples, M., Niazi, M., Jeffery, R., Abrahams, A., Byatt, P., & Murphy, R. (2007). An exploratory study of why organizations do not adopt CMMI. *Journal of Systems and Software, 80*(6), 883-895.

Steenbergen, M., Brinkkemper, S., & Berg, M. van den (2007). An instrument for the development of the enterprise architecture practice. *Proceedings of the 9th International Conference on Enterprise Information Systems*, Funchal, Madeira, Portugal, 14--22.

Stelzer, D., & Mellis, W. (1998). Success factors of organizational change in software process improvement. *Software Process Improvement and Practice 4*(4), 227-250.

Stone, B. (2009, May 13). *We Learned A Lot*. Retrieved May 15, 2009, from http://blog.twitter.com/2009/05/we-learned-lot.html.

Sweeney, A., & Bustard, D. W. (1997). Software process improvement: making it happen in practice. *Software Quality Journal, 6*, 265 - 274.

Szyperski, C. (2002). *Component Software: Beyond Object-Oriented Programming, 2nd ed*. Boston: Addison-Wesley Professional.

Terlouw, J., Terlouw, L., & Jansen, S. (2009). An assessment method for selecting a SOA delivery strategy: determining influencing factors and their value weights. *Proceedings of the Busital workshop*, Amsterdam, The Netherlands, 2009

Thiel, S. Ferber, S., Fischer, T., Hein, A. & Schlick, M. (2001). A case study in applying a product line approach for car periphery supervision systems. *Proceedings of In-Vehicle Software, SAE 2001 World Congress SP-1587*, Detroit, MI, 43-55.

Tolvanen, J.-P. (1998). *Incremental Method Engineering with Modeling Tools: Theoretical Principles and Empirical Evidence.* PhD Dissertation thesis, Jyväskylä Studies in Computer Science, Economics and Statistics 47, University of Jyväskylä, Finland.

Tolvanen, J. (2006). MetaEdit+: integrated modeling and metamodeling environment for domain-specific languages. *Proceedings of OOPSLA '06: Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*, Portland, OR, USA, 690--691.

Tsichritzis, D., & Klug, A. (1987). The ANSI/X3/SPARC DBMS Framework Report of the Study Group on Database Management Systems. *Information Systems, 1*, 173–191.

Unger, S. (2003). Ten marketing challenges that can make or break your business… and how to address them. *Productmarketing.com 1*(1).

Vaishnavi, V., & Kuechler, B. *Design Research in Information Systems*. Retrieved December 7, 2007, from AISWorld Net: http://www.isworld.org/Researchdesign/drisISworld.htm

Vähäniitty, J., Lassenius, C., & Rautiainen, K. (2002). An approach to product roadmapping in small software product businesses. *Conference Notes of Quality Connection - 7th European Conference on Software Quality*, Center for Excellence Finland, Helsinki, Finland, 12-13

Vähäniitty, J., & Rautiainen, K. (2005). Towards an approach for managing the development portfolio in small product-oriented software companies. *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*, Waikoloa, Hawaii, p. 314.

Versendaal J., & Brinkkemper, S. (2003). Benefits and success factors of buyer-owned electronic trading exchanges: procurement at Komatsu America Corporation. *Journal of Information Technology Cases and Applications 5*(4), 39-52.

Vise, D. A., & Malseed, M. (2005). *The Google Story*. New York: Delacorte Press.

Vliet, H. van (2008). *Software Engineering: Principles and Practice*. Hoboken, NJ: John Wiley & Sons.

Weerd, I, van de (2009). *IT and Software Product Management*, retrieved Februari 1, 2009, from IT- and Software Product Management: http://www.softwareproductmanagement.org/.

Weerd, I. van de, Bekkers, W., & Brinkkemper, S. (submitted). Developing a Maturity Matrix for Software Product Management.

Weerd, I. van de, Brinkkemper, S. (2008). Meta-modeling for situational analysis and design methods. *In: M. R. Syed & S. N. Syed (Eds.), Handbook of Research on Modern Systems Analysis and Design Technologies and Applications* (pp. 35-54). Hershey, PA: Information Science Reference.

Weerd, I. van de, Brinkkemper, S., Nieuwenhuis, R., Versendaal, J., & Bijlsma, L. (2006). Towards a reference framework for software product management. *Proceedings of the 14th International Requirements Engineering Conference*, Minneapolis/St. Paul, Minnesota, USA, 312-315

Weerd, I. van de , 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.

Weerd, I. van de , Brinkkemper, S., & Versendaal, J. (2006). *Incremental method evolution in requirements management: a case study at Baan 1994-2006*. Institute of Computing and Information Sciences, Utrecht University, Technical report UU-CS-2006-057.

Weerd, I. van de, Brinkkemper, S., & Versendaal, J. (2007). Concepts for incremental method evolution: empirical exploration and validation in requirements management. *Proceedings of the 19th International Conference on Advanced Information Systems Engineering, LNCS 4495*, 469–484.

Weerd, I. van de, Versendaal, J. & Brinkkemper, S. (2006). A product software knowledge infrastructure for situational capability maturation: Vision and case studies in product management. *Proceedings of the 12th Working Conference on Requirements Engineering: Foundation for Software Quality*, Luxembourg, 97-112.

Weerd, I. van de, Weerd, S. de, Brinkkemper, S. (2007). Developing a Reference Method for Game Production by Method Comparison. *Proceedings of the IFIP WG 8.1 Working Conference on Situational Method Engineering: Fundamentals and Experiences, IFIP Volume 244,* 313-327.

Wheelwright, S. C., & Clark, K. B. (1992). *Revolutionizing Product Development: Quantum Leaps in Speed, Efficiency, and Quality.* New York: The Free Press.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A.(1999). *Experimentation in Software Engineering: An Introduction*. Dordrecht, The Netherlands: Kluwer Academic Publishers.

Xu, L. & Brinkkemper, S. (2005). Concepts of Product Software: Paving the Road for Urgently Needed Research. In J. Castro & E. Teniente (Eds.), *The 1st International Workshop on Philosophical Foundations of Information Systems Engineering (PHISE'05)* (pp. 523-528). FEUP Press.

Xu, L., & Brinkkemper, S. (2007). Concepts of product software. *European Journal of Information Systems 16*(5), 531-541.

Yin, R. K. (2003). *Case Study Research: Design and Methods*. Thousand Oaks, CA: Sage Publications, Inc.

Zahran, S. (1997). Software *Process Improvement: Practical Guidelines for Business Success*, Reading MA: Addison-Wesley.

Publication list

In order of publication date

Weerd, I. van de, Souer, J., Versendaal, J., & Brinkkemper, S. (2005). Situational requirements engineering of web content management implementations. *Proceedings of the 1st International Workshop on Situational Requirements Engineering Processes (SREP'05)*, Paris, 13-30.

Nieuwenhuis, R., Weerd, I. van de, Bijlsma, L., Brinkkemper, S., & Versendaal J. (2006). The software product management workbench: An integrated environment for managing product releases in a distributed development context. *In Forum Proceedings of the 18th Conference on Advanced Information Systems Engineering, CEUR Workshop Proceedings 231*.

Weerd, I. van de (2006). A product software knowledge infrastructure for situational capability maturation. *Presented at the Doctoral Symposium of 4th International Requirements Engineering Conference*, Minneapolis/St. Paul, Minnesota, USA.

Weerd, I. van de, 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.

Weerd, I. van de, Brinkkemper, S., Nieuwenhuis, R., Versendaal, J., & Bijlsma, L. (2006). Towards a Reference Framework for Software Product Management. *Proceedings of the 14th International Requirements Engineering Conference*, Minneapolis/St. Paul, Minnesota, USA, 312-315.

Weerd, I. van de, Brinkkemper, S., Nieuwenhuis, R., Versendaal, J., & Bijlsma, L. (2006). On the creation of a reference framework for software product management: Validation and tool support. *Proceedings of the 1st International Workshop on Product Management*, Minneapolis/St. Paul, Minnesota, USA, 3-12.

Weerd, I. van de, Versendaal, J., & Brinkkemper, S. (2006). A product software knowledge infrastructure for situational capability maturation: Vision and

case studies in product management. *Proceedings of the 12th Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'06)*, Luxembourg, 97-112.

Souer, J., Weerd, I. van de, Versendaal, J., & Brinkkemper, S. (2007). Situational requirements engineering for the development of content management system-based web applications. *International Journal of Web Engineering and Technology 3*(4), 420-440.

Weerd, I. van de, Brinkkemper, S., & Versendaal J. (2007). Concepts for Incremental Method Evolution: Empirical Exploration and Validation in Requirements Management. *Proceedings of the 19th International Conference on Advanced Information Systems Engineering, LNCS 4495*, 469–484.

Weerd, I. van de, & Saeki, M. (2007). An Evaluation of Computerized Tools for Method Construction. *IEICE Technical Report 107*(176), SS2007-26, 59-63.

Weerd, I. van de, Weerd, S. de, & Brinkkemper, S. (2007). Developing a Reference Method for Game Production by Method Comparison. *Proceedings of the IFIP WG 8.1 Working Conference on Situational Method Engineering: Fundamentals and Experiences, IFIP Volume 244*, 313-327.

Amanatiadou, A., Weerd, I. van de, Spek, E. van der, & Brinkkemper, S. (2008). Developing a game categorization for situational game production. *Proceedings of IADIS International Conference Gaming 2008: Design for Engaging Experience and Social Interaction (Gaming 2008) Conference*, Amsterdam, The Netherlands, 95-98.

Bekkers, W., Weerd, I. van de, Brinkkemper, S., & Mahieu, A. (2008). The Influence of Situational Factors in Software Product Management: An Empirical Study. *Proceedings of the 21th International Workshop on Product Management*, Barcelona, Spain, 41-48.

Brinkkemper, S., Weerd, I. van de, Saeki, M., & Versendaal, J. (2008). Process improvement in requirements management: a method engineering approach. *Proceedings of the 13th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'08)*, LNCS 5025, 6-22.

Luinenburg, L., Jansen, S., Souer, J., Brinkkemper, S. & Weerd, I. van de, (2008). An Approach to Creating Product Software Design Methods: The

Case of Web Information Systems. *In: Advances in Conceptual Modeling – Challenges and Opportunities, LNCS 5232*, 426-436.

Luinenburg, L., Jansen, R.L., Souer, J., Weerd, I. van de, & Brinkkemper, S. (2008). Designing Web Content Management Systems Using the Method Association Approach. *Proceedings of the 4th International Workshop on Model- Driven Web Engineering (MDWE 2008)*, Toulouse, France, 106-120.

Weerd, I. van de, & Brinkkemper, S. (2008). Meta-modeling for situational analysis and design methods. In M.R. Syed and S.N. Syed (Eds.), *Handbook of Research on Modern Systems Analysis and Design Technologies and Applications* (pp. 38-58). Hershey: Idea Group Publishing.

Amanatiadou, A., & Weerd, I. van de (2009). Extending the Reference Method for Game Production: A Situational Approach. *Proceedings of the 1st International Conference on Games and Virtual Worlds for Serious Applications (VS-GAMES 2009)*, Coventry, United Kingdom, 20-27.

Weerd, I. van de, Brinkkemper, S., & Versendaal, J. (forthcoming). Incremental method evolution in global software product management: A retrospective case study. *Accepted for publication in the Journal of Information & Software Technology.*

Weerd, I. van de, Bekkers, W., & Brinkkemper, S. (submitted). Developing a maturity matrix for software product management.

Appendix A: SPM capabilities

Requirements Management

RM1. Requirements gathering

A. *Ad hoc requirements gathering.* Requirements are being gathered and registered.
B. *Organized requirements gathering.* All incoming requirements are stored in a central place (for example in a spreadsheet).
C. *Integrated requirements gathering.* Internal and external stakeholders use various channels (e.g. website, helpdesk) to submit requirements, which are automatically stored in a central database.
D. *Optimized requirements gathering.* Requirements are being gathered and stored automatically, for example by data mining in emails, setting out competitor spiders, etc.

RM2. Requirements identification

A. *Ad hoc requirements identification.* For incoming requirements, immediately a consideration is made whether they can be implemented in the product.
B. *Organized requirements identification.* Incoming requirements are being identified as market or product requirement. Product requirements are written in a pre-defined template.
C. *Optimized requirements identification.* Market and product requirements in the entire product portfolio are automatically linked, for example by using advanced techniques, such as linguistic engineering.

RM3. Requirements organizing

A. *Release-based requirements organization.* For each release, requirements are organized by, for example, theme, function or core asset.
B. *Roadmap-based requirements organization.* Organizing requirements is based on the product roadmap. Dependencies between different requirements are also identified.
C. *Portfolio-based requirements organization.* The organization of product requirements is automated. The various products, core assets, themes and the product roadmap are linked to each other.

Release Planning

RP1. Requirements prioritization

A. *Ad hoc requirements prioritization.* Requirements are prioritized.
B. *Release-based requirements prioritization.* Before defining the release content, requirements are prioritized by multiple internal stakeholders. A light-weight prioritization technique such as MOSCOW is used.
C. *Roadmap-based requirements prioritization.* During requirements prioritization, the roadmap is used as a guide and the dependencies between the different requirements are taken into account.
D. *Portfolio-based requirements prioritization.* During requirements prioritization, the products and core assets are taken into account, as well as information about the costs and revenues of each requirement. Sophisticated techniques such as integer linear programming are used.

RP2. Requirements selection

A. *Ad hoc requirements selection.* Requirements that have the highest priority are selected for the next release.
B. *Release-based requirements selection.* During requirements selection for the next release, constraints concerning engineering capacity are taken into account.
C. *Roadmap-based requirements selection.* Multiple releases are included in the requirements selection process and dependencies between requirements are handled.
D. *Portfolio-based requirements selection.* During the requirements selection process, multiple releases, products and core assets are taken into account.

RP3. Release definition

A. *Ad hoc release definition.* A release definition of approximately one page is being written, which contains an overview of the requirements that will be implemented, a time path and the needed capacity.
B. *Organized release definition.* A standard release definition template is used to write the release definition. The release definition is communicated to the internal stakeholders.
C. *Optimized release definition.* The release definition is generated from the list of selected requirements and the roadmap. It is communicated to internal and external stakeholders.

RP4. Release validation

A. *Ad hoc release validation.* De release definition is checked internal stakeholders, before the software is realized.
B. *Organized release validation.* The release definition must be formally approved, before the software is realized.
C. *Integrated release validation.* A business case (including the ROI) is being written before the software is realized.
D. *Optimized release validation.* Business intelligence is used to create a business case. Information concerning requirements' costs and revenues and experiences with earlier releases is used.

RP5. Launch preparation

A. *Ad hoc launch preparation.* Before the new release is launched, an informative email is being sent to internal stakeholders.
B. *Organized launch preparation.* Internal and external stakeholders are informed about the upcoming release and trainings are organized. A formal 'go' decision must be obtained from the board.
C. *Optimized launch preparation.* Internal and external stakeholders are notified automatically about the progress of the upcoming release, in order to prepare things like promotion material and trainings.

RP6. Scope change management

A. *Ad hoc scope change management.* In case of a scope change, project managers and/or developers are being informed.
B. *Organized scope change management.* A formal scope change management is in place, in which all involved stakeholders are informed.
C. *Optimized scope change management.* The scope change management process is automated, which makes is possible to: automatically notify the involved stakeholders; run an impact analysis and propose alternative plans.

Product Roadmapping

PR1. Theme identification

A. *Ad hoc theme identification.* A list of themes is created, which is used to decide on the contents of the release.
B. *Organized theme identification.* Release themes are identified and maintained. Themes are decided on together with the internal stakeholders.
C. *Optimized theme identification.* Identification themes results in a list of release themes that are stored centrally, so that requirements, core assets, market trends etc. can be linked to it.

PR2. Core asset coordination

A. *Organized core asset coordination.* All core assets are registered in standardized manner and are stored in a central location.
B. *Externally oriented core asset coordination.* External sources are investigated based on ROI in the search for core asset acquisition: partners, outsourcing or subcontracting of development.
C. *Optimized core asset coordination.* An automated procedure of acquiring core assets created by external parties, such as SOA, is in place to buy and sell core assets.

PR3. Roadmap construction

A. *Ad hoc roadmap construction.* A product roadmap exists in which the releases of the upcoming period are described.
B. *Organized roadmap construction.* Product roadmap(s) are created in consultation with internal stakeholders. The roadmap spans over multiple releases describing a period of at least one year and is actively maintained.y
C. *Optimized roadmap construction.* Roadmaps are created in consultation with both internal and external stakeholders. (Part of) the roadmap is communicated to the market.

Portfolio Management

PM1. Market trend identification

A. *Ad hoc market trend identification.* A document with market trends is being maintained.
B. *Organized market trend identification.* There is an active search for market trends. All search findings are recorded in standardized forms.
C. *Optimized market trend identification.* Large research projects are set up to investigate trends among every type of external party (customers, competitors, partners).

PM2. Partnering & contracting

A. *Organized partnering & contracting.* A process is in place to actively investigate make-or-buy decisions. Standard SLA's are used.
B. *Externally oriented partnering & contracting.* A partner network and/or partner portals are used to regulate partnering.
C. *Optimized partnering & contracting.* KPI's are set up to monitor the performance of partners on a regular basis.

PM3. Product lifecycle management

A. *Organized product lifecycle management.* All internal stakeholders are involved in product lifecycle management, and know in which phase of the lifecycle a product is.
B. *Externally oriented product lifecycle management.* External stakeholders are involved in the monitoring of deciding about product lifecycles.
C. *Optimized product lifecycle management.* Product lifecycles are tuned to each other across the entire portfolio, ensuring smooth transition between products, and maximizing the products lifecycle.

PM4. Product line identification

A. *Ad hoc product line identification.* Product lines are used, but not actively managed or monitored.
B. *Organized product line identification.* Product lines are actively managed and monitored, with several internal stakeholders are involved.
C. *Externally oriented product line identification.* Product lines are in line with those of external parties such as partners and suppliers.

Appendix B: Situational factors case companies

	SupplyComp	**CashComp**	**ProcessComp**
Business unit characteristics			
Development philosophy	Iterative	Waterfall	Iterative
Size of business unit team (fte)	13	50	15
Customer characteristics			
Customer loyalty	High	High	High
Customer satisfaction (1-10)	8	6	7
Customer variability	10%	10%	100%
Number of customers	59	<10	7
Type of customers	Large companies	SMEs & Large companies	SMEs & Large companies
Market characteristics			
Localization demand	3	5	4
Market growth	Growing	Growing	Growing
Market size (customers)	1500–3500	3500+	500-1500
Release frequency (days)	90	365	120
Sector	Logistics	Petrol, retail, transport	Utilities
Standard dominance	Medium	High	Low
Variability of feature requests	High	Medium	High
Product characteristics			

Defects per year	40	35	50
Development platform maturity	Ever changing	Ever changing	Fully developed
New requirements rate (per year)	250	150	100
Number of products	2	7	2
Product age (years)	4	4,5	4
Product lifetime (years)	>15	10	>15
Product size (KLOC)	320.000	500.000	-
Product tolerance	Medium	Low	Medium
Stakeholder involvement			
Company policy	High	Low	High
Customer involvement	High	Medium	High
Legislation	Loose	Strict	Non-existing
Partner involvement	Low	Low	Low

Summary

Hardly any figures exist of the success of product software companies. What we do know is that a good Software Product Management (SPM) practice pays off. However, not many IT-professionals know how to implement SPM practices in their organization, which causes many companies to not have the proper processes in place. One of the reasons for this lack of knowledge is that hardly any education exists in the SPM domain. As a consequence, software product managers have to learn the practice on the job. Without a solid body of SPM knowledge, this can be a difficult task.

We can address the lack of knowledge among product managers by giving them access to SPM methods and guide them in implementing them in their company. Immediately some other problems come to mind. For example, product software companies can be characterized by various situational factors; they operate in diverse sectors, have varying sizes and use a range of development methods. Subsequently, companies need different methods. For example, a company with 5 employees does not need an elaborate release planning method, whereas a large company, such as Microsoft, needs to have a very elaborate workflow process in place. These situational factors influence the decision whether to implement simple or elaborate SPM processes profoundly.

In this research, a knowledge infrastructure is proposed that provides methodical support to product software companies. The aim of this knowledge infrastructure is to assess and analyze a company's current situation and maturity level. Then, by using incremental method engineering and meta-modeling principles, previously stored method fragments can be selected and assembled into a process advice. By implementing this process advice, the overall maturity of the SPM practice increases.

This dissertation consists of three parts. First, the main processes in SPM (requirements management, release planning, product roadmapping, and portfolio management) and the internal and external stakeholders are described. In the second part, a modeling technique for storing method fragments is proposed. Furthermore, the principles for incremental method engineering are identified, formalized and validated in a retrospective case study. In the third part, an approach for incremental method evolution is described. In this

approach, the aforementioned concepts are combined with a maturity matrix for SPM, and integrated in one knowledge infrastructure. A comparative case study in three product software companies is carried out to test the approach. The results indicate that the knowledge infrastructure is able to create a useful process advice for improving a company's SPM practice.

Nederlandse samenvatting

Er zijn weinig cijfers bekend over het succes van productsoftwarebedrijven. Wat we wel weten is dat goed geïmplementeerde Software Product Management (SPM) processen hier een positieve invloed op hebben. Toch weten weinig IT professionals hoe ze SPM processen in hun organisatie kunnen implementeren. Een reden voor dit gebrek aan kennis is dat er nauwelijks onderwijs wordt gegeven in dit domein. Als gevolg hiervan leren productmanagers pas tijdens hun baan wat SPM inhoud en hoe de processen moeten worden ingericht in een organisatie. Omdat de kennis over SPM gefragmenteerd is, kan dit een moeilijke taak zijn.

Het probleem dat hierboven geschetst is, kan opgelost worden door productmanagers toegang te geven tot SPM methoden en ze te helpen met de implementatie hiervan in hun organisatie. Een moeilijkheid hierbij is dat elk productsoftwarebedrijf een eigen context heeft die gekarakteriseerd kan worden door verschillende situationele factoren. Zo opereren productsoftwarebedrijven in verschillende sectoren, zijn ze van verschillende groottes en gebruiken ze verschillende ontwikkelmethodes voor hun software. Deze situationele factoren hebben invloed op de methodes die een bedrijf gebruikt. Een bedrijf met vijf werknemers zal geen uitgebreide release planning methode nodig hebben, terwijl grote bedrijven, zoal Microsoft of SAP, een uitgebreide workflow in dit domein moeten hebben geïmplementeerd.

In dit onderzoek wordt een kennisinfrastructuur ontwikkeld die methodische ondersteuning kan geven aan productsoftwarebedrijven. Het doel van deze kennisinfrastructuur is het analyseren van de volwassenheid en situationele factoren van een bedrijf. Door middel van incrementele method engineering principes kan een proces advies worden geassembleerd dat is samengesteld uit bestaande, eerder opgeslagen, methodefragmenten. Door dit advies bij het bedrijf in het bestaande proces te implementeren wordt de volwassenheid van de SPM activiteiten verhoogd.

Dit proefschrift bestaat uit drie delen. In het eerste deel wordt het referentieraamwerk voor SPM geïntroduceerd, waarin de belangrijkste processen (requirements management, release planning, product roadmapping en portfolio management) en interne en externe stakeholders worden beschreven. In het tweede deel wordt een modelleringstechniek voor het

analyseren en opslaan van methodefragmenten beschreven. Ook worden de principes van incrementele method engineering geïdentificeerd, geformaliseerd en gevalideerd in een retrospectieve case study. In het derde deel wordt het proces van incrementele method evolutie beschreven. In dit proces worden de eerder genoemde concepten gecombineerd in een volwassenheidsmatrix voor SPM welke vervolgens geïntegreerd wordt in de kennisinfrastructuur. Dit proces wordt getest in een vergelijkende case study waarin drie productsoftwarebedrijven geanalyseerd worden. De resultaten geven aan dat met de kennisinfrastructuur een bruikbaar procesadvies kan worden gecreëerd waarmee bedrijven hun SPM processen kunnen verbeteren.

Curriculum Vitae

Inge van de Weerd was born on January 3, 1981 in The Netherlands. In 1999 she started with the study General Arts. After receiving her propedeutics, she made a switch to Information Science at Utrecht University. During this time she worked as a teaching assistant and ran her own web design company. She obtained her Masters degree in Business Informatics in 2005 and immediately started with her PhD research. After she was awarded the JSPS (Japan Society for the Promotion of Science) Postdoctoral Fellowship for Foreign Researchers in 2007, she moved to Japan for four months to conduct research at the Tokyo Institute of Technology. In 2009, within the appointed four years, she successfully finished her PhD research.

Since September 2009, Inge van de Weerd is employed as an assistant professor at Utrecht University at the Department of Information and Computing Sciences. She publishes on an international level and is active as a committee member for several scientific conferences. Her research interests focus on three main areas: software product management, game production and method engineering. In addition she teaches several Master courses and supervises students in their thesis projects.

Inge van de Weerd is very active in the software product management domain. She is a member of the International Software Product Management Board that aims for standardization of education and certification in software product management. In addition, she is the organizer and lecturer of a sofware product management course, aimed at product management professionals.

SIKS PhD theses

1998-1 Johan van den Akker (CWI)
DEGAS - An Active, Temporal Database of Autonomous Objects

1998-2 Floris Wiesman (UM)
Information Retrieval by Graphically Browsing Meta-Information

1998-3 Ans Steuten (TUD)
A Contribution to the Linguistic Analysis of Business Conversations within the Language/Action Perspective

1998-4 Dennis Breuker (UM)
Memory versus Search in Games

1998-5 E.W.Oskamp (RUL)
Computerondersteuning bij Straftoemeting

1999-1 Mark Sloof (VU)
Physiology of Quality Change Modelling; Automated modelling of Quality Change of Agricultural Products

1999-2 Rob Potharst (EUR)
Classification using decision trees and neural nets

1999-3 Don Beal (UM)
The Nature of Minimax Search

1999-4 Jacques Penders (UM)
The practical Art of Moving Physical Objects

1999-5 Aldo de Moor (KUB)
Empowering Communities: A Method for the Legitimate User-Driven Specification of Network Information Systems

1999-6 Niek J.E. Wijngaards (VU)
Re-design of compositional systems

1999-7 David Spelt (UT)
Verification support for object database design

1999-8 Jacques H.J. Lenting (UM)
Informed Gambling: Conception and Analysis of a Multi-Agent Mechanism for Discrete Reallocation.

2000-1 Frank Niessink (VU)
Perspectives on Improving Software Maintenance

2000-2 Koen Holtman (TUE)
Prototyping of CMS Storage Management

2000-3 Carolien M.T. Metselaar (UVA)
Sociaal-organisatorische gevolgen van kennistechnologie; een procesbenadering en actorperspectief.

2000-4 Geert de Haan (VU)
ETAG, A Formal Model of Competence Knowledge for User Interface Design
2000-5 Ruud van der Pol (UM)
Knowledge-based Query Formulation in Information Retrieval.
2000-6 Rogier van Eijk (UU)
Programming Languages for Agent Communication
2000-7 Niels Peek (UU)
Decision-theoretic Planning of Clinical Patient Management
2000-8 Veerle Coup, (EUR)
Sensitivity Analyis of Decision-Theoretic Networks
2000-9 Florian Waas (CWI)
Principles of Probabilistic Query Optimization
2000-10 Niels Nes (CWI)
Image Database Management System Design Considerations, Algorithms and Architecture
2000-11 Jonas Karlsson (CWI)
Scalable Distributed Data Structures for Database Management
2001-1 Silja Renooij (UU)
Qualitative Approaches to Quantifying Probabilistic Networks
2001-2 Koen Hindriks (UU)
Agent Programming Languages: Programming with Mental Models
2001-3 Maarten van Someren (UvA)
Learning as problem solving
2001-4 Evgueni Smirnov (UM)
Conjunctive and Disjunctive Version Spaces with Instance-Based Boundary Sets
2001-5 Jacco van Ossenbruggen (VU)
Processing Structured Hypermedia: A Matter of Style
2001-6 Martijn van Welie (VU)
Task-based User Interface Design
2001-7 Bastiaan Schonhage (VU)
Diva: Architectural Perspectives on Information Visualization
2001-8 Pascal van Eck (VU)
A Compositional Semantic Structure for Multi-Agent Systems Dynamics.
2001-9 Pieter Jan 't Hoen (RUL)
Towards Distributed Development of Large Object-Oriented Models, Views of Packages as Classes
2001-10 Maarten Sierhuis (UvA)
Modeling and Simulating Work Practice
BRAHMS: a multiagent modeling and simulation language for work practice analysis and design
2001-11 Tom M. van Engers (VUA)
Knowledge Management: The Role of Mental Models in Business Systems Design

2002-01 Nico Lassing (VU)
Architecture-Level Modifiability Analysis
2002-02 Roelof van Zwol (UT)
Modelling and searching web-based document collections
2002-03 Henk Ernst Blok (UT)
Database Optimization Aspects for Information Retrieval
2002-04 Juan Roberto Castelo Valdueza (UU)
The Discrete Acyclic Digraph Markov Model in Data Mining

2002-05 Radu Serban (VU)
The Private Cyberspace Modeling Electronic Environments inhabited by Privacy-concerned Agents
2002-06 Laurens Mommers (UL)
Applied legal epistemology; Building a knowledge-based ontology of the legal domain
2002-07 Peter Boncz (CWI)
Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications
2002-08 Jaap Gordijn (VU)
Value Based Requirements Engineering: Exploring Innovative E-Commerce Ideas
2002-09 Willem-Jan van den Heuvel(KUB)
Integrating Modern Business Applications with Objectified Legacy Systems
2002-10 Brian Sheppard (UM)
Towards Perfect Play of Scrabble
2002-11 Wouter C.A. Wijngaards (VU)
Agent Based Modelling of Dynamics: Biological and Organisational Applications
2002-12 Albrecht Schmidt (Uva)
Processing XML in Database Systems
2002-13 Hongjing Wu (TUE)
A Reference Architecture for Adaptive Hypermedia Applications
2002-14 Wieke de Vries (UU)
Agent Interaction: Abstract Approaches to Modelling, Programming and Verifying Multi-Agent Systems
2002-15 Rik Eshuis (UT)
Semantics and Verification of UML Activity Diagrams for Workflow Modelling
2002-16 Pieter van Langen (VU)
The Anatomy of Design: Foundations, Models and Applications
2002-17 Stefan Manegold (UVA)
Understanding, Modeling, and Improving Main-Memory Database Performance
2003-01 Heiner Stuckenschmidt (VU)
Ontology-Based Information Sharing in Weakly Structured Environments
2003-02 Jan Broersen (VU)

Modal Action Logics for Reasoning About Reactive Systems

2003-03 Martijn Schuemie (TUD)
Human-Computer Interaction and Presence in Virtual Reality Exposure Therapy

2003-04 Milan Petkovic (UT)
Content-Based Video Retrieval Supported by Database Technology

2003-05 Jos Lehmann (UVA)
Causation in Artificial Intelligence and Law - A modelling approach

2003-06 Boris van Schooten (UT)
Development and specification of virtual environments

2003-07 Machiel Jansen (UvA)
Formal Explorations of Knowledge Intensive Tasks

2003-08 Yongping Ran (UM)
Repair Based Scheduling

2003-09 Rens Kortmann (UM)
The resolution of visually guided behaviour

2003-10 Andreas Lincke (UvT)
Electronic Business Negotiation: Some experimental studies on the interaction between medium, innovation context and culture

2003-11 Simon Keizer (UT)
Reasoning under Uncertainty in Natural Language Dialogue using Bayesian Networks

2003-12 Roeland Ordelman (UT)
Dutch speech recognition in multimedia information retrieval

2003-13 Jeroen Donkers (UM)
Nosce Hostem - Searching with Opponent Models

2003-14 Stijn Hoppenbrouwers (KUN)
Freezing Language: Conceptualisation Processes across ICT-Supported Organisations

2003-15 Mathijs de Weerdt (TUD)
Plan Merging in Multi-Agent Systems

2003-16 Menzo Windhouwer (CWI)
Feature Grammar Systems - Incremental Maintenance of Indexes to Digital Media Warehouses

2003-17 David Jansen (UT)
Extensions of Statecharts with Probability, Time, and Stochastic Timing

2003-18 Levente Kocsis (UM)
Learning Search Decisions

2004-01 Virginia Dignum (UU)
A Model for Organizational Interaction: Based on Agents, Founded in Logic

2004-02 Lai Xu (UvT)
Monitoring Multi-party Contracts for E-business

2004-03 Perry Groot (VU)
A Theoretical and Empirical Analysis of Approximation in Symbolic Problem Solving

2004-04 Chris van Aart (UVA)
Organizational Principles for Multi-Agent Architectures
2004-05 Viara Popova (EUR)
Knowledge discovery and monotonicity
2004-06 Bart-Jan Hommes (TUD)
The Evaluation of Business Process Modeling Techniques
2004-07 Elise Boltjes (UM)
Voorbeeldig onderwijs; voorbeeldgestuurd onderwijs, een opstap naar abstract denken, vooral voor meisjes
2004-08 Joop Verbeek(UM)
Politie en de Nieuwe Internationale Informatiemarkt, Grensregionale politiële gegevensuitwisseling en digitale expertise
2004-09 Martin Caminada (VU)
For the Sake of the Argument; explorations into argument-based reasoning
2004-10 Suzanne Kabel (UVA)
Knowledge-rich indexing of learning-objects
2004-11 Michel Klein (VU)
Change Management for Distributed Ontologies
2004-12 The Duy Bui (UT)
Creating emotions and facial expressions for embodied agents
2004-13 Wojciech Jamroga (UT)
Using Multiple Models of Reality: On Agents who Know how to Play
2004-14 Paul Harrenstein (UU)
Logic in Conflict. Logical Explorations in Strategic Equilibrium
2004-15 Arno Knobbe (UU)
Multi-Relational Data Mining
2004-16 Federico Divina (VU)
Hybrid Genetic Relational Search for Inductive Learning
2004-17 Mark Winands (UM)
Informed Search in Complex Games
2004-18 Vania Bessa Machado (UvA)
Supporting the Construction of Qualitative Knowledge Models
2004-19 Thijs Westerveld (UT)
Using generative probabilistic models for multimedia retrieval
2004-20 Madelon Evers (Nyenrode)
Learning from Design: facilitating multidisciplinary design teams
2005-01 Floor Verdenius (UVA)
Methodological Aspects of Designing Induction-Based Applications
2005-02 Erik van der Werf (UM))
AI techniques for the game of Go
2005-03 Franc Grootjen (RUN)
A Pragmatic Approach to the Conceptualisation of Language
2005-04 Nirvana Meratnia (UT)
Towards Database Support for Moving Object data
2005-05 Gabriel Infante-Lopez (UVA)

Two-Level Probabilistic Grammars for Natural Language Parsing
2005-06 Pieter Spronck (UM)
Adaptive Game AI
2005-07 Flavius Frasincar (TUE)
Hypermedia Presentation Generation for Semantic Web Information Systems
2005-08 Richard Vdovjak (TUE)
A Model-driven Approach for Building Distributed Ontology-based Web Applications
2005-09 Jeen Broekstra (VU)
Storage, Querying and Inferencing for Semantic Web Languages
2005-10 Anders Bouwer (UVA)
Explaining Behaviour: Using Qualitative Simulation in Interactive Learning Environments
2005-11 Elth Ogston (VU)
Agent Based Matchmaking and Clustering - A Decentralized Approach to Search
2005-12 Csaba Boer (EUR)
Distributed Simulation in Industry
2005-13 Fred Hamburg (UL)
Een Computermodel voor het Ondersteunen van Euthanasiebeslissingen
2005-14 Borys Omelayenko (VU)
Web-Service configuration on the Semantic Web; Exploring how semantics meets pragmatics
2005-15 Tibor Bosse (VU)
Analysis of the Dynamics of Cognitive Processes
2005-16 Joris Graaumans (UU)
Usability of XML Query Languages
2005-17 Boris Shishkov (TUD)
Software Specification Based on Re-usable Business Components
2005-18 Danielle Sent (UU)
Test-selection strategies for probabilistic networks
2005-19 Michel van Dartel (UM)
Situated Representation
2005-20 Cristina Coteanu (UL)
Cyber Consumer Law, State of the Art and Perspectives
2005-21 Wijnand Derks (UT)
Improving Concurrency and Recovery in Database Systems by Exploiting Application Semantics
2006-01 Samuil Angelov (TUE)
Foundations of B2B Electronic Contracting
2006-02 Cristina Chisalita (VU)
Contextual issues in the design and use of information technology in organizations
2006-03 Noor Christoph (UVA)
The role of metacognitive skills in learning to solve problems

2006-04 Marta Sabou (VU)
Building Web Service Ontologies
2006-05 Cees Pierik (UU)
Validation Techniques for Object-Oriented Proof Outlines
2006-06 Ziv Baida (VU)
Software-aided Service Bundling - Intelligent Methods & Tools for Graphical Service Modeling
2006-07 Marko Smiljanic (UT)
XML schema matching -- balancing efficiency and effectiveness by means of clustering
2006-08 Eelco Herder (UT)
Forward, Back and Home Again - Analyzing User Behavior on the Web
2006-09 Mohamed Wahdan (UM)
Automatic Formulation of the Auditor's Opinion
2006-10 Ronny Siebes (VU)
Semantic Routing in Peer-to-Peer Systems
2006-11 Joeri van Ruth (UT)
Flattening Queries over Nested Data Types
2006-12 Bert Bongers (VU)
Interactivation - Towards an e-cology of people, our technological environment, and the arts
2006-13 Henk-Jan Lebbink (UU)
Dialogue and Decision Games for Information Exchanging Agents
2006-14 Johan Hoorn (VU)
Software Requirements: Update, Upgrade, Redesign - towards a Theory of Requirements Change
2006-15 Rainer Malik (UU)
CONAN: Text Mining in the Biomedical Domain
2006-16 Carsten Riggelsen (UU)
Approximation Methods for Efficient Learning of Bayesian Networks
2006-17 Stacey Nagata (UU)
User Assistance for Multitasking with Interruptions on a Mobile Device
2006-18 Valentin Zhizhkun (UVA)
Graph transformation for Natural Language Processing
2006-19 Birna van Riemsdijk (UU)
Cognitive Agent Programming: A Semantic Approach
2006-20 Marina Velikova (UvT)
Monotone models for prediction in data mining
2006-21 Bas van Gils (RUN)
Aptness on the Web
2006-22 Paul de Vrieze (RUN)
Fundaments of Adaptive Personalisation
2006-23 Ion Juvina (UU)
Development of Cognitive Model for Navigating on the Web
2006-24 Laura Hollink (VU)

Semantic Annotation for Retrieval of Visual Resources
2006-25 Madalina Drugan (UU)
Conditional log-likelihood MDL and Evolutionary MCMC
2006-26 Vojkan Mihajlovic (UT)
Score Region Algebra: A Flexible Framework for Structured Information Retrieval
2006-27 Stefano Bocconi (CWI)
Vox Populi: generating video documentaries from semantically annotated media repositories
2006-28 Borkur Sigurbjornsson (UVA)
Focused Information Access using XML Element Retrieval
2007-01 Kees Leune (UvT)
Access Control and Service-Oriented Architectures
2007-02 Wouter Teepe (RUG)
Reconciling Information Exchange and Confidentiality: A Formal Approach
2007-03 Peter Mika (VU)
Social Networks and the Semantic Web
2007-04 Jurriaan van Diggelen (UU)
Achieving Semantic Interoperability in Multi-agent Systems: a dialogue-based approach
2007-05 Bart Schermer (UL)
Software Agents, Surveillance, and the Right to Privacy: a Legislative Framework for Agent-enabled Surveillance
2007-06 Gilad Mishne (UVA)
Applied Text Analytics for Blogs
2007-07 Natasa Jovanovic' (UT)
To Whom It May Concern - Addressee Identification in Face-to-Face Meetings
2007-08 Mark Hoogendoorn (VU)
Modeling of Change in Multi-Agent Organizations
2007-09 David Mobach (VU)
Agent-Based Mediated Service Negotiation
2007-10 Huib Aldewereld (UU)
Autonomy vs. Conformity: an Institutional Perspective on Norms and Protocols
2007-11 Natalia Stash (TUE)
Incorporating Cognitive/Learning Styles in a General-Purpose Adaptive Hypermedia System
2007-12 Marcel van Gerven (RUN)
Bayesian Networks for Clinical Decision Support: A Rational Approach to Dynamic Decision-Making under Uncertainty
2007-13 Rutger Rienks (UT)
Meetings in Smart Environments; Implications of Progressing Technology
2007-14 Niek Bergboer (UM)
Context-Based Image Analysis
2007-15 Joyca Lacroix (UM)

NIM: a Situated Computational Memory Model

2007-16 Davide Grossi (UU)
Designing Invisible Handcuffs. Formal investigations in Institutions and Organizations for Multi-agent Systems

2007-17 Theodore Charitos (UU)
Reasoning with Dynamic Networks in Practice

2007-18 Bart Orriens (UvT)
On the development an management of adaptive business collaborations

2007-19 David Levy (UM)
Intimate relationships with artificial partners

2007-20 Slinger Jansen (UU)
Customer Configuration Updating in a Software Supply Network

2007-21 Karianne Vermaas (UU)
Fast diffusion and broadening use: A research on residential adoption and usage of broadband internet in the Netherlands between 2001 and 2005

2007-22 Zlatko Zlatev (UT)
Goal-oriented design of value and process models from patterns

2007-23 Peter Barna (TUE)
Specification of Application Logic in Web Information Systems

2007-24 Georgina Ramírez Camps (CWI)
Structural Features in XML Retrieval

2007-25 Joost Schalken (VU)
Empirical Investigations in Software Process Improvement

2008-01 Katalin Boer-Sorbán (EUR)
Agent-Based Simulation of Financial Markets: A modular, continuous-time approach

2008-02 Alexei Sharpanskykh (VU)
On Computer-Aided Methods for Modeling and Analysis of Organizations

2008-03 Vera Hollink (UVA)
Optimizing hierarchical menus: a usage-based approach

2008-04 Ander de Keijzer (UT)
Management of Uncertain Data - towards unattended integration

2008-05 Bela Mutschler (UT)
Modeling and simulating causal dependencies on process-aware information systems from a cost perspective

2008-06 Arjen Hommersom (RUN)
On the Application of Formal Methods to Clinical Guidelines, an Artificial Intelligence Perspective

2008-07 Peter van Rosmalen (OU)
Supporting the tutor in the design and support of adaptive e-learning

2008-08 Janneke Bolt (UU)
Bayesian Networks: Aspects of Approximate Inference

2008-09 Christof van Nimwegen (UU)
The paradox of the guided user: assistance can be counter-effective

2008-10 Wauter Bosma (UT)

Discourse oriented summarization
2008-11 Vera Kartseva (VU)
Designing Controls for Network Organizations: A Value-Based Approach
2008-12 Jozsef Farkas (RUN)
A Semiotically Oriented Cognitive Model of Knowledge Representation
2008-13 Caterina Carraciolo (UVA)
Topic Driven Access to Scientific Handbooks
2008-14 Arthur van Bunningen (UT)
Context-Aware Querying; Better Answers with Less Effort
2008-15 Martijn van Otterlo (UT)
The Logic of Adaptive Behavior: Knowledge Representation and Algorithms for the Markov Decision Process Framework in First-Order Domains.
2008-16 Henriette van Vugt (VU)
Embodied agents from a user's perspective
2008-17 Martin Op 't Land (TUD)
Applying Architecture and Ontology to the Splitting and Allying of Enterprises
2008-18 Guido de Croon (UM)
Adaptive Active Vision
2008-19 Henning Rode (UT)
From Document to Entity Retrieval: Improving Precision and Performance of Focused Text Search

2008-20 Rex Arendsen (UVA)
Geen bericht, goed bericht. Een onderzoek naar de effecten van de introductie van elektronisch berichtenverkeer met de overheid op de administratieve lasten van bedrijven
2008-21 Krisztian Balog (UVA)
People Search in the Enterprise
2008-22 Henk Koning (UU)
Communication of IT-Architecture
2008-23 Stefan Visscher (UU)
Bayesian network models for the management of ventilator-associated pneumonia
2008-24 Zharko Aleksovski (VU)
Using background knowledge in ontology matching
2008-25 Geert Jonker (UU)
Efficient and Equitable Exchange in Air Traffic Management Plan Repair using Spender-signed Currency
2008-26 Marijn Huijbregts (UT)
Segmentation, Diarization and Speech Transcription: Surprise Data Unraveled
2008-27 Hubert Vogten (OU)
Design and Implementation Strategies for IMS Learning Design
2008-28 Ildiko Flesch (RUN)
On the Use of Independence Relations in Bayesian Networks
2008-29 Dennis Reidsma (UT)

Annotations and Subjective Machines - Of Annotators, Embodied Agents, Users, and Other Humans

2008-30 Wouter van Atteveldt (VU)
Semantic Network Analysis: Techniques for Extracting, Representing and Querying Media Content

2008-31 Loes Braun (UM)
Pro-Active Medical Information Retrieval

2008-32 Trung H. Bui (UT)
Toward Affective Dialogue Management using Partially Observable Markov Decision Processes

2008-33 Frank Terpstra (UVA)
Scientific Workflow Design; theoretical and practical issues

2008-34 Jeroen de Knijf (UU)
Studies in Frequent Tree Mining

2008-35 Ben Torben Nielsen (UvT)
Dendritic morphologies: function shapes structure

2009-01 Rasa Jurgelenaite (RUN)
Symmetric Causal Independence Models

2009-02 Willem Robert van Hage (VU)
Evaluating Ontology-Alignment Techniques

2009-03 Hans Stol (UvT)
A Framework for Evidence-based Policy Making Using IT

2009-04 Josephine Nabukenya (RUN)
Improving the Quality of Organisational Policy Making using Collaboration Engineering

2009-05 Sietse Overbeek (RUN)
Bridging Supply and Demand for Knowledge Intensive Tasks - Based on Knowledge, Cognition, and Quality

2009-06 Muhammad Subianto (UU)
Understanding Classification

2009-07 Ronald Poppe (UT)
Discriminative Vision-Based Recovery and Recognition of Human Motion

2009-08 Volker Nannen (VU)
Evolutionary Agent-Based Policy Analysis in Dynamic Environments

2009-09 Benjamin Kanagwa (RUN)
Design, Discovery and Construction of Service-oriented Systems

2009-10 Jan Wielemaker (UVA)
Logic programming for knowledge-intensive interactive applications

2009-11 Alexander Boer (UVA)
Legal Theory, Sources of Law & the Semantic Web

2009-12 Peter Massuthe (TUE, Humboldt-Universitaet zu Berlin)
Operating Guidelines for Services

2009-13 Steven de Jong (UM)
Fairness in Multi-Agent Systems

2009-14 Maksym Korotkiy (VU)

From ontology-enabled services to service-enabled ontologies (making ontologies work in e-science with ONTO-SOA)

2009-15 Rinke Hoekstra (UVA)
Ontology Representation - Design Patterns and Ontologies that Make Sense

2009-16 Fritz Reul (UvT)
New Architectures in Computer Chess

2009-17 Laurens van der Maaten (UvT)
Feature Extraction from Visual Data

2009-18 Fabian Groffen (CWI)
Armada, An Evolving Database System

2009-19 Valentin Robu (CWI)
Modeling Preferences, Strategic Reasoning and Collaboration in Agent-Mediated Electronic Markets

2009-20 Bob van der Vecht (UU)
Adjustable Autonomy: Controling Influences on Decision Making

2009-21 Stijn Vanderlooy (UM)
Ranking and Reliable Classification

2009-22 Pavel Serdyukov (UT)
Search For Expertise: Going beyond direct evidence

2009-23 Peter Hofgesang (VU)
Modelling Web Usage in a Changing Environment

2009-24 Annerieke Heuvelink (VUA)
Cognitive Models for Training Simulations

2009-25 Alex van Ballegooij (CWI)
"RAM: Array Database Management through Relational Mapping"

2009-26 Fernando Koch (UU)
An Agent-Based Model for the Development of Intelligent Mobile Services

2009-27 Christian Glahn (OU)
Contextual Support of social Engagement and Reflection on the Web

2009-28 Sander Evers (UT)
Sensor Data Management with Probabilistic Models

2009-29 Stanislav Pokraev (UT)
Model-Driven Semantic Integration of Service-Oriented Applications

2009-30 Marcin Zukowski (CWI)
Balancing vectorized query execution with bandwidth-optimized storage

2009-31 Sofiya Katrenko (UVA)
A Closer Look at Learning Relations from Text

2009-32 Rik Farenhorst (VU) and Remco de Boer (VU)
Architectural Knowledge Management: Supporting Architects and Auditors

2009-33 Khiet Truong (UT)
How Does Real Affect Affect Affect Recognition In Speech?

2009-34 Inge van de Weerd (UU)
Advancing in Software Product Management: An Incremental Method Engineering Approach