Published in proceedings of REFSQ'08, LNCS 5025, 6-22

Process Improvement in Requirements Management: a Method Engineering Approach

Sjaak Brinkkemper[1], Inge van de Weerd[1], Motoshi Saeki[2], Johan Versendaal[1]

[1] Department of Information and Computing Sciences
Utrecht University, Utrecht, The Netherlands
{s.brinkkemper, i.vandeweerd, j.versendaal}@cs.uu.nl

[2] Department of Computer Science
Tokyo Institute of Technology, Tokyo, Japan
saeki@cs.titech.ac.jp

Abstract. 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.

Keywords: method engineering, requirements engineering, software process improvement, incremental method evolution, root cause analysis, computer aided method engineering, CAME

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 formulate more formally, the 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 [1]. 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 [2], into the requirements management methods of a product software company [3].

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 [4], product development [5] [6], management of software products [3] [7], requirements management [8], release planning [9] [10], product line engineering [11] [12], product delivery [13], and so on.

Xu and Brinkkemper [4] 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 [4]. 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 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.

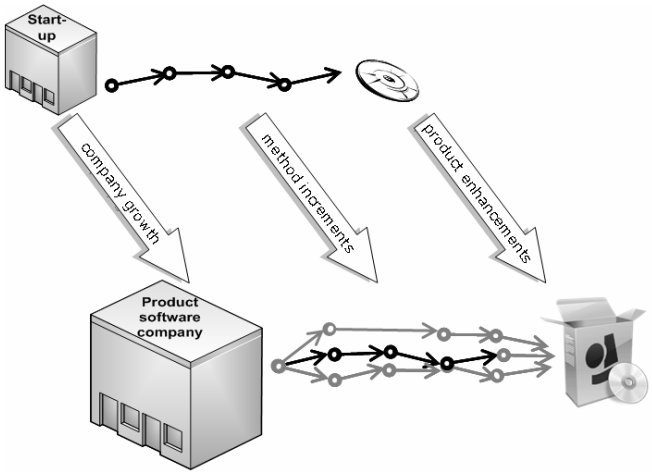

Fig. 1. Incremental method evolution in a product software company

Several software process improvement approaches have been proposed to improve software development processes [14] [15]. 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 [16].

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) [17] [18], 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.

1.2 Research Approach

This research project is carried out following the design research methodology for performing research in information systems as described by [19] and [20]. Research in design science is done through the processes of *building* and *evaluating* artifacts [19] [20]. According to Hevner et al. [19], 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 [17], we described our vision on this issue and introduced the PSKI, our main new artifact. Subsequently, in [18], 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 [3].

[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.

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.

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 [14] and SPICE [15]. Secondly, the problem-based approach uses the mechanism of solving the underlying problems, or root causes, that cause a certain process to under perform. 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 [21].

In literature, some critique exists on capability-based approaches. For example, a capability-based approach is can be encountered as too superficial for small companies [22]. In addition, these kinds of process improvement approaches are often difficult to implement. In [23], it was 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. More over, 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 2, we illustrate this approach.

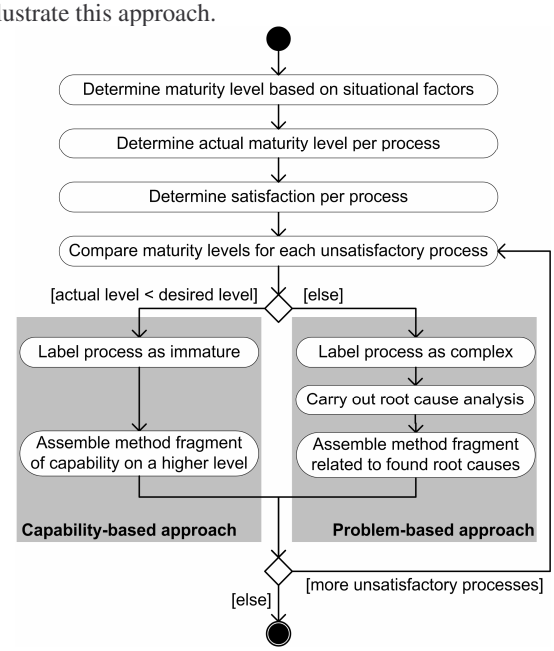

Fig. 2. Process improvement approach

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.

2.2 Functional Architecture

Starting from [17], the following components in the PSKI (see Figure 3) 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.

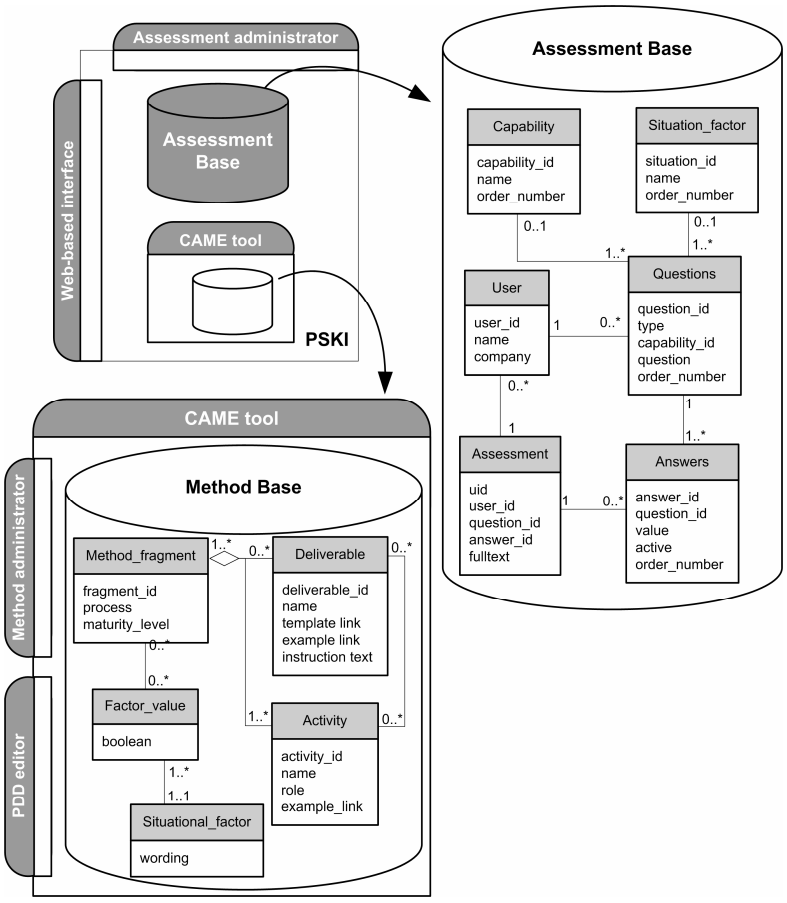

Fig. 3. Functional architecture of the PSKI

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.

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 4 illustrates the current requirements management process of ERPComp in PDD notation [24]. 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.

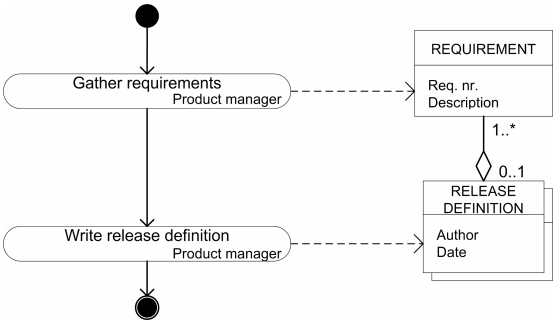

Fig. 4. Snapshot of increment #0

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 5. By following the solid arrows, the activities concerning the problem-based approach (*Present root cause map* and *Store root causes*) are skipped.

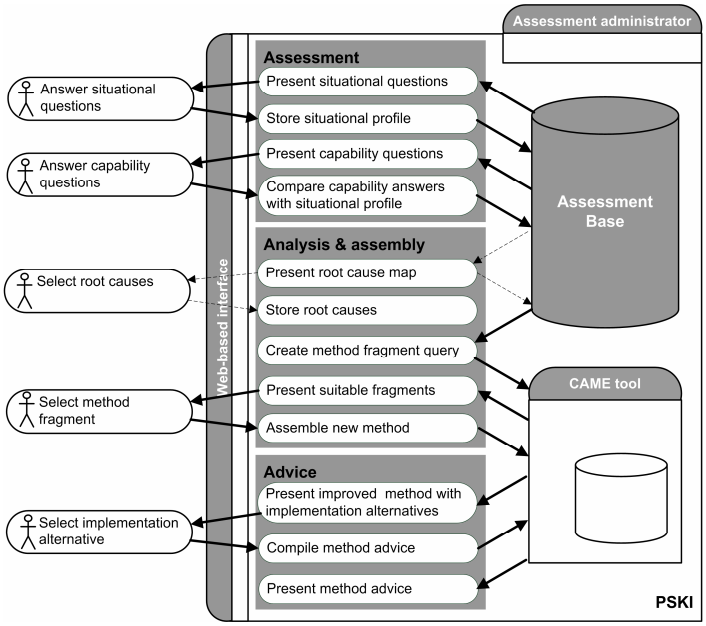

Fig. 5. Capability-based PSKI scenario

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.

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 [25] [26]. 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 10.

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 8 and 9 and described in [24]. The structure of a generic method fragment is depicted in Figure 6.

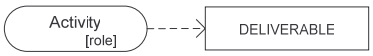

Fig. 6. Generic method fragment structure

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 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.

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 [27] and the Test Process Improvement model [28]. We distinguish 16 SPM processes in the maturity matrix that originate from the reference framework for SPM [3] and 11 maturity levels. The number of maturity levels is determined by the implementation dependencies of the capabilities. In Table 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 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:

A. Requirements prioritization per release
B. Requirements prioritization as an ongoing process
C. Requirements prioritization as an ongoing process, over multiple product lines
D. 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:

A. Requirements validation per release
B. Requirements validation as an ongoing, automated process,
C. Requirements validation as an ongoing process, over multiple products
D. 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.

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 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 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. The cost-value approach is proposed in [2] an evaluated in [29] as a method for requirements prioritization in market-driven software product development.

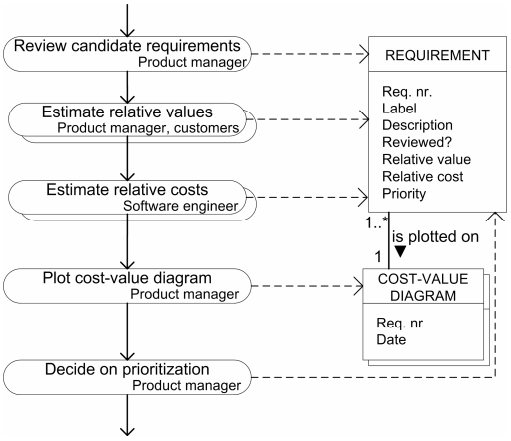

Fig. 7. Method fragment linked to 'Requirements prioritization per release'

In Figure 8, we illustrate the method fragment related to the capability 'Requirements validation per release'.

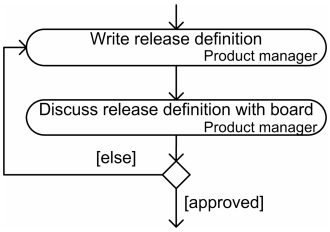

Fig. 8. Method fragment linked to 'Requirements validation per release'

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

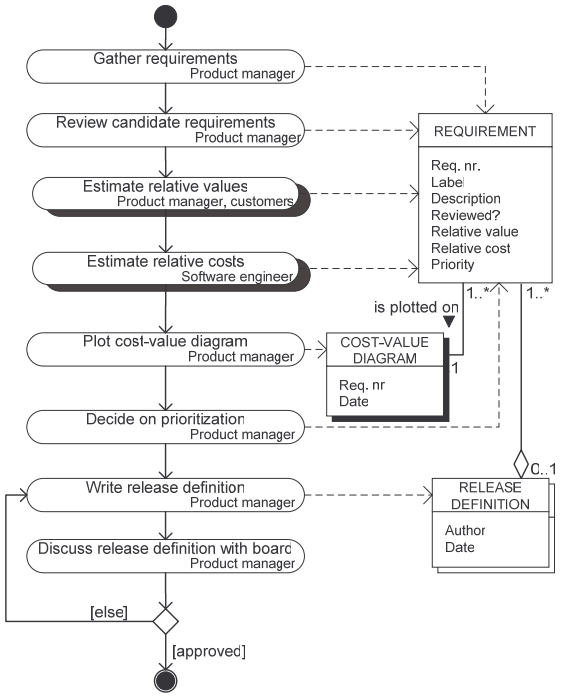

Fig. 9. Snapshot of increment #1

Finally, we want to show how we created the method fragments we presented in this paper. In Figure 10 we show a screenshot of MetaEdit+, in which a new method is modeled. Looking at the scenario we explained in Section 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.

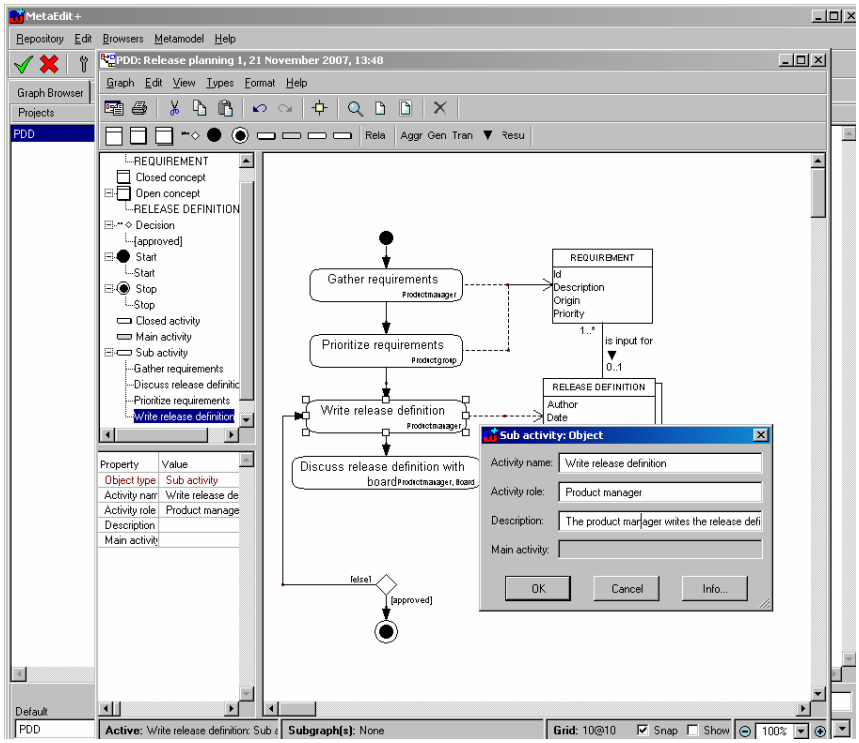

Fig. 10. Assembly of the improved method in MetaEdit+

5. Related Literature

In [30], it is stated that there is a scarcity of requirements engineering-related software process improvement initiatives in the literature. In addition, in [31] and [32], the authors 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, [33] describes a requirements engineering process improvement programme, based on lessons learned from the implementation of a requirements engineering approach for packaged software. The authors in [34] 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 [35] [36] [37], 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 [26]. 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.

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.

References

1. Nuseibeh, B. and Easterbrook, S.: Requirements engineering: a roadmap. In: Proceedings of the Conference on the Future of Software Engineering. ICSE '00. ACM, New York, NY, 35--46 (2000)
2. Karlsson, J., Ryan, K.: A cost-value approach for prioritizing requirements. IEEE Software 14, 67–74 (1997)
3. van de Weerd, I., Brinkkemper, S., Nieuwenhuis, R., Versendaal,J., Bijlsma, L.: Towards a Reference Framework for Software Product Management. In: Proceedings of the 14th IEEE International Requirements Engineering Conference, pp. 319--322 (2006)
4. Xu, L. & Brinkkemper, S.: Concepts of product software. European Journal of Information Systems 16, 531--541 (2007)

5. MacCormack, A.: Product-Development Practices that Work: How Internet Companies Build Software. MIT Sloan Management Review 42, 75--84 (2001)
6. Hietala, J., Kontio, J., Jokinen, J.P., Pyysiainen, J.: Challenges of software product companies: results of a national survey in Finland. In: Proceedings of the 10th IEEE International Symposium on Software Metrics, pp. 232--243 (2004)
7. Ebert, C. The impacts of software product management. Journal of Systems and Software 80, 850--861 (2007)
8. Regnell, B., Höst, M., Nat och Dag, J.N., Beremark, P., Hjelm, T.: An Industrial Case Study on Distributed Prioritisation in Market-Driven Requirements Engineering for Packaged Software. Requirements Engineering 6, 51--62 (2001)
9. van den Akker, M., Brinkkemper, S., van Diepen, G., Versendaal, J.: Flexible Release Planning Using Integer Linear Programming. In: Proceedings of the 11th International Workshop on Requirements Engineering for Software Quality, pp. 13--14. Essener Informatik Beitrage, Band 10 (2005)
10. Ruhe, G. & Saliu, M.O. The art and science of software release planning. IEEE Software 22, 47--53 (2005)
11. Pohl, K., Böckle, G., van der Linden, F.: Software Product Line Engineering: Foundations, Principles, and Techniques. Springer-Verlag, Heidelberg (2005)
12. Kang, K.C., Lee, J. & Donohoe, P.: Feature-oriented product line engineering. IEEE Software, 19, 58--65 (2002)
13. Jansen, S., Ballintijn, G., Brinkkemper, S., van Nieuwland, A.: Integrated development and maintenance for the release, delivery, deployment, and customization of product software: a case study in mass-market ERP software. J. Softw. Maint. Evol.: Res. Pract 18, 133--151 (2006)
14. Paulk, M.C., Weber, C.V., Curtis, B., Chrissis, M.B.: The capability maturity model: guidelines for improving the software process. Addison-Wesley Longman Publishing, Boston, MA (1995)
15. El Emam, K., Drouin, J.N., Melo, W.: SPICE: the theory and practice of software process improvement and capability determination. IEEE Computer Society Press , Los Alamitos, CA (1998)
16. Process Maturity Profile Software CMM 2005 End-Year Update. Retrieved at March 22, 2008, from: http://www.sei.cmu.edu/appraisal-program/profile/pdf/ SW-CMM/2006marSwCMM.pdf (2006)
17. van de Weerd, I., Versendaal, J. & Brinkkemper, S.: A product software knowledge infrastructure for situational capability maturation: Vision and case studies in product management. In: Proceedings of the Twelfth Working Conference on Requirements Engineering: Foundation for Software Quality, 97--112. Luxembourg (2006)
18. van de Weerd, I., Brinkkemper, S., Versendaal, J.: Concepts for Incremental Method Evolution: Empirical Exploration and Validation in Requirements Management. CAiSE 2007. LNCS, vol. 4495, pp. 469--484. Springer, Heidelberg (2007)
19. Hevner, A.R., March, S.T., Park, J., Ram, S.: Design Science in Information Systems Research. MIS Quarterly 28, 75--105 (2004)
20. March, S.T. & Smith, G.F. Design and natural science research on information technology. Decision Support Systems 15, 251--266 (1995)
21. Leszak, M., Perry, D.E., Stoll, D.: A case study in root cause defect analysis. In: Proceedings of the 22nd international conference on Software engineering, pp. 428--437 (2000)
22. Demirors , O., Demirors , E.: Software process improvement in a small organization: Difficulties and suggestions. Software Process Technology. LNCS 1487, pp. 1--12. Springer, Heidelberg (1998)

23. Staples, M., Niazi, M., Jeffery, R., Abrahams, A., Byatt, P., Murphy, R.: An exploratory study of why organizations do not adopt CMMI. Journal of Systems and Software 80, Issue 6, 883--895 (2007)
24. van de Weerd, I., Brinkkemper, S.: Meta-Modeling for Situational Analysis and Design Methods. Handbook of Research on Modern Systems Analysis and Design Technologies and Applications, Information Science Reference, Hershey PA (2008)
25. Tolvanen, J.: MetaEdit+: integrated modeling and metamodeling environment for domain-specific languages. In: OOPSLA '06, pp. 690--691. ACM, New York (2006)
26. Kelly, S., Lyytinen, K. & Rossi, M.: MetaEdit+: A Fully Configurable Multi-User and Multi-Tool CASE and CAME Environment. CAiSE 1996. LNCS, vol. 1080, pp. 1--21. Springer, Heidelberg (1996)
27. Steenbergen, M., Brinkkemper, S., van den Berg, M.: An Instrument for the Development of the Enterprise Architecture Practice. In: Proceedings of the 9th International Conference on Enterprise Information Systems, pp. 14--22 (2007).
28. Koomen, T., Pol, M.: Test Process Improvement: A Step-by-step Guide to Structured Testing. Addison-Wesley Longman Publishing, Boston, MA, (1999)
29. Lehtola, L., Kauppinen, M.: Suitability of Requirements Prioritization Methods for Market-driven Software Product Development. Software Process: Improvement and Practice 11, 7--19 (2006)
30. Damian, D., Zowghi, D., Vaidyanathasamy, L., Pal. Y.: An Industrial Case Study of Immediate Benefits of Requirements Engineering Process Improvement at the Australian Center for Unisys Software. Empirical Softw. Eng. 9, 45--75 (2004)
31. Sawyer, P., Sommerville, I., Viller, S.: Requirements process improvement through the phased introduction of good practice. Software Process Improvement and Practice 3, 19--34 (1997)
32. Niazi, M.K.: Software Process Improvement: A Road to Success. In: Proceedings of the Seventh Australian Workshop on Requirements Engineering, pp.125—139 (2002)
33. Regnell, B., Beremark, P., Eklundh, O.: A Market-driven Requirements Engineering Process: Results from an Industrial Process Improvement Programme. Requirements Engineering 3, 121--129 (1998)
34. Beecham, S., Hall, T. Defining a Requirements Process Improvement Model. Software Quality Journal 13(3), 247--279 (2005)
35. Brinkkemper, S. Method Engineering: Engineering of Information Systems Development Methods and Tools. Inf. and Softw. Techn. 38, 275--280 (1996)
36. Kumar, K. & Welke, R.J.: Methodology Engineering: a proposal for situation-specific methodology construction. In: Challenges and strategies for research in systems development 257--269. John Wiley & Sons, Inc. New York, NY (1992)
37. Rolland, C., Prakash, N.: A proposal for context-specific method engineering. In: Proceedings 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, 191--208 (1996)