A Product Software Knowledge Infrastructure for Situational Capability Maturation: Vision and Case Studies in Product Management

Inge van de Weerd, Johan Versendaal and Sjaak Brinkkemper

Department of Information and Computing Sciences
Utrecht University
Technical Report UU-CS-2006-008
www.cs.uu.nl
ISSN: 0924-3275

A Product Software Knowledge Infrastructure for Situational Capability Maturation: Vision and Case Studies in Product Management

Inge van de Weerd, Johan Versendaal and Sjaak Brinkkemper

Department of Information and Computing Sciences
Utrecht University
Utrecht, The Netherlands
{i.vandeweerd, j.versendaal, s.brinkkemper}@cs.uu.nl
http://www.cs.uu.nl

Abstract. Product software companies face the challenge of shipping new releases 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 started to build 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. Many product software companies have identified product management as a major function to deal with matters of release and requirements management. Therefore the infrastructure focuses particularly on these processes. In building the infrastructure we performed case studies at two companies. We found that product management processes change over time, and have become more mature. As such the study of evolution of product management processes is a promising step in building the full product software knowledge infrastructure.

Keywords. method engineering, meta-modeling, situational capability maturity, knowledge infrastructure

1 Situational maturity in product software companies

Product software 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 [6], time-to-completion factors are studied in 37 product software companies. Also large organizations, like Netscape and Microsoft, come across difficulties in their product development process [9].

Product software companies face complex challenges: an existing customer asks for a bug-fix, and usable features. A sales manager asks for features that convince buyers; 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 [8] 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 product software companies. As many product software companies have a separate product management function to cope with the mentioned challenges, we relate our infrastructure to product management. 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 [12], and the concept of evolving maturity (companies can grow in maturity with respect to product management).

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 using concepts of method engineering and situational capability maturation?

We address this question by elaborating our vision on situational maturity. Next we map this onto management, and method engineering. Subsequently we describe interview results of three product software companies and identify their product management processes as applicable today and in the past (different point in time). With the assumption that product software companies have improved their product management processes 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.

In describing the product management processes we use a meta-modeling technique successfully used earlier in describing software product implementation processes. This technique is used to reveal the relations between activities (the process) and data (the deliverables produced in the process) of the method. This makes it possible to configure both the process and data perspective of the method. Also, this technique provides the ability to fragmentize method into method fragments that can easily be reused.

1.2 Situational maturity vision

The typical evolutionary growth of a product software company goes hand in hand with the evolution of its internal processes for product development, marketing, sales, implementation services, and support. Each time a weakness in these processes is identified, a small improvement is made.

Important constraints 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.
- *Company condition*. Process execution is heavily influenced by the status of the overall product software company.
- *Organizational culture*. Most companies have their own languages, operational style, and technical platform and tools.

Methodical support for product software 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.

1.3 Related literature

In product software 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 [9].

Several software process improvement approaches exist. CMM is a framework representing a path of improvements recommended for software organizations that want to increase their software process capability [24]. 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) [7]. However, some organizations find CMMI too heavy and difficult to use [22]. Another software improvement model is ISO 9001, the model for quality assurance in design, development, production, installation and servicing [15]. 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 [24]. A third approach, ISO/IEC TR 15504, also known as SPICE (Software Process Improvement and Capability dEtermination), is used for software process assessment, capability determination and process improvement [16].

Method engineering is the engineering discipline to design, construct and adapt methods, techniques and tools for the development of information systems [4]. Most research in method engineering is focused on the situation of the project at hand, see e.g. [26], 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 [29]. This technique is based ona meta-modeling technique for the purpose of attaching semantic information to the artifacts and for measuring their quality using this information [28]. 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 [14]. 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 An knowledge infrastructure for product software

To support product software companies with their processes, we develop a Product Software Knowledge Infrastructure (PSKI). With this infrastructure, product software companies can obtain a custom-made advice that helps them to improve their processes. A schematic overview of the PSKI is illustrated in Figure 1. We recognize two main elements in this figure: the PSKI and the product software company.

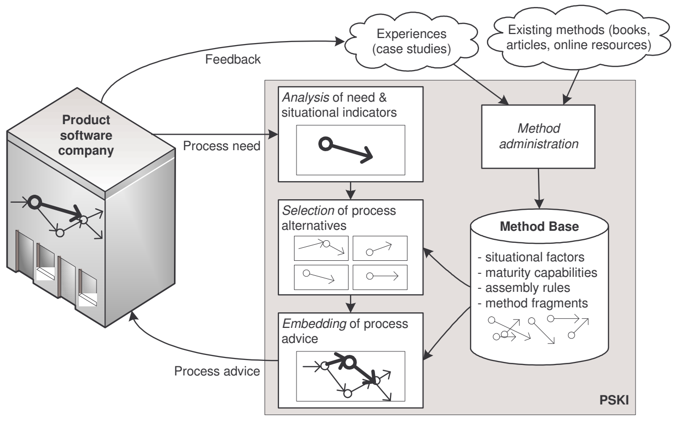

Experiences (case studies)
Existing methods (books, articles, online resources)
Feedback
Analysis of need & situational indicators
Product software company
Process need
Method administration
Selection of process alternatives
Method Base
- situational factors
- maturity capabilities
- assembly rules
- method fragments
Embedding of process advice
Process advice
PSKI

Fig. 1. Product Software Knowledge Infrastructure

The PSKI is loaded with experiences (acquired via case studies) and existing methods in the product software 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 is any factor relevant for product development and product services. Examples are company size, branch and the number of submitted requirements per month, whether or not currently a waterfall-based method is used for product software development, etc.

- **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** – The methods that are administrated need to be divided into method fragments, in order to be able to easily reuse them [30]. In section 3.1 a modeling technique is provided for storing method fragments. Method fragments are related to situational factors and capability maturities.
- **Assembly rules** – Derived from the experiences and existing methods, assembly rules can be identified [5]. On the one hand, the matching process of method fragments originating from different sources is bound by rules. For example, 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 in when the associated requirements document is produced earlier.

All four information types are related to each other. The situational factors influence the desired maturity level, for which several capabilities are defined. Both capability maturities and situational factors affect the method fragment choice. The knowledge of the relations between these concepts is captured in the assembly rules, see Figure 2.

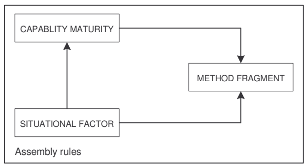


Fig. 2. Relations of the method base concepts

Starting point of the method engineering process is the product software company that has a need to improve one or more of its processes. 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 identified.

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 method is put together and the advice is *embedded* in the company. A process advice, which contains a process description, templates and examples, is sent to the product software company.

Afterwards, feedback from the company is used for a constant growth and improvement of the knowledge infrastructure. In section 4.5 we elaborate further on the PSKI by giving an example application of the infrastructure.

3 Infrastructure operationalization

3.1 Modeling technique

For the method administration, a meta-modeling technique is developed [29]. The method is based on UML [23]. With this technique we model processes on the left-hand side and data on the right-hand side.

In Figure 3, an example of a process-data diagram is depicted. The diagram represents the release planning activity and the release execution activity of the release management function. 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 requirement management activity. The process ends with the delivery of a new RELEASE.

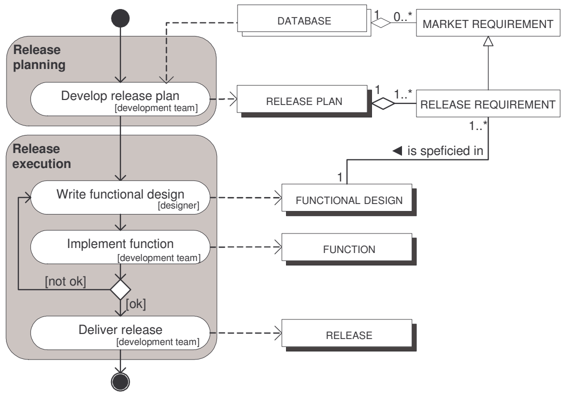


Fig. 3. Process-data diagram of a release management process

Some unique adjustments to the standard UML notation in both the activity diagram side and class diagram side have been made. The most important adjustment 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; see for example the concept 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 stan-

dard format exists and the content differs from time to time. All we know is that a number of RELEASE REQUIREMENTS are specified in it.
For other details of the meta-modeling technique, we refer to [29] and [30].

3.2 Product Management and the release process

For product software companies product managers are crucial [8], but their role is complex. In many companies product managers seem to have all the responsibility and none of the authority. Moreover, they receive many triggers and suggestions from the stakeholders to enhance or change their products. Whatever (automated) support product managers can receive is hence valuable. In this paper we focus on processes of product management.

In 1996, a research aimed at improving software product management in three SMEs, recognized that although product management is used in technical industries since the 19th century, this perspective in the software industry is a new approach [19].The last decade more and more software companies recognize the importance of product management. The area of product management and its support by professional tools is increasingly addressed by scholars. Particularly, [1], [17], [18] and [27] address the process of release definition from a computational perspective: they define algorithms to determine the content of a next release. In our research we focus on the whole process and human activities of the release management process of product management. Other product management functions can be recognized; examples are product roadmapping, and portfolio management. As releasing a product version is a complex process during which much can go wrong we focus particularly on *release management*.

3.3 A maturity framework for product management

To be able to provide 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 [24]. We use CMM as an important source 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 [3]. 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 [9].

Related to CMM, in Table 1 we distinguish maturity levels for product management, derived from literature [8] [12]:

Table 1. Maturity levels for product management

	Associated PM 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. Both the product management processes and products are quantitatively understood and controlled from an organization wide perspective.
	Product (line) orientation The different product management processes are standardized, documented and integrated into one standard product management process. Only approved, tailored versions of the organization's standard product management processes are used. 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.

The release management process 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 management, release planning and release execution. Without pretending to be complete, we first identify the capabilities for this function. These capabilities come from literature,[8], [11] and [12], from the second and third author's years of experience in product management in software industry, from the authors' personal network in the Netherlands through the Platform of Product Software companies (www.productsoftware.nl), and from the case studies discussed in detail in the next section.

In Table 2, we list the capability maturity matrix. Every capability is linked to one or more maturity levels. We define this as the *maturity level range*. With each capability / maturity level combination come method fragment(s). An example is the capability *functional design document construction*. The maturity level range covers the second, third and fourth levels. For the release-oriented level, this capability indicates that per release a functional design document should be delivered. An associated method fragment for this capability deals with communication for the determination of the release with members from the team, and especially the project manager. Which method fragment will eventually be selected for the company, depends on the situational factors. On the product level also the product perspective (among others earlier releases, anticipated release in the future) is covered for this capability. The accompanying method fragments treat features from earlier releases, as well as roadmapped release themes. Finally, on the integrated level, other departments and products are involved in constructing the functional design document.

Table 2. Capability maturity matrix for the release management process

	PM maturity level				
Capability	Ad hoc	Release oriented	Product oriented	Organization oriented	External oriented
Regulatory acceptance for release	x	x	x	x	x
Pro-active customer needs determination for release					x
Re-active customer needs determination for release	x	x	x	x	
Competitive product strategy determination					x
Trust establishment with customers for release					x
Distribution partner determination	x	x	x	x	x
Scope change management	x	x	x	x	x
Release promotion determination	x	x	x	x	x
Sales input for release				x	x
Product sales kit creation and sales training				x	x
Service input for release				x	x
Service department training				x	x
R&D input for release				x	x
R&D instruction for release				x	x
Pricing / price determination	x	x	x	x	x
Strategic platform, localization and languages determination	x	x	x	x	x
Functional design document construction	x	x	x	x	
Release requirements document construction	x	x	x	x	x
Clear definition of team responsibilities	x	x	x		
Prioritization of requirements	x	x	x	x	x
Collateral documentation	x	x	x	x	x
Validation of release requirements document		x	x	x	x
Validation of functional design document		x	x	x	x
Manage structured requirements database		x	x	x	x

4 Case studies

4.1 Context

The case studies reported in this section have been carried out in the context of the Dutch Platform for Product Software (www.productsoftware.nl), an initiative of a number of product software developers 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. We conducted two case studies with the purpose to model the method fragments and to identify the situational factors and more detailed capability maturities. These results are used to fill the method base. We also complemented and tested the validity of the capability maturity matrix in Table 2.

The case studies were carried out at two international product software companies in The Netherlands. The companies are 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 HRM and payroll administration (HRM Software). The other company focuses on facility management (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 was organized to affirm the results.
- **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.
- **Studying the software** – Several software programs are used to store requirements. The functionality of these software programs was studied with respect to their role in the concerning product management process.

A second interview session was conducted to cross-reference the obtained results of the first interview and the documentation. To increase the reliability of the case studies, we maintained a case study database containing raw data such as interview notes, case study documentation and process-data diagrams of the researched methods.

The case studies provided us with useful data. 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 maturing capabilities. Each of the case studies resulted in a snapshot of the release management process in the past (a few years back), and in the present. This provided us information on the evolvement of the release management process.

4.2 Situational factor analysis

In this section, we elaborate on the extraction of situational factors from the case studies. To illustrate this process, we describe the analysis of the situational factors extracted 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
- customer amount: 600
- customer base: SMEs, salary processing & accountancy, healthcare

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. On the other hand, changing a process in a small company is much easier than in a large organization. These considerations should 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 clear overview. 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 first. This situational factor leads to the capability *Prioritization of requirements*.

4.3 Method 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 4, the same process example as in section 3.1 is illustrated, with the difference that the process snapshot is taken a few years later. 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. Also, a prioritization activity is added to decide which PRODUCT REQUIREMENTS will be in the new release. The extension is marked by giving the new activity and data a thicker line.

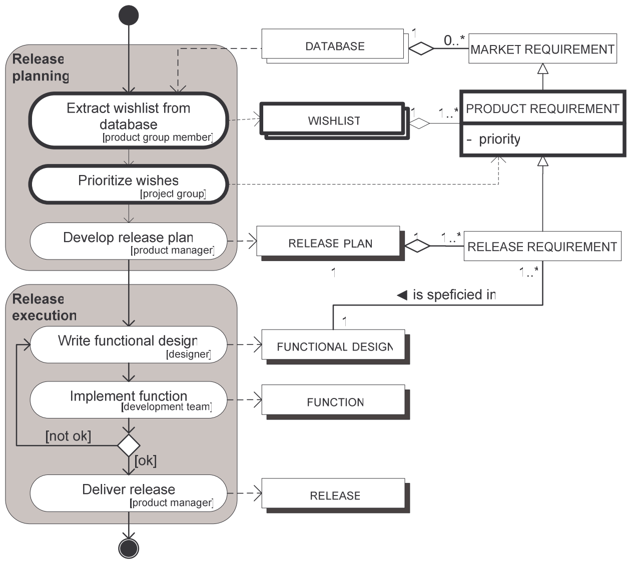


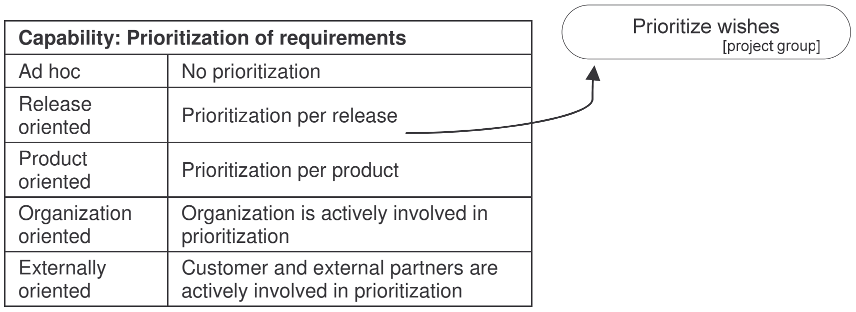
Fig. 4. Product-data diagram of a release planning and execution process

Another difference is the fact that clear roles are assigned to the sub-activities, instead of giving the whole development team the responsibility for one role. The release execution activity has stayed the same.

Summarizing the results, the release management process of HRM Software in our first case study company matured in four years from ad hoc to (mostly) the release-oriented level, and the release management process of FacMan Software in our second case study company matured in six years from ad hoc to (partly) the product-oriented level.

4.4 Capabilities analysis

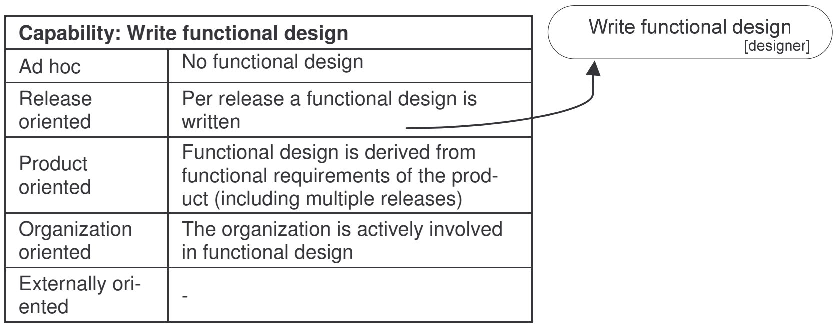
When we project the difference between the two process snapshots, described in the section above, to our capability maturity matrix, we can see that the release planning activity has matured. In the first snapshot prioritization is nowhere in the process. Likely, this was done implicitly by the product 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 5.



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	Customer and external partners are actively involved in prioritization

Fig. 5. Capability-method fragment mapping: Prioritization of requirements

Another example is the activity Write functional design. In our capability maturity matrix we can see that this capability ranges from ad-hoc to organization-oriented maturity. In our case the functional design is now written on release-oriented level. This implies a level 2 capability which described the production of a release oriented functional design document, as can be seen in Figure 6.



Capability: Write functional design	
Ad hoc	No 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	The organization is actively involved in functional design
Externally oriented	-

Fig. 6. Capability - method fragment mapping: Functional design

4.5 Example application of the infrastructure

To more integrally illustrate the PSKI in operation, we use a fictive example based on the snapshots of the case study we conducted at HRM Software. This product software company notices that the amount of new requirements suggested by customers has risen explosively the last years. The product manager needs a way to structure the requirements in order to improve the release management process. 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 determined as ad hoc.

The PSKI stores the information and compares it with the existing situation data in the method base. By comparing the situational indicators and desired maturity of the company with the situational factors and capability maturities in the method base one matching method fragment is selected, namely the release-oriented fragment *Prioritization of requirements*.

In the next step the new method, containing this selected method fragment, is put together and embedded into the existing process, analogous with Figure 4. This is presented to the company in an advice report. This report contains a process description of the release planning process; a document template for the prioritization activity, which can be used by the project team members to indicate the requirements that have their priority; and a filled in prioritization 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.

5 Conclusions and further research

Our research question, stated in section 1.1 is: how can product software 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 product software companies.

We introduced the Product Software Knowledge Infrastructure PSKI), to enable product software 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 product software companies is 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. By performing literature studies and 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. In the future, we will make it operational and test it at product software 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.

References

[1] Akker, M. van den, Brinkkemper, S., Diepen, G., Versendaal, J.: Flexible Release Planning Using Integer Linear Programming. In: Proceedings of the 11th International Workshop on Requirements Engineering: Foundation for Software Quality (2005)

[2] Batenburg, R., Versendaal, J.: Business Alignment in the CRM Domain: Predicting CRM Performance. In: Proceedings of the 12th European Conference on Information Systems (2004)

[3] Beecham, S., Hall, T., Rainer, A.: Defining a Requirements Process Improvement Model. Software Quality Journal, Vol. 13. Springer Science & Business Media (2005) 247–279

[4] Brinkkemper, S.: Method Engineering: Engineering of Information Systems Development Methods and Tools. Information and Software Technology, Vol, 38. Elsevier Science Publishers (1996) 275- 280

[5] Brinkkemper, S., Saeki, M., Harmsen, F.: Meta-Modelling Based Assembly Techniques for Situational Method Engineering. Information Systems, Vol. 24 (1999) 209-228

[6] Carmel, E.: Time-to-completion factors in packaged software development. Information & Software Technology, Vol. 37 (1995) 515-520

[7] CMMI Product Team, Software Engineering Institute Capability Maturity Model Integration (CMMI), Version 1.1, CMU/SEI-2002-TR-012 (2002)

[8] Condon, D.: Software Product Management – Managing Software Development from Idea to Product to Marketing to Sales. Aspatore Books Boston (2002)

[9] Cusumano, M.: The Business of Software. New Yokk: Free Press (2004)

[10] Dooley, K., Subra, A., Anderson, J.: Maturity and its Impact on New Product Development Project Performance. Research in Engineering Design, Vol. 12. Springer-Verlag London Ltd (2001) 23-29

[11] Dver, A. S.: Software Product Management Essentials, Anclote Press, Tampa, Fla. (2003)

[12] Gorchels, L.: The Product Management Handbook – The Complete Product Management Resource (2nd edition). NTC Business Books, (2000)

[13] Harmsen, F., Brinkkemper, S., Oei, J. L. H.: Situational Method Engineering for Informational System Project Approaches. In: Proceedings of the IFIP WG8.1 Working Conference on Methods and Associated Tools for the IS Life Cycle (1994) 169-194

[14] Helms, R.W., Brinkkemper, S., Oosterum, J. van, Nijs, F. de: Knowledge Entry Maps: Structuring of Method Knowledge in the IT Industry. In: Kiyoki, Y., Kangasslo, H., Jaakkola, H., Henno, J. (eds.): Proceedings of the 15 European Japanese Conference on Information Modelling and Knowledge Bases. Amsterdam: IOP Press (2005) 12-25

[15] ISO 9001. Quality systems - Model for Quality Assurance in Design, Development, Production, Installation and Servicing, ISO, Geneva, Switzerland (1997)

[16] ISO/IEC-15504. Information Technology - Software Process Assessment. Technical Report - Type 2 (1998)

[17] Jung, H.-W.: Optimizing Value and Cost in Requirements Analysis, IEEE Software, Vol. 15 (1998) 74-78

[18] Karlsson, J., Ryan, K.: A Cost-Value Approach for Prioritising Requirements, IEEE Software (1997) 67-74

[19] Kilpi, Tapani, 1997: Product Management Challenge to Software Change Process: Preliminary Results from Three SMEs Experiment, Software Process - Improvement and Practice, Vol. 3. John Wiley & Sons, Ltd. (1997) 165-175

[20] Kumar K., Welke R.J.: Methodology Engineering: A Proposal for Situation-Specific Methodology Construction. In: Cotterman, W. W., Seen, J. A. (eds.): Challenges and Strategies for Research in Systems Development. John Wiley & Sons Ltd (1992)

[21] Natt och Dag, J., Gervasi, V., Brinkkemper, S., Regnell, B.L: A Linguistic Engineering Approach to Large-Scale Requirements Management, IEEE software, Vol. 22 (2005) 32-39.

[22] Nawrocki, J.R., Walter, B., Wojciechowski, A.: Comparison of CMM level 2 and eXtreme programming. In: 7th European Conference on Software Quality. Springer (2002)

[23] OMG: UML 2.0 Superstructure Final Adopted specification, Document reference ptc/04-10-02 (2003) Available from the Object Management Group website: www.omg.org.

[24] Paulk, M. C.: How ISO 9001 compares with the CMM. In: IEEE Software, Vol. 12. (1995) 74-83.

[25] Paulk, M.C., B. Curtis, M.B. Chrissis, C.V. Weber. Capability Maturity Model, Version 1.1. IEEE Software, 10, 4 (1993) 18-27

[26] Ralyté, J,, Deneckère, R. and Rolland, C. (2003). Towards a Generic Model for Situational Method Engineering. In: Lecture Notes in Computer Science, Vol. 2681. Springer-Verlag, (2003) 95-110

[27] Ruhe, G., Saliu, M. O.: The Science and Practice of Software Release Planning, IEEE Software (2005)

[28] Saeki M.: Embedding Metrics into Information Systems Development Methods: An Application of Method Engineering Technique. Proceedings of the 15th Conference On Advanced Information Systems Engineering (2003) 374-389

[29] Weerd, I. van de: WEM: A Design Method for CMS-based Web Applications, Technical report UU-CS-2005-043. Institute of Computing and Information Sciences, Utrecht University, the Netherlands (2005)

[30] Weerd, I. van de, Souer, J., Versendaal, J., Brinkkemper, S.: Situational requirements engineering of web content management implementations. In: Proceedings of the 1st International Workshop on Situational Requirements Engineering Processes (2005) 13-30