Assembly-based Method Engineering with Method Patterns

Masud Fazal-Baqaie, Markus Luckey, Gregor Engels

s-lab – Software Quality Lab
Universität Paderborn
Zukunftsmeile 1
33102 Paderborn
{masudf,luckey,engels}@uni-paderborn.de

Abstract: Software development methods prescribe and coordinate the activities necessary to plan, build, and deliver software. To provide methods that account for the situational context of a development project, e.g., an acquirer-supplier-relationship or specific communication needs, the existing method creation approaches represent a trade-off between flexibility and ease of use. On the one side, less flexible configurable methods offer a fixed set of configurations to quickly adapt a method to the situation at hand. On the other side, assembly-based approaches allow creating methods from scratch by combining preexisting building blocks. Thus, they are more flexible and capable of creating methods not covered by configurations of configurable methods, e.g., a mixture of agile and plan-driven ideas. However, assembly-based approaches are not easy to use and require considerable expert knowledge. In this paper we suggest the use of method patterns during the assembly-based method creation. Method patterns represent desirable principles for the to-be-method and therefore support the right choice and combination of method building blocks, simplifying assembly-based method creation.

1. Introduction

Large software development projects often involve many stakeholders and different organizations. One example for such a project is the development of an ePassport system. An ePassport system covers all lifecycle phases of an ePassport, from the data collection during the enrollment of its holder, over its personalization (the "printing") to its delivery to its holder, its usage, and finally its destruction.

In order to successfully accomplish large software projects like ePassport projects, software engineering methods are applied. By software engineering methods we denote the full set of elements needed to describe a software development project, including the development process and its activities, the artifacts produced, and the tools and techniques that are employed as well as relationships between these concepts [ES10].

There exist several widespread software engineering methods based on different philosophies for different purposes, e.g., RUP [Kr99], V-Modell XT [Vm12] or Scrum [SS11]. However, even for one specific domain like the development of ePassport systems there

is no one-size-fits-all method. The very different nature and priorities of each project, i.e., the situational context [HR10], has an impact on the method's activities and artifacts. As an example, consider the trade-off between plan-driven activities and agility [Bo03]. That trade-off is influenced, e.g., by having an acquirer-supplier-relationship, like it is typical for ePassport projects, or by the stability of the requirements base. There may also exist certain regulatory constraints that have to be taken into account, e.g., to meet a certain process maturity level regarding CMMI [Cm10].

Situational method engineering (SME) is the field dedicated to engineering situation-specific software development methods from scratch or adapting existing methods [HR10]. The adaption of existing methods is often summarized under the term tailoring, disregarding the differences between unrestricted free adaption and guided configuration. Every SME approach represents an individual tradeoff between the effort to design a method and the flexibility in terms of possible choices during the method design process [HB94]. On the one side there exist more rigid approaches to create a project-specific method. For example, configurable methods like V-Modell XT [Vm12] offer several variability points to adjust the method to the situation. On the other side, less rigid SME approaches are more flexible, in terms of the variety and specialization of the creatable methods, however, the design of methods requires more effort and more expertise.

A particular group of these less rigid SME approaches is called assembly-based SME [BS98]. The basic idea is to maintain a repository of predefined method building blocks, e.g., called method fragments [Br96], method chunks [Ro96], method components [GL98], or as in our case method services [Ro09]. Based on the situational context, method building blocks suitable for the current project are selected and assembled to a method (see **Figure 1**). The flexibility of this approach is restricted only by the set of available building blocks. These can be defined on-the-fly without requiring changes to existing method building blocks. Hence, it is possible to incorporate, e.g. the latest best practices.

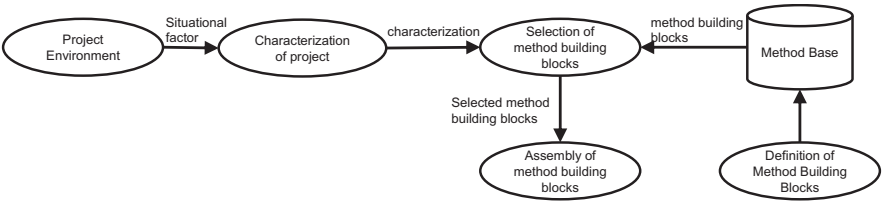

Figure 1. Assembly-based SME (cf. [Br96])

The major drawback of assembly-based SME is that creating meaningful methods requires a certain level of method engineering knowledge as it is more tedious and error-prone than configuration-based SME, where the possible configurations are already known beforehand. In this we see the main reason why assembly-based SME has not achieved noticeable attention in industry. Suitable building blocks have to be identified and combined in such a way that a consistent method is created. Furthermore, the method has to comply with the requirements imposed by the situation. To benefit from the

advantages of assembly-based SME, we introduce the new concept of method patterns to support the method engineer following that approach in his work.

Method patterns represent methodological aspects and quality constraints that shall be incorporated into the method, e.g., "iterative development" or "use of quality gates". They are combined to form a "method frame", which ensures that the combined method building blocks do not violate the pattern-specific properties. For example, a method pattern that prescribes the creation of a specification can be combined with a method pattern for iterative development. That ensures that in the created method that specification is created iteratively.

This paper is organized as follows. In Section 2 we use ePassport system development as an example to illustrate the rationale for assembly-based SME with method patterns. In Section 3 we exemplify the use of method patterns by combining patterns of a plan-driven software engineering method with patterns reflecting agile aspects. We conclude with a discussion of our contributions and the planned future work in Section 4.

2. Motivational Scenario

We use the following scenario as a running example to illustrate why configuration-based SME may fall short and to motivate assembly-based SME and the benefits of method patterns. It is based on real life industry projects carried out by one of our co-operation partners. The project in the scenario deals with the introduction of a distributed ePassport system connected to several national (e.g. border control, civil register) and international (e.g. Interpol) databases and information systems. Typically, such a system is not developed by the government organization itself, but by a supplier that is awarded the project after a public tender of the government organization (acquirer). Normally, a passport domain expert is the project manager. Right before the project's start the expert chooses the software development method of the project. As domain experts usually have no particular SME knowledge they choose fixed off-the-shelf methods or create a method using configuration-based SME, e.g., V-Modell XT [Vm12].

Figure 2 illustrates the lifecycle of such a project with the decision gates of V-Modell XT-based methods.[1] Each decision gate, depicted by a left leaning parallelogram, marks the end of a lifecycle phase, where the produced deliverables are examined.

Different from other domains, V-Modell XT has not been established as a standard for ePassport system projects. Nevertheless, the scope and formality of the methods created with V-Modell XT define a frame for the legal and commercial cooperation of acquirer and supplier. The desirable characteristics include:

- the work division between acquirer and supplier, e.g., support of tender activities
- the definition of formal documents, e.g. for legal and commercial reasons

[1] As we use V-Modell XT as an illustrative example, we abstract from different project execution strategies.

- the definition of formal handover activities, e.g., for legal and commercial reasons
- coverage of the lifecycle presented in **Figure 2**

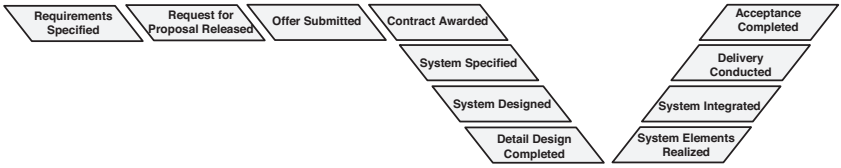

Figure 2. Sequence of decision gates in V-Modell XT-based methods

According to experience, the document-centric philosophy of methods created with V-Modell XT and their rigid formality are however also seen as the root for severe problems in practice. Typically, stakeholder are not familiar with ePassport technology and do not understand the implications caused by "just an additional chip on the passport". Therefore, they have difficulties to formulate all their requirements upfront and the requirements specification does not reflect the stakeholders' real intends. However, by the document-centric nature of a V-Modell XT-based method, the requirements specification is the main and dominant source of information for the supplier during the specification and development. Only little participation of the acquirer takes place during development and the stakeholders often do not see the system before it is ready to be delivered. Flaws uncovered then have to be removed at high costs.

In order to improve the situation, the method to use for the ePassport project shall therefore incorporate aspects of the agile software development philosophy [Be12] that fosters information exchange and collaboration. However, V-Modell XT is not designed to create a method that exhibits the following characteristics:

- iterative and incremental development towards decision gates
- informal coordination meetings
- sharing of intermediate work results

Using the scenario as an example we illustrated the limitations of configuration-based SME. Assembly-based SME, in contrast, allows incorporating method building blocks of different methods, especially following different development philosophies, as requested in our example. However, the available literature on assembly-based approaches provides no formal guidance for people without any particular SME knowledge during the method construction (e.g., [GH98, Fi09]). Additionally, if the effort to create a method is too high, the project manager will not be able to timely create a method for the initiated project. We therefore propose the concept of method patterns, which are used additionally to method building blocks during the method construction. They encode methodological aspects and quality constraints like the required presence and order of activities. In our Scenario the project manager could use and combine these patterns to assure that the method creates all the documents required by V-Modell XT and additionally shows the desired agile characteristics. Violated constraints of method patterns provide him with additional guidance during the method construction.

3. Situational Method Engineering with Method Patterns

In this section we exemplify the use of method patterns by the assembly-based creation of a method that incorporates the desired characteristics described in Section 2. On the one hand we define two *method patterns* that embody the essential constraints of these methods. For V-Modell XT we create a method pattern that reflects the development process of V-Model XT with the order of its decision gates. For Scrum we create a method pattern for the sprint loop. On the other hand we define *method services*, method building blocks that reflect the software development activities and artifacts of these methods. Recall that method patterns constrain the assembly of method building blocks (method services). By combining the method patterns and respective method services we could reconstruct the two original methods. However, we show how the combination of method patterns from both methods guides the method engineer to create a hybrid method, which maintains the order of document creation conforming to V-Modell XT, but ensures that they are developed in sprint loops.

3.1 Extraction of Method Patterns and Method Services from V-Modell XT

As stated in Section 2 methods created with V-Modell XT define a flow of decision gates that have to be passed to accomplish the project (see **Figure 2**). In V-Modell XT at each decision gate a set of documents has to be approved using the activity "Project coming to a progress decision". For example, **Figure 3** shows on the right the three documents that have to be approved for the decision gate "System Specified" depicted on the left. Consequently, these documents have to be produced before the decision gate can be passed. Thus, in methods based on V-Model XT the sequence of decision gates indirectly specifies the order of activities that have to be performed. For example, for the decision gate "System Specified*"* the document "Overall System Specification" has to be created with a software development activity called "Preparing Overall System Specification" and it has to be approved like every other document using the activity "Project coming to a progress decision".

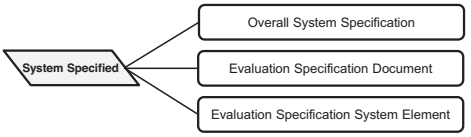

Figure 3. Decision gate "System Specified" and related documents of V-Modell XT

We now translate the flow of decision gates into a method pattern, by first creating a method pattern for every decision gate and then combining them into an overall method pattern that reflects the V-Modell XT development process. We later reuse this method pattern when creating the hybrid method.

Figure 4 illustrates the relationship between the constituents of a method pattern and method services. A method pattern consists of *method compartments*, denoted by dotted rectangles. These are restricted by the attached *pattern constraints*, depicted by grey

boxes. Pattern constrains restrict their respective method compartment, because the hosted method services must fulfill these constraints.

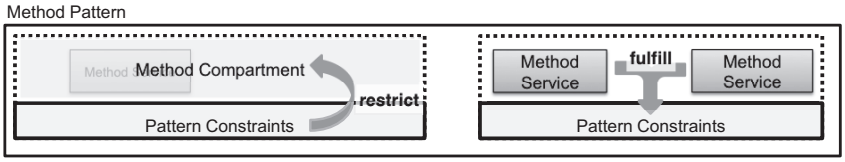

Figure 4. Overview of the relationship between method patterns, compartment, and services and pattern constraints

Figure 5 shows a method pattern that encodes the concrete decision gate "System Specified*"*. The method pattern consists of two consecutive method compartments. The first method compartment fulfills its pattern constraints, if it hosts a method service that has the respective artifact among its outputs, for each of the three artifacts named in **Figure 3**. The second method compartment has to contain a method service that has the value `reviewing` assigned to its attribute `activity type`. Thus, the method pattern describes, that method services have to create the three named documents and that they have to be followed by a method service encapsulating a reviewing activity. **Figure 6** shows a combination of method services that fulfills the constraints.

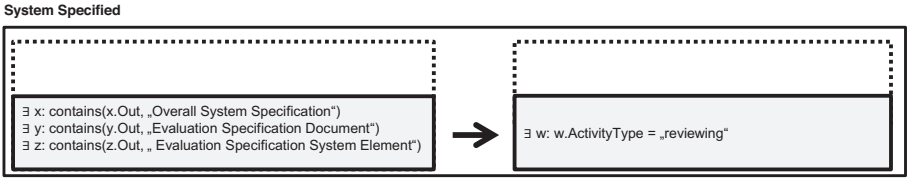

Figure 5. Method pattern for the decision gate "System Specified"

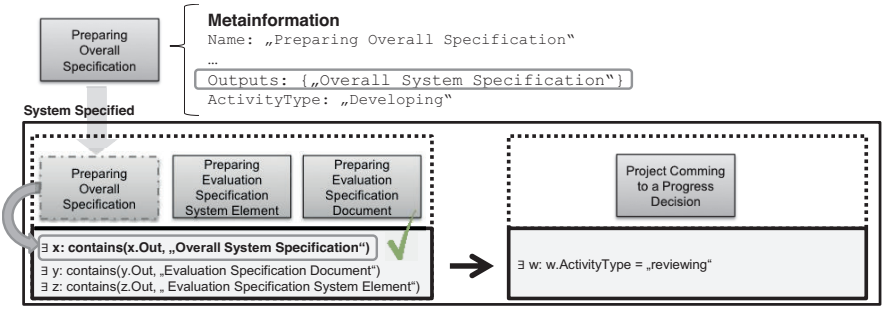

Figure 6. A method pattern with fulfilled pattern constraints

Each software development activity and its input and output documents are translated to a method service with respective input and output artifacts. For example, the method service "Preparing Overall Specification" encapsulates the equally named V-Modell XT activity and its output "Overall System Specification" (see **Figure 6**). Additional metain-

formation specifies that this is a development activity: the attribute `ActivityType` of the method service has the value `developing` assigned to it. Thus the first method service in the first method compartment "Preparing Overall Specification" fulfills the first pattern constraint as it has the required artifact among its outputs. Similar, the other three method services fulfill the remaining pattern constraints. **Figure 6** is only an example for a fulfilled method pattern; we do not combine method patterns and method services yet.

To obtain a method pattern that reflects the development process of V-Model XT with the order of its decision gates we chain the concrete method patterns of all decision gates to an overall "V-Modell XT" method pattern. **Figure 7** illustrates this for the three consecutive decision gates depicted on the left. The three method patterns are combined to a new method pattern that has no pattern constraints on its own, but specifies the order of the contained method patterns "System Specified", "System Designed" and "Detail Design Completed".

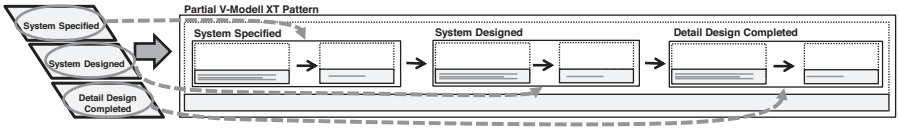

Figure 7. Decision gate sequence reflected as a method pattern

3.2 Extraction of Method Patterns and Method Services from Scrum

Scrum is a widespread agile development method that we use in our example to define agile method patterns and agile method services. One of the core aspects of Scrum is a time-boxed execution loop called "Sprint" that is repeated throughout the duration of the project. **Figure 8** shows the method pattern "Sprint Loop", which requires method services that reflect agile activities. The method pattern consists of the three sub method patterns "Sprint Planning", "Agile Construction" and "Sprint Review" that are combined to a loop. For example, "Agile Construction" describes that all the method services in the respective method compartment have to either encapsulate developing activities or contain the backlog artifact among their inputs. Additionally, the use of a method service named "Standup Meeting" is prescribed and has to be present in the method compartment.

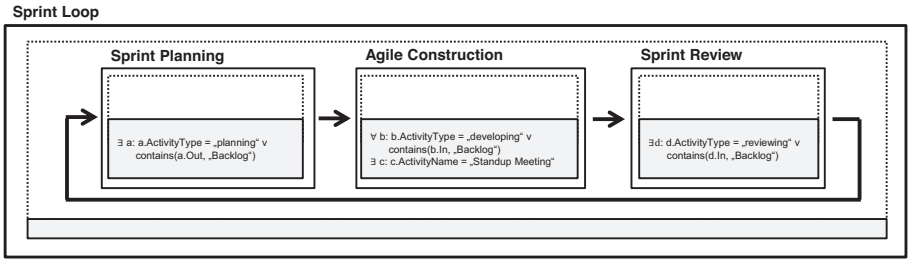

Figure 8. The Sprint Loop pattern extracted from Scrum

Based on the Scrum guide [SS11] the method services "Sprint Planning", "Standup Meeting", "Update Backlog" and "Sprint Review*"* are identified. They fit into the appropriate method compartments of the method pattern depicted in **Figure 8**.

3.3 Creation of a Situational Method for ePassport System Development

For a typical method creation procedure, the previously constructed and presented method patterns and method services would have been identified and retrieved from the method base instead of being defined from scratch (see **Figure 1**). The next step now is their combination. Different from traditional assembly-based approaches we first combine method patterns and then place method services into the method compartments of these patterns. In our example the Project Manager picks the overall "V-Modell XT" method pattern, to assure the conformance to the prescribed order of activities. In addition he adds a "Sprint Loop" method pattern into every decision gate method pattern, because he wants it to be executed in an agile manner. **Figure 9** illustrates this for the decision gate "System Specified". The combination of method patterns now prescribes and ensures that the created method will contain a "Sprint Loop" in every decision gate method pattern and that all necessary activities of V-Modell XT are executed in the right order. Compared to other assembly-based approaches, with this frame of method patterns it is much easier to decide, which method services to use and where in the process to put them. **Figure 10** shows the combination of method patterns after adding method services to fulfill the pattern constraints depicted in **Figure 9**.

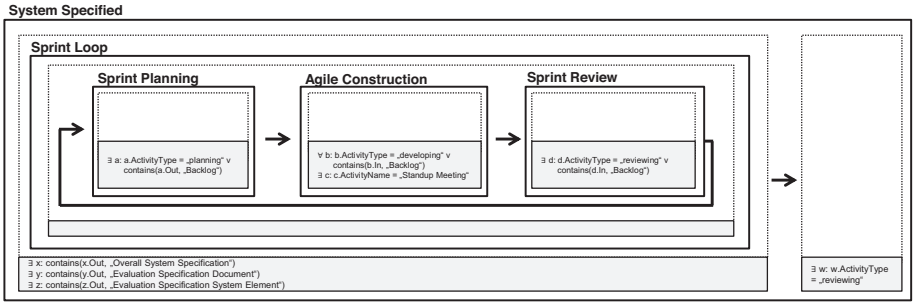

Figure 9. Combined method patterns derived from V-Modell XT and Scrum

With the method creation state shown in **Figure 10** there could be additional refinement iterations. For example, in the method compartment of "Agile Construction" illustrated in **Figure 11** the "Sprint Loop" method pattern could be used again to model the daily Scrum, which is a daily sprint loop, of the Scrum method. As this additional formality is not desired for this ePassport project, the method creation procedure is finished by (manually) connecting the method services with control flow. According to the practices in the Scrum Guide the control flow specifies that the work is carried out in a loop, where "Standup Meeting" precedes the parallel execution of the development method services. "Update Backlog" is executed continuously in parallel.

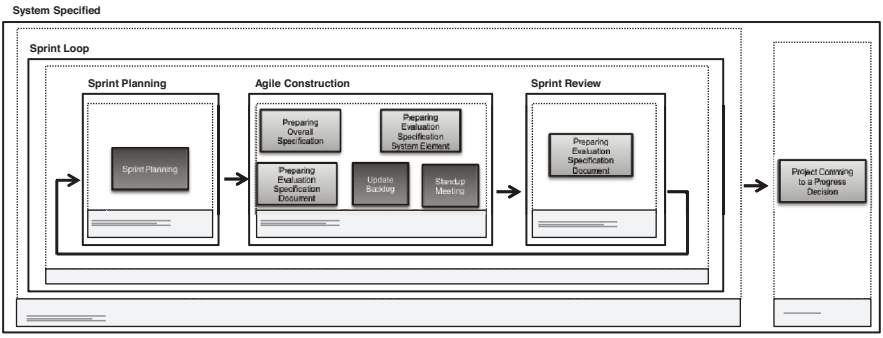

Figure 10. Combination of method services that fulfill the constraints of the method patterns in Figure 8

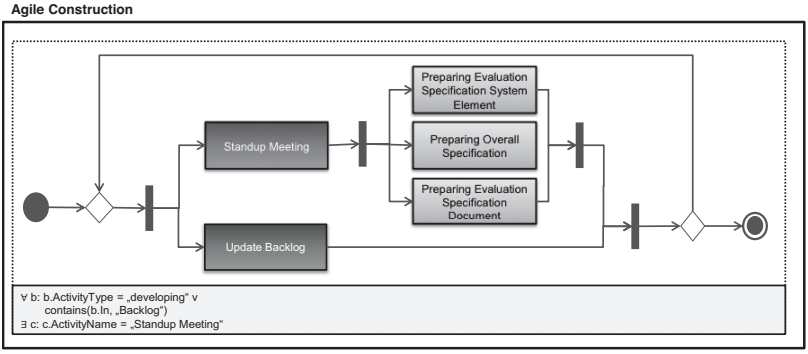

Figure 11. Method services of a method compartment connected with control flow

4 Conclusions and Future Work

Using a realistic example scenario from the ePassport system development domain we exemplify that in some cases more rigid SME approaches are not flexible enough to reflect the situational context, e.g., agile aspects in our example. Here, assembly-based SME provides the required flexibility, however, we criticize that assembly-based approaches require too much method engineering knowledge and offer insufficient support to create methods of good quality. We exemplified the use of method patterns and show how they can guide in choosing and combining suitable method building blocks, supporting the method engineer in his work. Doing so we also show how building blocks from different methods can be incorporated. The resulting created method preserves the flow of activities and the document-based approvals at specific decision gates known from V-Modell XT. In addition, by incorporating sprints of Scrum, there are fixed cycles, where results are planned, produced, presented, and discussed between the approvals of two consecutive decision gates.

Although we argue, that by the use of method patterns we have an advantage over pure assembly-based approaches, the additional freedom compared to more rigid approaches as configuration-based SME still requires more time and more skills to create the method. However, we will be able to gradually improve on the status quo in the future. We continue our work in two directions. In this paper we focused on activities and control flow. First, we work on formalizing other aspects of methods like roles, artifact lifecycles and object flow. Second, applying this approach in practice is feasible only with sufficient tool support. In a project with an industrial partner we work on an expert system that supports the different activities of method creation. We are also evaluating how such method specifications can be enacted in terms of a workflow engine, task management and integrated tooling like version control and plan to evaluate the whole approach in their industry projects.

References

[Be12] Beck, K. et al.: Manifesto for Agile Software Development, http://agilemanifesto.org/

[Bo03] Boehm, B.W., Turner, R.: Observations on Balancing Discipline and Agility. In: ADC 2003, pp. 32–39. IEEE Computer Society, Los Alamitos, Calif (2003)

[Br96] Brinkkemper, S.: Method engineering: engineering of information systems development methods and tools. Inf. Softw. Technol. 38, 275–280 (1996)

[BS98] Brinkkemper, S., Saeki, M., Harmsen, A.F.: Assembly Techniques for Method Engineering. In (Pernici, B., Thanos, C. eds.): CAiSE '98, pp. 381–400. Springer, Berlin (1998)

[Cm10] CMMI Product Team: CMMI for Development, Version 1.3. Improving processes for developing better products and services Pittsburgh, Pennsylvania (2010)

[ES10] Engels, G., Sauer, S.: A Meta-Method for Defining Software Engineering Methods. In (Engels, G., Lewerentz, C., Schäfer, W., Schürr, A., Westfechtel, B. eds.): Graph Transformations and Model-Driven Engineering, pp. 411–440. Springer, Berlin (2010)

[Fi09] Firesmith, D.G.: The method framework for engineering system architectures. CRC Press, Boca Raton (2009)

[GH98] Graham, I., Henderson-Sellers, B., Younessi, H.: The OPEN process specification. ACM Press, New York (1997)

[GL98] Goldkuhl, G., Lind, M., Seigerroth, U.: Method Integration: The Need For A Learning Perspective. IEE Proceedings Software 145, 113–118 (1998)

[HB94] Harmsen, F., Brinkkemper, S., J. L. Han Oei: Situational method engineering for informational system projects. In (Verrijn-Stuart, A.A., Olle, T.W. eds.): CRIS'94, pp. 169–194. North-Holland Publishers, Amsterdam (1994)

[HR10] Henderson-Sellers, B., Ralyté, J.: Situational Method Engineering: State-of-the-Art Review. j-jucs 16, 424–478 (2010)

[Kr99] Kruchten, P.: The rational unified process. An introduction. Addison-Wesley, Reading, Mass (1999)

[Ro96] Rolland, C., Prakash, N.: A proposal for context-specific method engineering. In (Brinkkemper, S., Lyytinen, K., Welke, R.J. eds.): Method Engineering: Principles of method construction and tool support, pp. 191–208. Chapman & Hall, London (1996)

[Ro09] Rolland, C.: Method engineering: towards methods as services. Softw. Process: Improve. Pract 14, 143–164 (2009)

[SS11] Schwaber, K., Sutherland, J.: The Scrum Guide (2011)

[Vm12] V-Modell XT (english version), http://v-modell.iabg.de/v-modell-xt-html-english/index.html