Method-User-Centred Method Configuration

Fredrik Karlsson[a] and Pär J Ågerfalk[b, a]

[a] Örebro University, MELAB, Dept of Informatics (ESI), SE-701 82 Örebro, Sweden, fredrik.karlsson@esi.oru.se, www.oru.se/esi/melab

[b] Dept of Computer Science and Information Systems, University of Limerick, Limerick, Ireland, par.agerfalk@ul.ie, www.b4step.ul.ie/web/Par.Agerfalk/

Abstract

Method engineering approaches are often based on the assumption that method users are able to explicitly express their situational method requirements. However, similar to software requirements, situational method requirements are often vague and hard to explicate. In this paper we address the issue of involving method users early during method configuration. This is done through borrowing ideas from user-centred design and prototyping and implementing them on the method engineering layer in a computerized tool support. This tool has proven useful in eliciting situational method requirements in a continuously ongoing dialog with the method users during configuration work-shops.

Keywords: Situational process, adaptation, configuration, method

1 Introduction

Method engineering (including method tailoring and adaptation) has received increased interest among practitioners. Given the trend to implement one commercial or brand-named software/systems development method (method for short) as the organization-wide method, this growing interest is natural since there is no method that fits all situations. As pointed out by Fitzgerald et al. (2003), there are hundreds of methods, although surprisingly little research has focused on how to tailor methods when used as organization-wide standard approaches.

In recognition of this, we can also conclude that method requirements vary between individual projects. Projects differ with respect to the development context, delivery, project team, deadline, et cetera. These characteristics constitute method requirements that need to be taken into consideration when tailoring a method. This is one major challenge when working with situational methods: how to handle the method requirements process? Method requirements are often elicited through interviews with project members. The method engineer then uses these requirements to define a situational method, which is introduced to the project team. For example, method requirements can be used in selecting method fragments (Brinkkemper et al., 1999) or method chunks (Rolland *et al.*, 1998) using modular method construction. In such a case, methods' atomic concepts (e.g. modelling primitives, such as 'class' and 'state') are seen as minimal method modules, which are selected and included in the situational method. Method fragments are stored in repositories and retrieved based on the characterization that is provided by the meta-language used. Consequently, method requirements need to be formalized and expressed in the meta-language.

This type of method engineering approach is anchored in at least two basic assumptions. The first assumption is that it is possible for project members to explicitly specify the required situational method and communicate the requirements to the method engineer. The second assumption is that these requirements do not

method engineer. The second assumption is that these requirements do not change during the project's lifetime. Evolutionary method engineering (Rossi et al., 2004) have addressed the latter using MetaEdit+ to change the situational method and corresponding tool support through the course of the project. However, the first assumption is still valid and practical experiences confirm that method requirements are often vague during the initial phases of a project. The problem is similar to that faced in requirements engineering as part of systems development, where knowledge has to be formalized and communicated between different various actors. In the systems development case, system requirements have to be communicated between end users and systems developers. In the method engineering case, method requirements have to be communicated between systems developers and method engineers.

When it comes to vague systems requirements, a commonly cited approach to bridge between end-users and system developers is user-centred design and prototyping. In order to make such an approach effective, appropriate tool support is imperative. In this paper we present an approach to situational method engineering that draws on user-centred design. The focus of this paper is mainly on MC Sandbox, a computerized tool support for method configuration that incorporates important design principles of method-user-centred method configuration. To arrive at these design principles, we elaborate on the design of an approach to capturing method requirements through the use of method-user-centred method configuration, hence bridging the gap between project members and method configurators[1]. The remainder of the paper is organized as follows. Section 2 presents the research method adopted. Section 3 is devoted to ideas from user-centred design and to a theoretical discussion of method configuration concepts. Section 4 presents MC Sandbox, a computerized tool support for method configuration that incorporates the design principles of method-user-centred method configuration. Section 5 reports empirical experiences related to the use of MC Sandbox. Finally, Section 6 concludes the paper with a short summary of the main points.

2 Research Method Adopted

The design of MC Sandbox is part of a larger research project on developing flexible, reusable yet project specific method support. This project includes the development of a theoretical framework and a Method for Method Configuration (MMC). These are operationalized in the MC Sandbox computerized tool, together with design ideals of how best to capture method requirements.

The phenomena discussed in this paper are largely artificial ones. This means that they can be both designed and studied. The design concept itself is a duality, it is both a process and a product (Hevner et al., 2004). The purpose of designing a product is to 'support achievement of goals' (Walls et al., 1992), in our case a method-user-centred requirements process for method tailoring. Since we are dealing with artificial phenomena, we have the possibility to affecting the ways of working that are operationalized in MC Sandbox. According to March and Smith (1995) 'design science products are of four types, constructs, models, methods, and implementations.' This means that the concepts and models found in MMC (the method) shapes the MC

[1] Method configuration is a special kind of method engineering where one specific method is the starting point for tailoring (Karlsson and Ågerfalk, 2004). Consequently, 'method configurator' is treated as a specialization of the method engineer role.

Sandbox implementation. Consequently, in order to achieve method-user-centred method configuration we had to involve qualified practitioners in the design of concepts, models, MMC and MC Sandbox.

Based on this fundamental view on knowledge development our choice of research strategy fell on what has been called 'collaborative practice research' (Mathiassen, 2002), a strategy often referred to as action research (McKay and Marshall, 2001). Action research can be viewed as collaboration between researchers and practitioners where the researchers share the problems and concerns of the practitioners. The collaborative action research strategy has been operationalized in four action cases where the concepts, models, MMC and MC Sandbox have been designed incrementally. The iterative research process has included three grounding processes: internal, external theoretical and empirical grounding (Goldkuhl and Cronholm, 2003). Internal grounding focuses on internal consistency of concepts, which are included in the meta-method and MC Sandbox. External theoretical grounding means turning to the body of knowledge that exists outside the knowledge developed in the research project. For this paper, external theoretical grounding is to do with how user-centred design can be used in method engineering to facilitate method-user-centred method configuration. Finally, empirical grounding involves application of the developed knowledge, which in our case focuses MC Sandbox. Since practitioners are participating in the design process, application is interpreted in a broad sense, involving analysis, design, implementation and test in the empirical environment.

Demarcating ourselves to the design of MC Sandbox, this part of the research project has been performed as a systems development project using an action research strategy. The development process was structured using a situational version of the Rational Unified Process (RUP). The chosen parts focused on requirements, analysis and design, implementation and test. In total, the development process included six iterations with clear milestones. The choice of using RUP as the systems development method during development of MC Sandbox was mainly based on the collaborating partners' preferences. Thus this was a trade-off for practice collaboration.

3 Theoretical Base

3.1 Why Method-User-Centred?

Methods exist for the purpose of supporting project members in systems and software development projects. These people are users of the method in the same sense that end-users are users of information systems created through systems development. In the latter case, the importance of harmonizing the mental models of systems designers with those of end users is often cited (Norman, 1988). These mental models are continuously shaped by experience and through interaction with the information system. However, expressing these mental models as requirements is difficult. When working with requirements in systems development, end-users have to express their requirements and their creativity has to be stimulated during that process (Maiden *et al.*, 2004). Malcolm (2001) states that user-centred approaches, as part of Rapid Application Development, is a rigorous approach to systems development and are appropriate when addressing tacit, semi-tacit and future systems knowledge. The later is by its nature incomplete and can therefore be subject to rapid change.

By means of analogous reasoning, we can similarly discuss mental models and tacit knowledge in the realm of method engineering. Stolterman (1991) discusses the

importance of creating an understanding of the method creator's mental model of the method. Stolterman and Russo (1997) use the terms public and private rationality, which, figuratively speaking, could be pictured as two spheres which are possible to move towards each other. Public rationality is inter-subjective understanding about prescribed actions and produced results, or why a specific part of a method is prescribed. This argumentative dimension of methods has been referred to as a method's rationale (Ågerfalk and Wistrand, 2003; Ågerfalk and Fitzgerald, 2005). Private rationality is expressed, according to Stolterman and Russo (1997), 'in the skills and in the professional ethical and aesthetic judgments' of a person. They argue that in order to make methods valuable the method creator has not only to influence public rationality but he or she has to change the fundamental thinking of the method user, i.e. the method user's private rationality. Otherwise method users do not find the necessary support during systems development. Consequently, it is important to involve method users early when crafting a situational method. Just as when involving end-users early in systems development, this involvement should focus on, to method users, concrete aspects.

3.2 Transferring User-Centred Ideas to Method Engineering

According to Cato (2001) it is possible to view user-centred design as a triad: the user, the use and the information. This triad focuses *who* is using the technology, *how* the technology is used and *what* is required to support that use. Translated into method engineering we thus focus on who the method users are as a team and these users' needs during a project; i.e. what kinds of challenges are found in the project. Part of that need is the situational method.

Storyboarding and prototyping are techniques frequently used in user-centred approaches to create a feel for the proposed solution (e.g. Carroll, 1994; Madsen and Aiken, 1993). The idea is to let use-scenarios and visualization drive the design, thus making it more tangible. Visualization often starts out as simple sketches and a distinction is often made between low-fi and high-fi prototypes. A paper-based storyboard often contains the structure, possible navigation through the parts of the information system, information provided by the system to assist the user, information provided by the user, and the result of the user's actions (Cato, 2001).

Nickols (1993) emphasizes that a prototype is a working model. As such, it does not need to be complete with regard to functionality. Low-fi and high-fi prototypes differ in the degree of technical implementation and the cost of change. Low-fi prototypes have a low degree of technical complexity and hence a low cost of change. On the other hand, they also have a low degree of functionality, which is extended when we move towards high-fi prototypes.

When moving these ideas to method engineering we see that visualizing the method design and its parts is essential. This means visualizing the situational method as a prototype during method configuration. Prototyping also involve a continuous evolution of the prototype and its design. Nauman and Jenkins (1982) present prototyping as a four step procedure: identify basic requirements, develop a working prototype, implement and use, and revise and enhance. The two latter activities are performed in an iterative pattern, which to some extent corresponds to evolutionary method engineering (Rossi et al., 2004) and scenario approaches (Rolland *et al.*, 1999). Consequently, it is not surprising that the implementation of evolutionary method engineering in MetaEdit+ shares several characteristics with high-fi prototyp-

ing. Methods are implemented incrementally as CASE tools, with a high degree of functionality. However, there is little attention given to the initial requirements and how to provide an initial prototype. The method users are involved when a situational method already exists and is subject for revision.

Consequently, evolutionary method engineering should be complemented with an approach where the method user is involved in the initial tailoring of the method. One hurdle in achieving such an involvement is the inherent complexity in method engineering. Method engineering tends to be a tedious process, especially if high-fi prototypes, such as runtime CASE tool implementations of methods, are factored into the equation.

Therefore we propose to use the basic idea of low-fi prototypes and storyboarding together with the method users in method configuration. We thus combine the idea of visualizing the situational method as a prototype and reducing the amount of detail. The latter shares similarities with illustrating navigation using a storyboard, and the map construction presented by Rolland *et al.* (1999). Furthermore, from storyboarding we borrow the idea of visualizing information provided by the method to assist the user, the information provided by the method user, and the results of the method user's action. The focus during method-user-centred method configuration is on what method parts add value to the development project and the project members as a team. Hence, we move away from the use of complex meta-languages when working together with the method users in the same way prototypes are preferred over complex diagrams in discussions with end users.

If we return to March and Smith's (1995) discussion about design science products we can conclude that these ideas have an affect on the design products found behind MC Sandbox which we intend to use; these design products are the concepts, models, and the meta-method that are implemented in MC Sandbox. Together these design products have to support the simplification of method modules to emulate a low-fi prototype and still provide the information needed for discussing method assistance and potential results of different choices.

3.3 Concepts in MC Sandbox

MC Sandbox incorporates three basic concepts for method configuration: the method component, the configuration package, and the configuration template. A method component is a self-contained part of a method that expresses the transformation of one or several artefacts into a defined target artefact together with the rationale of such a transformation. The concept is designed to provide black-boxing of the content of the method component. This means that the concept has two views: one internal view containing all the details and one external view providing a filtered version of these details (Karlsson and Wistrand, 2004). The filter is termed the method component's interface which contains the input and output artefact of the component, together with the component's overall goals. The latter thus returns to the idea of achieving rationality resonance. Together these parts of the method component provide a simplified view of the method's intended assistance and the intended results of prescribed actions it contains.

The basic idea behind method configuration in MC Sandbox is to use characteristics and the method rationale, which is expressed by method components, to make choices whether or not to include a method component in the current configuration. A characteristic is a delimited part of a development situation type, focusing on a certain

problem or aspect which the method configuration aims to solve or handle. Characteristics are used to narrow the focus to a delimited part of the development situation type, which is an abstraction of similar projects. Each characteristic addresses one or several method components and their purpose of existence in a situational method. In order to facilitate reuse, which improves the possibilities of prototyping situational methods, the concepts of configuration package and configuration templates are introduced and associated to the characteristic concept (Karlsson and Ågerfalk, 2004).

A configuration package is a method configuration of the base method suitable for one single value of a characteristic. Thus, a configuration package is a classification of method components based on overall goal relevance for a specific value of a characteristic. For example, a characteristic may be 'Difficult to elicit requirements?' and the value is the answer 'Yes.' Such a characteristic focuses the requirements part of the base method, and the intention is to select method components that ease communication of requirements between end users and project members, such as prototypes and storyboards.

Configuration packages contain demarcated parts of a situational method. However, real world situations are often complex settings that include multiple characteristics. Therefore, there is a need for a combination of characteristics, and consequently configuration packages, to capture many situations. A configuration template does all that. A configuration template is a combined method configuration, based on configuration packages, for a set of recurrent project characteristics. Hence, configuration templates are aggregates of configuration packages. These templates can be reused during method configuration as starting points for retrieving a closer match between the situational method needed in a project and the original base method – configuration templates are reusable pre-tailored versions of the base method (Karlsson and Ågerfalk, 2004). A situational method is a configuration template fine-tuned and adapted to a specific project, and used in that project. The configuration template selection is based on the project characterization, where the existing set of characteristics is reused as a base for questions to ask to project members.

Method configuration in MC Sandbox can have different starting points depending on the possibilities to find existing patterns to reuse. If a suitable configuration template can be found then the configuration process is rather straightforward – it is fine-tuned and used as a situational method. In situations where no matching configuration template can be found, a new one can be constructed based on existing configuration packages. Hence method configuration can involve the selection of configuration packages as well as complementing with new ones.

4 Method Configuration Using MC Sandbox

Method configuration using MC Sandbox is divided into five sub-processes, of which two are purely supportive: define method component and edit base method. In the remainder we will focus on the three core processes of method configuration: define a configuration package, define a configuration template, and define a situational method. We thus assume that method components exist in the MC Sandbox repository as does a base method defined based on these components. The reason for this choice is our focus on how to involve method users in an interactive process of tailoring the base method together with the method configurator.

4.1 Define Configuration Package

The starting point for defining a configuration package is the base method, and hence each configuration package inherits the structure of the base method. Thus, when we add a method component to the base method in MC Sandbox, it is added also to the configuration packages. Through this approach we have a starting point for a story-board-inspired discussion about the base method given a specific characteristic. Figure 1 illustrates the graphical user interface (GUI) of MC Sandbox when working with configuration packages. The screen is divided horizontally, where the lower section contains the method modelling area (labelled modelling view), and the upper section contains the status of selected method components (labelled method component status). The right part of the upper section contains the complete set of existing configuration packages based on the base method in use. The tree structure is sorted by the characteristics that the configuration packages belong to.

The modelling view makes use of the two views of method components. The external view is used when modelling a configuration package together with method users. A method component is depicted as a rectangle in the modelling view. Arrows connecting method components show the flow of artefacts between the components, which can be one-way or two-way. For example, in Figure 1 the result from the Current State Infrastructure component is recommended as input to the Vision/Scope component. Different colours are used to illustrate the classification of method components (in the current version of MC Sandbox four classifications exist: perform *as is*, *omit* from the configuration, *exchanges* an existing component, *not applicable* for this characteristic). Hence, the base method is illustrated as a low-fi prototype, useful for discussing effects of suppressing different components as well as introducing new ones, with a minimum of details.

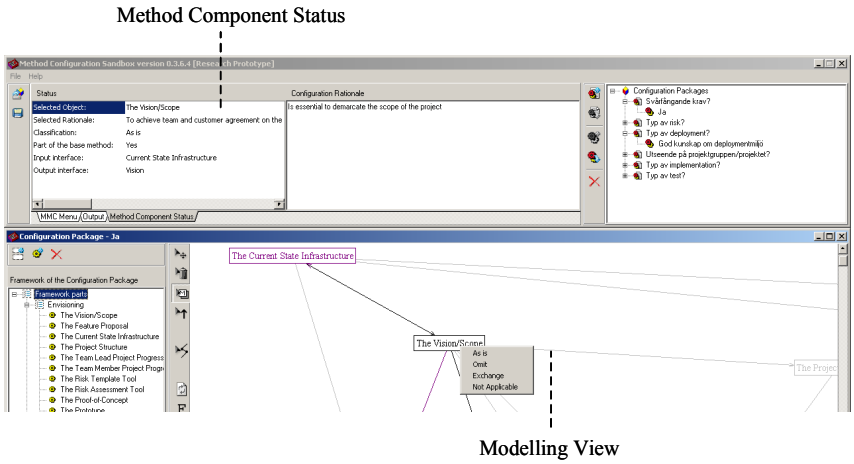

Figure 1. Defining a Configuration Package.

When a method component is selected in the modelling view, the content of that method component's interface is presented in the upper right section of the GUI. For example, the Vision/Scope component has been selected in Figure 1 and hence its interface is presented. The interface contains information about recommended input,

the component's deliverable (i.e. its output artefacts), the component's rationale and the current classification (*as is* means that it is to be performed as prescribed by the base method).

Modelling the configuration package begins with specifying the demarcation of the configuration package, that is, defining what parts of the base method is of interest when discussing this specific characteristic. When creating a new configuration package, all of the base method's components are classified *not applicable*. This means they are considered to be outside the characteristic's scope. Subsequently, relevant method components are brought into scope based on the method users' opinions. This is an iterative process where each method component is discussed based on their method rationale, as expressed in the method components' interface. Classification of existing method components is done based on the method users' needs given the constraints of the characteristics. The configuration rationale is documented in MC Sandbox using the upper middle part of the GUI, which means we can always examine the arguments leading to a specific classification.

4.2 Define Configuration Template

In Section 3.3 we defined a configuration template as an aggregate of configuration packages. It follows that the selection of relevant configuration packages and their integration are central parts when defining a configuration template. The GUI used for this shares basic structure with the GUI for defining a configuration package. This means that the screen is divided into an upper and lower part. The upper part contains information about selected items. Figure 2 illustrates the lower section, which is vertically divided. The left part contains the functionality to work with the development situation type profile. The right-hand section of the user interface is devoted to the actual modeling of the configuration template's content.

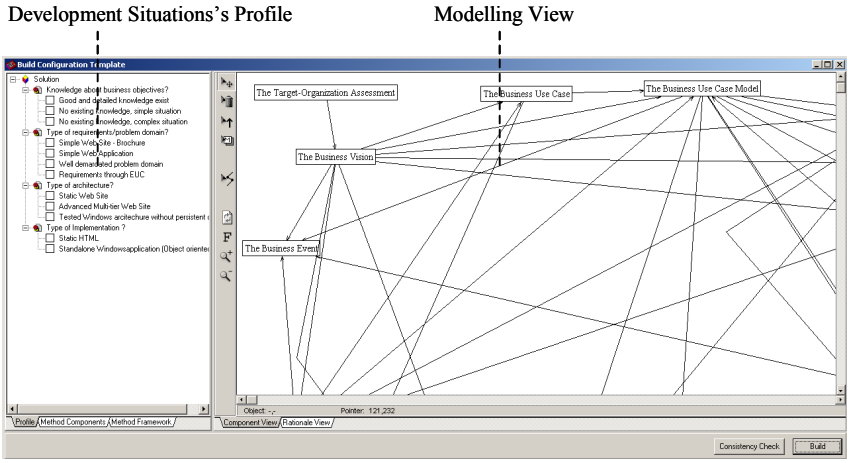

Figure 2. Defining Configuration Template.

A development situation type is characterized by means of the configuration packages that are selected for the existing range of characteristics. In Figure 2 we see examples of four characteristics and their affiliated configuration packages. Each characteristic

is formulated as a question, and the configuration packages act as possible answers. The method configurator uses the checkboxes to make choices based on the ongoing discussion with the method users. Only one configuration package is possible, but not necessary, to choose per characteristic. Consequently, if a characteristic is found to be irrelevant, it can be left out when selecting configuration packages.
When the relevant selection has been made, the method configurator clicks on the build button found in the bottom right corner of the GUI. This causes MC Sandbox to build a configuration template based on the selected configuration packages. If conflicts arise during this build, the tool lists them together with a reason. The method configurator and the method users have to solve these conflicts in each specific case and give priority to one of the configuration packages that are involved in the conflict. Yet again, the method rationale of the method components is matched with the needs of the method users. When the conflict has been resolved, the configuration template can be rebuilt.

The result of the building process is a prototype of a situational method, presented in the same low-fi fashion as used when working with configuration packages. If the method users are not satisfied, modifications can be made either through selection of a different set of configuration packages and rebuilding the template, or through manual reclassifications of method components.

4.3 Define a Situational Method

As stated above, each situational method is based on a configuration template. The range of characteristics is used once again, in this case to create a project profile. The method configurator and the method users use the checkboxes shown in Figure 3 to indicate these choices.

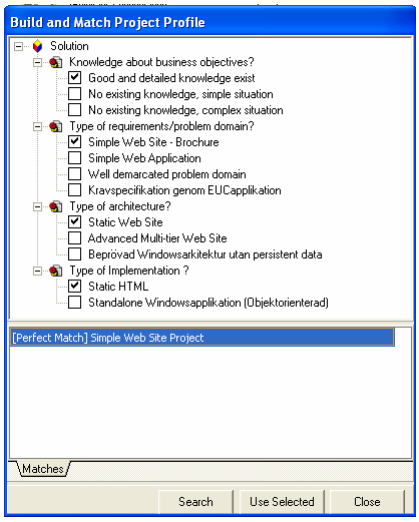

Figure 3. Create Project Profile and Search a Configuration template.

One configuration package can be selected for each characteristic, if the characteristic is found to be relevant. A project profile contains the criteria for searching in the repository for matching development situation types and corresponding configuration

templates. Hence, this approach does not rely on a complex query language, as for example Method Engineering Language (Harmsen, 1997) or SGMLLQL (Ralyté, 1999), for administration and retrieval of reusable method assets. The decreased complexity improves possibilities to perform method configuration interactively together with the method users.

The search result is a list of possible candidate configuration templates. The method configurator and the method users can choose from the list of configuration templates and the MC Sandbox then generates a situational method. The situational method is presented in the same type of view as used for presenting a configuration package or template.

Adjustments of the situational method are possible since the configuration tool is available at the user interface. Consequently, it is possible to reclassify method components using the classification schema mentioned above. Typically, as the project progresses and the understanding of the method requirements evolve, both characteristics and corresponding configuration packages evolve accordingly.

5 Empirical Experience

The MC Sandbox functionality addressed in this paper has been tried in a small-sized systems development organization and in controlled experiment situations. The tool has been used both during reconstruction of existing situational methods and in building new ones. The former has been a starting point for building the repository content based on existing projects. However, at the same time, adjustments have been made based on post-project experiences.

Sub-sets of the project teams have been involved during both construction and reconstruction of reusable patterns (configuration packages and templates). The content of the repositories has been used as the starting point for discussions about specific configurations. For example, we have discussed the options of reusing existing configuration templates based on the profile of the project. The characteristics formulated as questions have been a good starting point. They have helped the method users to formulate characterization of their projects and missing configuration packages. One of the project managers' comments about a configuration package illustrates this aspect 'this configuration package looks similar to the one I have in mind, except for …' Consequently, earlier configuration work was reused and made concrete through visualisation of parts of a situational method.

Furthermore, the use of characteristics made the discussion with method users focused, since a smaller part of the base method was addressed. The implementation of configuration packages in MC Sandbox made it possible to quickly sketch a prototype of a configuration, which could later be returned to for further elaboration. During the initial part of the configuration workshop discussions, several characteristics and configuration packages where drafted. They were documented as brief ideas and were later elaborated in more detail using an iterative pattern, similar to the ideas of refining a prototype. For example, such decisions involved moving the demarcation for a configuration package, which also affected related configuration packages. The method users often drove these decisions, based on their experience and needs using the low-fi method prototype as a starting point for analysis.

During the method configuration workshops the external view of method components was used to discuss implications of different choices of method components.

The storyboard inspired design made it easy to visualize the effects of different classifications. The project manager of one of the projects stated, 'it [MC Sandbox] is a good idea, it is an easy way to elicit steps to perform.' For example, the method users discussed the effect of suppressing the usage scenarios in Microsoft Solution Framework when working with requirements engineering. In cases where the requirements are difficult to elicit, the method users preferred the prototype component. The main reason for this decision is found in the rationale behind a prototype, which was interpreted as 'To express the features needed and their design.' The main difference compared to a usage scenario is the possibility to express the design in a more concrete way.

During this discussion it was also possible to directly address the impact on later parts of the method support. This was made possible through the low-fi prototype view of the base method and visualization of relationships between method components. Subsequently, the method users were, as a team, made aware of the fact that this could give weaker input when writing test cases. The team aspect is important, since it improves the different members' understanding of why a method component has to be included. This is a major advantage of using a method-user-centred approach to method configuration. Since major sets of actor roles were represented during the method configuration workshops it was easy to receive response on collaboration issues. This provides for the method configurator to work with a multi-role perspective during method configuration and continuously have an ongoing discussion about decisions and consequences.

The line of reasoning about specific classifications was included in the configuration rationale, using the configuration rationale textbox. These arguments were later reused when combining different configuration packages into configuration templates. Thus, it is possible to build awareness of how certain classifications could affect other parts of the base method, which are not included in the configuration package.

The GUI design used for modelling configuration packages, templates and situational methods has received different opinions depending on the complexity of the base method. When the complexity of the base method increases, these views tend to become cluttered, since the number of relationships between method components increases. This has been reported on by Karlsson and Wistrand (2004).

6 Conclusion

In this paper we have addressed a basic assumption of many method engineering approaches: that it is possible for project members to explicitly specify requirements on a situational method. In order to do so we have turned to user-centred design and specifically prototyping and storyboarding. We have shown that appropriate tool support provides the means to achieve method-user-centred method configuration. Given appropriate high-level modelling concepts, such as the Method for Method Configuration concepts of method components, configuration packages and configuration templates, it is possible for a method configurator to work interactively with method users to construct a suitable situational method by means of collaboratively elicit, negotiate and commit to method requirements. This way, the method users' and the method configurator's understanding of the current development practice develops incrementally and rationality resonance is achieved at reasonable cost.

Acknowledgements

This work has been financially supported by the Knowledge Foundation (KK-Foundation) through the programme for the promotion of research in IT at new universities and university colleges in Sweden (IT-Lyftet), and the Science Foundation Ireland Investigator Programme, B4-STEP (Building a Bi-Directional Bridge Between Software ThEory and Practice).

References

ÅGERFALK PJ and FITZGERALD B (2005) Methods as Action Knowledge: Exploring the Concept of Method Rationale in Method Construction, Tailoring and Use. In *Proceedings of EMMSAD'05: Tenth IFIP WG8.1 International Workshop on Exploring Modeling Methods in Systems Analysis and Design* (HALPIN T, KROGSTIE J and SIAU K, Eds), p 413–426, Porto, Portugal.

ÅGERFALK PJ and WISTRAND K (2003) Systems Development Method Rationale: A Conceptual Framework for Analysis, In *Proceedings of the 5th International Conference on Enterprise Information Systems (ICEIS 2003)*, Angers, France.

BRINKKEMPER S, SAEKI M and HARMSEN F (1999) Meta-Modelling Based Assembly Techniques for Situational Method Engineering, *Information Systems*, 24(3) 209–228.

CARROLL JM (1994) Making Use a Design Representation, *Communications of the ACM*, 37(12) 29–35.

CATO J (2001) *User-Centred Web Design*, Addison Wesley, Harlow, England.

FITZGERALD B, RUSSO NL and O'KANE T (2003) Software Development Method Tailoring at Motorola, *Communications of the ACM*, 46(4) 65–70.

GOLDKUHL G and CRONHOLM S (2003) Multi-Grounded Theory: Adding Theoretical Grounding to Grounded Theory, In *European Conference on Research Methods in Business and Management (ECRM 2003)*, Reading, UK.

HARMSEN AF (1997) *Situational Method Engineering*, Doctoral Dissertation, Moret Ernst & Young Management Consultants, Utrecht, The Netherlands.

HEVNER AR, MARCH ST, PARK J and RAM S (2004) Design Science in Information Systems Research, *MIS Quarterly*, 28(1) 75-105.

KARLSSON F and WISTRAND K (2004) MC Sandbox: Tool Support for Method Configuration, In *Proceedings of Ninth CAiSE/IFIP8.1/EUNO International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design (EMMSAD'04)*, Riga, Latvia.

KARLSSON F and ÅGERFALK PJ (2004) Method Configuration: Adapting to Situational Characteristics While Creating Reusable Assets, *Information and Software Technology*, 46(9) 619-633.

MADSEN KH and AIKEN PH (1993) Experiences Using Cooperative Interactive Storyboard Prototyping, *Communications of the ACM*, 36(4).

MAIDEN NAM, GIZIKIS A and ROBERTSON S (2004) Provoking Creativity: Imagine What Your Requirements Could Be Like, *IEEE Software*, 21(5).

MALCOLM E (2001) Requirements Acquisition for Rapid Applications Development, *Information & Management*, 39101-107.

MARCH ST and SMITH GF (1995) Design and Natural Science Research on Information Technology, *Decision Support Systems* 15, pp. 251-266.

MATHIASSEN L (2002) Collaborative Practice Research, *Information Technology & People*, 15(4) 321-345.

MCKAY J and MARSHALL P (2001) The Dual Imperatives of Action Research, *Information Technology & People*, 14(1) 46–59.

NAUMANN JD and JENKINS AM (1982) Prototyping: The New Paradigm for Systems Development, *MIS Quarterly*, 6(3) 29-44.

NICKOLS FW (1993) Prototyping: Systems Development in Record Time, *Journal of Systems Management*, 44(9).

NORMAN DA (1988) *The Psychology of Everyday Things*, Basic Books, New York.

RALYTÉ J (1999) Reusing Scenario Based Approaches in Requirement Engineering Methods: Crews Method Base, In *The First International Workshop on the Requirements Engineering Process*, Florence, Italy.

ROLLAND C, PLIHON V and RALYTÉ J (1998) Specifying the Reuse Context of Scenario Method Chunks, In *Advanced Information Systems Engineering, 10th International Conference CAiSE'98*, Pisa, Italy, June 8-12, (Eds, Pernici B and Thanos C).

ROLLAND C, PRAKASH N and BENJAMEN A (1999) A Multi-Model View of Process Modelling, *Requirements Engineering*, 4(4) 169–187.

ROSSI M, RAMESH B, LYYTINEN K and TOLVANEN J-P (2004) Managing Evolutionary Method Engineering by Method Rationale, *Journal of the Association of Information Systems*, 5(9) 356-391.

STOLTERMAN E (1991) *Designarbetets Dolda Rationalitet: En Studie Av Metodik Och Praktik Inom Systemutveckling*, Doctoral Dissertation in Swedish, Department of Informatics, Umeå University, Umeå, Sweden.

STOLTERMAN E and RUSSO NL (1997) The Paradox of Information Systems Methods -- Public and Private Rationality, In *The British Computer Society 5th Annual Conference on Methodologies*, Lancaster, England.

WALLS JG, WIDMEYER GR and EL SAWY OA (1992) Building Information System Design Theory for Vigilant Eis, *Information Systems Research*, 3(1).