**European Journal of Information Systems (2009) 18,** 300–316

www.palgrave-journals.com/ejis/

# Exploring agile values in method configuration

Fredrik Karlsson[1] and
Pär Ågerfalk[2]

[1]*MELAB, Swedish Business School at Örebro University, Sweden;* [2]*Department of Information Science, Uppsala University, Sweden*

**Correspondence: Fredrik Karlsson, MELAB, Swedish Business School at Örebro University, SE-701 82 Örebro, Sweden.**
**Tel: +46 19 30 39 94;**
**Fax: +46 19 33 25 46;**
**E-mail: fredrik.karlsson@oru.se**

**Abstract**
The Method for Method Configuration (MMC) has been proposed as a method engineering approach to tailoring information systems development methods. This meta-method has been used on a variety of methods, but none of these studies have focused on the ability to manage method tailoring with the intention to promote specific values and goals, such as agile ones. This paper explores how MMC has been used during three software development projects to manage method tailoring with the intention to promote agile goals and values. Through content examples of method configurations we have shown that it is possible to use MMC and its conceptual framework on eXtreme Programming and we report on lessons learned with regard to maintaining coherency with the overall goals of the original method.
*European Journal of Information Systems* (2009) **18,** 300–316.
doi:10.1057/ejis.2009.20; published online 21 July 2009

**Keywords:** method configuration; method tailoring; method engineering; extreme programming; agile method

## Introduction

As business requirements are changing at an ever-increasing speed, the need for more flexible development processes has made it to the top of the agenda for both researchers and practitioners. Agile methods with their emphasis on 'just enough method' are exemplars of a solution to this problem. However, it is a well-known fact that there is no one-size-fits-all method to software and information systems development (Henderson-Sellers & Serour, 2005; Fitzgerald *et al*., 2006; Ågerfalk & Fitzgerald, 2006b). Earlier studies (Russo *et al*., 1995; Tolvanen, 1998; Fitzgerald *et al*., 2003; Serour & Henderson-Sellers, 2004) have shown that as all projects are unique they require flexible method support (Brinkkemper *et al*., 1994; Van Slooten & Hodes, 1996).

The quest for flexible and situated processes has been on the research agenda for quite some time (Basili & Rombach, 1987; Kumar & Welke, 1992; Cameron, 2002; Fitzgerald *et al*., 2003). Existing inventories show a wide range of existing approaches and computerized environments for method tailoring (Odell, 1996; Ralyté *et al*., 2003; Niknafs & Ramsin, 2008; Sunyaev *et al*., 2008). These approaches range from selection of methods to tailoring (e.g. Cameron, 2002; Bajec *et al*., 2006) and construction of project-specific (situational) methods (Brinkkemper, 1996; Rolland & Prakash, 1996), and are supported by computerized environments to different degrees (e.g. Plihon, 1996; Harmsen, 1997; Rossi *et al*., 2004). Although agile methods seem adequately to address flexibility in the development process, how best to adapt those methods to particular development situations is still not well understood (Aydin *et al*., 2005; Fitzgerald *et al*., 2006) and there are not many studies on the tailoring of agile methods, Paige & Brooke (2005); Serour & Henderson-Sellers (2004) and Qumer & Henderson-Sellers (2006, 2008) being notable exceptions.

Ironically, the suggestion by early agile method proponents that agile methods must be applied in their entirety to achieve a synergistic

**Received: 6 May 2008**
**Revised: 30 September 2008**
**2nd Revision: 16 June 2009**
**Accepted: 17 June 2009**

combination of individual practices is somewhat at odds with the spirit of flexibility (Ågerfalk & Fitzgerald, 2006b). It is also not borne out in recent research, which, on the contrary, suggests that also agile methods need to be tailored to the particular development context (Fitzgerald *et al.*, 2006). Indeed, the ability to adapt the process to current circumstances is one of the principles of the agile manifesto (see below).

When tailoring a method you want to achieve a method that fits the situation and at the same time aligns with the basic goals and values of the method. Otherwise, the core philosophy of the method is lost. In the case of agile methods you want to maintain agility by adhering to agile values and goals. Method Configuration (Karlsson & Ågerfalk, 2004, 2009) has been proposed as an approach where one method, termed base method, is taken as the starting point for configuring a situational method suitable to the project at hand. This is particularly useful when an organization wants to establish an organization-wide method to be used across all projects, thus sharing a common structure. Incentives for such an effort include more effective communication, reduced training costs due to common modeling languages and the utilization of industry standards, as well as the access to existing computerized tools. The Method for Method Configuration (MMC) and its computerized tool support MC Sandbox was devised as a way to carry out method configuration in order to strike a balance between the structure of organization-wide methods and the flexibility required of project-specific methods (Karlsson & Ågerfalk, 2009). This approach is anchored in Activity Theory (Korpela *et al.*, 2004; Karlsson & Wistrand, 2006) and the concept of method rationale (Rossi *et al.*, 2004; Ågerfalk & Fitzgerald, 2006a). The idea is to emphasize the collaborative aspect of methods and to focus goals and values during method tailoring. The use of goals can be found also in other approaches to method tailoring (Nilsson, 1999; Rossi *et al.*, 2000, 2004; Ralyté *et al.*, 2003; Gonzalez-Perez *et al.*, 2009). However, most of these approaches have a method integration focus. Furthermore, when it comes to goal-driven method tailoring approaches, no analysis has been made of the ability to manage method tailoring with the intention to promote specific values and goals, such as agile ones.

The aim of this paper is to explore the extent to which MMC can support method tailoring with the intention to preserve or even emphasize agile goals and values of the base method. Generally, method engineering covers both the development process and the modeling languages and constructs used in that process (e.g. Kumar & Welke, 1992; Brinkkemper, 1996). The focus of this paper is on the development process and the products (deliverables) that are results of and inputs to process activities. Consequently, modeling languages *per se* are not addressed.

The paper is structured as follows: the next section introduces a selection of key MMC concepts along with MC Sandbox. The following section presents the research approach adopted. The following section analyses agile method values. With this basis, the next section reports on empirical experience from the use of MMC during three agile system development projects. Finally, the last section concludes the paper and points at some future research.

## The MMC key concepts and values

The aim of the MMC is to support method configuration: the planned and systematic adaptation of a specific method through the use of reusable assets. The meta-method embraces the need for both structure and flexibility in systems development practice and is anchored in the following design principles (Karlsson & Ågerfalk, 2005, 2009): (1) the principle of modularization: (1a) self-contained modules, (1b) internally consistent and coherent modules, (1c) support for information hiding; (2) the principle of method rationale for selecting method parts: (2a) support for analysis of potential to achieve rationality resonance (2b) support method-in-action decisions; and (3) the principle of a multi-layered reuse model. An additional design principle is found for MC Sandbox: (4) involve method users in an interactive process of tailoring the base method, together with the method engineer. Design principle (4) is a modification of the original view of method engineering proposed by Kumar & Welke (1992) where the method engineer creates a situational method and the method users are passive information providers.

These design principles are implemented as three core concepts that provide the possibility to work with reusable method assets: method components (Karlsson & Wistrand, 2006), configuration packages (Karlsson & Ågerfalk, 2009), and configuration templates (Karlsson & Ågerfalk, 2009). The presentation below focuses on these three key concepts in MMC/MC Sandbox. For a more comprehensive treatment of all concepts and the computerized tool, see Karlsson & Ågerfalk (2005, 2009), Wistrand & Karlsson (2004) and Karlsson & Wistrand (2006).

On first inspection MMC may not intuitively be seen as particularly agile. However, the basic tenet is that by spending some time on structuring the tailoring process early in a project, flexibility increases when the process is actually executed. In principle, flexibility requires structure at the process tailoring (or meta-method) stage (Karlsson & Ågerfalk, 2009).

## The method component concept

A modularization concept is needed to enable systematic ways of working with method configuration. Several such concepts exist in the method engineering literature where method components (Karlsson & Wistrand, 2006), method chunks (Rolland & Prakash, 1996) and method fragments (Brinkkemper, 1996) are perhaps the most prominent ones. Through such concepts, specific parts can be suppressed, added or exchanged within the confines of a coherent method. Furthermore, through the use of a modularization concept it is possible to

achieve information hiding during method configuration and thus avoiding irrelevant details. MMC is based on the method component concept – a self-contained part of a method expressing the transformation of one or several artifacts into a defined target artifact, and the rationale for such a transformation. This concept aligns with the method chunk construct and can be viewed as an aggregate of method fragments. Hence, just like in the method chunk concept we find a 'hard-wired' (Henderson-Sellers *et al*., 2008) connection between process and product parts of a method. However, unlike method chunks and method fragments, a method component is non-hierarchic. This means that method components exist on a 'fixed' level of granularity. Since a method component is defined based on the artifacts (or deliverables) of the base method, the types of identifiable artifacts determine its granularity.

A method component has two parts: its *content* and its *interface*. The content describes the input, the output, the transformation process, and the method rationale (see below). The interface presents a selection of those parts, to facilitate information hiding.

### Method component content

The content of a method component is an aggregate of method elements: A *method element* is a part of a method that manifests a method component's target state or facilitates the transformation from one defined state to another. Method elements are constituted by prescribed actions (e.g. Detail a user story), concepts (e.g. Task), notations (e.g. textual), artifacts (e.g. User Story) and actor roles (e.g. Customer). A development activity is essentially a set of prescribed actions with associated sequence restrictions that guide project members' actions in specific situations. In performing these actions, developers' attention is directed towards specific phenomena in the problem domain. These are concepts that express an understanding of the problem domain, and also of the method itself. Results of actions are documented by artifacts using a prescribed notation (a modeling language), giving the concepts a concrete representation. Artifacts are thus both deliverables from and input to (subsequent stages of) the process. In MMC, methods are viewed as heuristic procedures and hence specified inputs are only recommended inputs. Finally, actor roles describe the functions that actors play in the method component. The selection of actor roles is determined by the prescribed actions that are needed for the transformation process.

The rationale part of the method component consists of two concepts: goals and values. Each method element is included in the method component for reasons, which are made explicit by associating method elements to goals. These goals are anchored in values. Together the goals and values reflect the underlying perspective of the method from which the method component originates. When working with MMC, method rationale is more important than the deliverable as such. Through the method rationale it is possible to address the goals that are essential in order to fulfil the overall goal of a specific project. Prescribed actions and artifacts are viewed as a means to achieve those goals. Hence, method rationale can help developers not to lose sight of the ultimate result, and also help them to find alternative ways forward.

Figure 1 illustrates the internal view of the User Story component as it is presented in MC Sandbox, The graphical user interface is divided into three sections. The leftmost section contains a tree view of all included method elements. Here we find the User Story artifact. The middle part shows the rationale of the method component, based on the goals of individual method elements. Finally, the rightmost section provides links to external sources that contain method element content. For example, to a User Story template. These external sources are used in the situational method depending on the classification of the method components. As MMC and MC Sandbox focus on modification of the development processes on the deliverable level, no changes to the modeling languages or the tools themselves are required in the process.

### The method component interface

The purpose of the interface is to hide unnecessary details during method configuration. It draws on the component construct as used in software engineering and the fact that the primary interest during method configuration is the results offered and the required inputs, not on how a task is executed. This reduction of complexity is achieved through the *method component interface*: a reference to a selection of method elements and rationale that is relevant to the task at hand. The interface represents an external view of a method component. During method configuration, the method component's overall goals and the artifacts are the primary focus. The artifacts are designated as input and/or deliverable (output), as discussed above. This is necessary in order to deal with the four fundamental actions that can be performed on an artifact (Gonzalez-Perez *et al*., 2009): create, read, update and delete. An artifact is classified as a deliverable when it is only created by the method component. If the artifact can be updated by the same method component it is classified as input as well. Per definition, a component can take one or several input artifacts, but has only one deliverable. Finally, the interface expresses the method component's overall goals, representing the method rationale. These goals are used to discuss possible rationality resonance in a project with certain characteristics – the extent to which different methods' and system developers' rationality overlap (Stolterman & Russo, 1997; Ågerfalk & Fitzgerald, 2006a).

## The configuration package

Method configuration is about deciding whether or not method components in a base method are to be

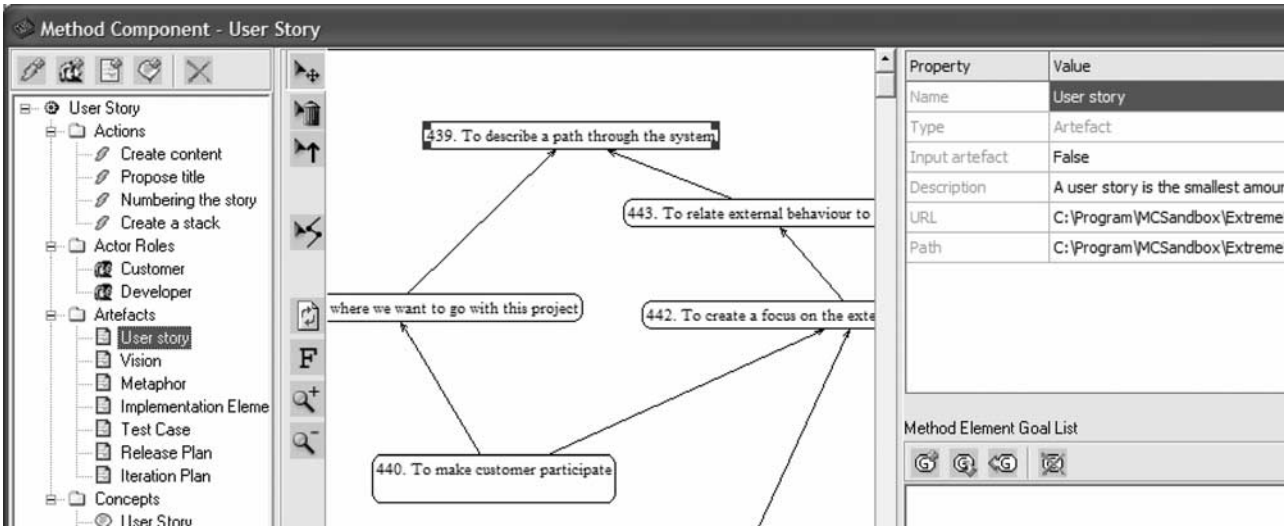


**Figure 1** The internal view of the user story method component.

performed, and to what extent. In MMC this is done through the use of method rationale, which is expressed in the method components' interfaces as discussed above. The result of this selection, with respect to particular project characteristics, is represented in a configuration package. This selection of method components can, if required, include components from complementing methods. A characteristic is viewed as a question about one isolated aspect of the development situation. Such a question typically has several possible answers that constitute the characteristic's dimension; one possible answer is called a *configuration package*. A configuration package is thus a configuration of the base method suitable for a characteristic's value. Hence, each characteristic addresses one or several method components and their purpose. That is, a configuration package is a classification of method components with regard to how relevant their overall goals are for a specific answer in a characteristic's dimension.

Figure 2 shows the graphical user interface in MC Sandbox when working with configuration packages. The screen is divided into three main sections: the lower part of the screen contains the method modeling area, and the upper left part contains the interface and classification status of a selected method component. For example, in Figure 2, the interface of the Vision component is presented. In this case the interface contains the overall goal (selected rationale in MC Sandbox), the recommended input (in this case none) and the output (the Vision artifact). The upper right part of the graphical user interface 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 method modeling area makes use of the external view of method components. A method component is presented as a rectangle. Arrows connecting method components show the flow of artifacts between the components, which can be one-way or two-way. For example, in Figure 2 the result from the Vision component is recommended as input to the User Story component.

The classification of method components is based on a two-dimensional classification schema, as shown in Table 1. The vertical dimension focuses on how much attention the developers should devote to a particular method component: 'None,' 'Insignificant,' 'Normal' or 'Significant'. If at this stage a method component is found to be unimportant, it can be classified as 'Omit' outright. The three aspects of the horizontal dimension – 'Satisfactory,' 'Unsatisfactory' and 'Missing' – cut across the vertical dimension. This dimension is referred to as the potential for achieving rationality resonance between the base method's content and the system developers' intensions. Together, the rationality resonance dimension and the attention dimension provide different variants of the fundamental method configuration scenarios that need to be supported: selection, exchange and addition. Different colors are used in MC Sandbox to illustrate these classifications.

## The configuration template

Although configuration packages and characteristics are used to simplify analyses of the base method, there is also a need to handle more complicated situations where characteristics exist in combinations. This is the purpose of the *configuration template*. A configuration template is a combined method configuration that covers the complete base method. It is based on one or more configuration packages, for a set of recurrent project characteristics. The concept allows for the reuse of combined configuration packages that target development situation types common within the organization. In MMC, the selection is based on the set of available characteristics and configuration packages. One configuration package

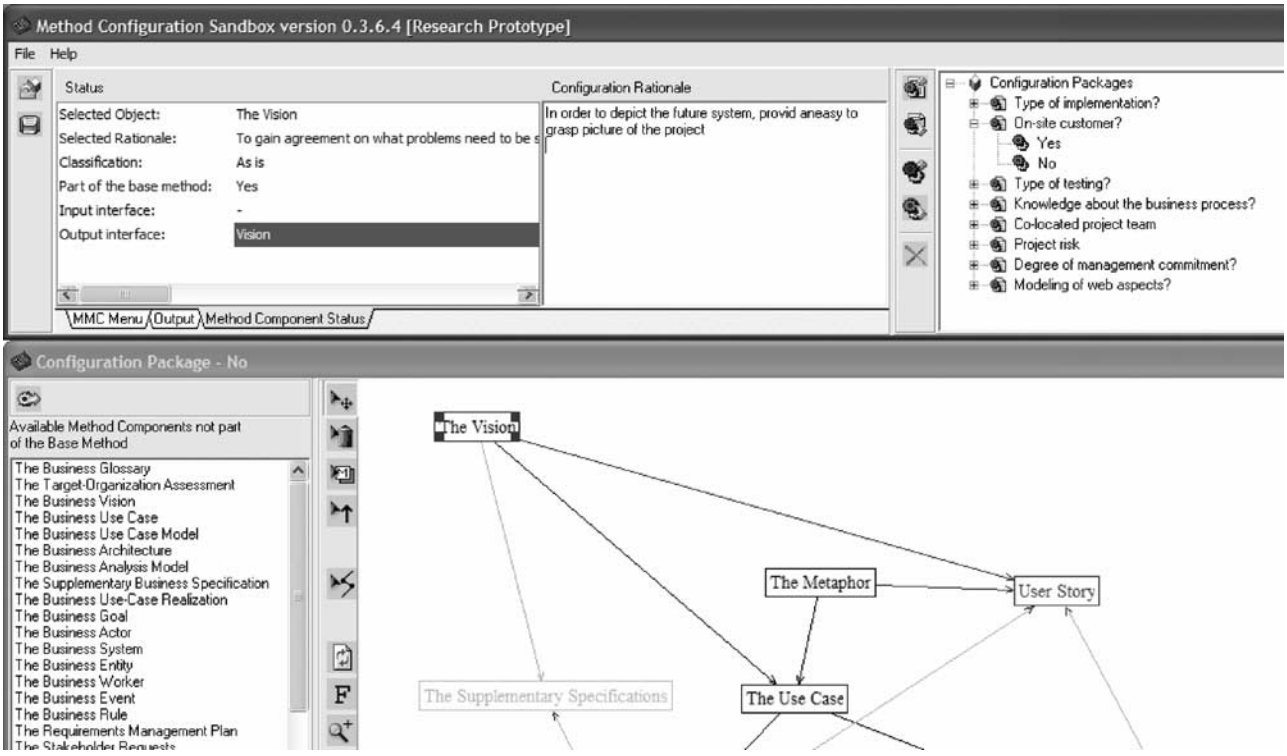


**Figure 2** Defining a configuration package.

**Table 1 Classification schema for method components**

| *Attention given to method component* | | *Potential to achieve rationality resonance* | | |
|---|---|---|---|---|
| | | *Satisfactory* | *Unsatisfactory* | *Missing* |
| | None | Omit | — | — |
| | Insignificant | Perform informal | Exchanges informal | Added informal |
| | Normal | Perform as is | Exchanges as is | Added as is |
| | Significant | Emphasize as is | Exchanges emphasized | Added emphasized |

can be chosen per characteristic and, if a characteristic is irrelevant, it can be left out when selecting configuration packages. The graphical user interface used for defining configuration templates shares basic structure with the one used for defining configuration packages. This means that the screen is divided into an upper and a lower part. The upper part contains information about selected method components. Figure 3 illustrates the lower section, which is divided vertically. The left part contains the functionality to select configuration packages, while the right part shows the method modeling area with the current version of the configuration template. This area shares the design with the method modeling area for configuration packages using the external view of method components. When the relevant selection of configuration packages has been made MC Sandbox semi-automatically builds a configuration template based on these selections. If conflicts arise between overlapping configuration packages, they are listed together with the reasons for conflict. Such conflicts have to be manually resolved.

For reasons of efficiency, relevant configuration templates can be retrieved with a search engine based on a selection of characteristics and configuration packages. The situational method is based on a selected configuration template and is the method delivered to the project team for use. When this method is enacted in a project, experiences should be fed back to the configuration process in order to improve configuration templates and/or configuration packages and to facilitate knowledge sharing between projects. This is typically done continuously throughout the project.

## Research approach

Considering the purpose of the paper, exploring agile method tailoring with MMC in a real world setting, this

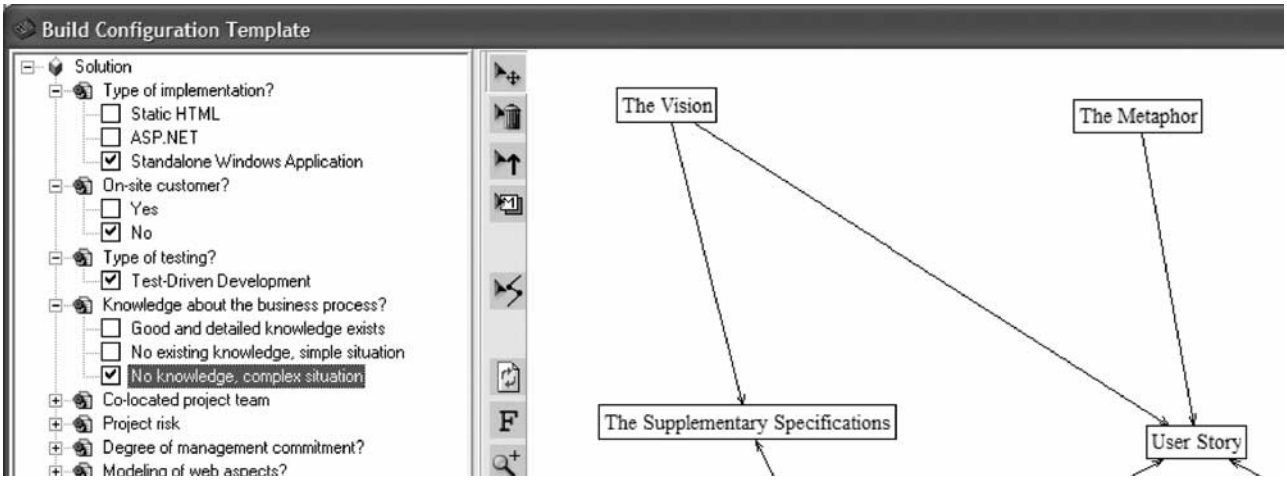


**Figure 3** Defining a configuration template.

**Table 2 Characteristics of projects**

| *Project* | *Type of information system* | *Person-hrs* | *Calendar months* | *No of developers* |
|---|---|---|---|---|
| 1 | Web-based inventory system | 1500 | 4 | 5 |
| 2 | Web-based time report system | 800 | 2 | 4 |
| 3 | Web-based quotation system | 1100 | 3 | 5 |

study can be classified as a case study (Lee, 1989; Yin, 1994). The case unfolds the method configuration process in a small software development company in Sweden that builds industrial software for the global market. They are to a large extent a virtual organization with a philosophy to work with system development projects on a networked basis. This means that only certain critical competence exists in-house and that project teams are formed through external contacts. The company agreed to evaluate MMC through the use of MC Sandbox in three commercial system development projects.

The choice of industrial partner was based on the following premises: they had to agree to provide access to systems development projects, they had to use an agile method, and they had agree to set aside resources to introduce MMC. The choice of eXtreme Programming (XP) as the base method was the industrial partner's. They do not use it as a true organization-wide method, but only with some specific clients. This repeated use, however, is similar to the use of an organization-wide method.

The first author of this paper assumed the project manager role in all three projects. However, he was not responsible for the method configuration, except for mentoring one of the system developers when introducing MMC and when building an initial version of the base method (XP) in MC Sandbox. The latter was necessary to enable the method configuration using MMC. Method engineering was distributed among the developers and the majority of the method configuration decisions were taken together during method-user-centred method configuration (Karlsson & Ågerfalk, 2005) workshops. In order to facilitate such workshops, each team received an introduction to MMC and MC Sandbox – approximately 3 h. The mentored system developer gave this introduction. Some of the developers participated in more than one of the projects, which facilitated knowledge sharing. The selection of projects was based on the industrial partner's current project portfolio. A summary of the projects' characteristics is shown in Table 2.

## Data collection

This research is based on several data sources from the three systems development projects: configurations stored in MC Sandbox, log books from the method configuration workshops and interviews with the developers. The method configuration workshops were carried out along the lines of MMC and MC Sandbox. This means that all project members were involved in the tailoring work where configuration packages and template were either created or reused. The starting point for the configuration work was XP implemented as a base method in MC Sandbox (see Figure 2) with some additional method components, mainly taken from the Rational Unified Process (Kruchten, 2004). In addition, the organization could gradually add configuration packages and templates. During the configuration work, the project members used the external view of method

components to discuss implications of different choices concerning method components. The internal view was only used in cases were the method component content was unknown to them. In the spirit of Joint Application Development (Wood & Silver, 1995) the objective was to strive for consensus decisions and to increase the understanding of each other's method needs. Despite that one project member was in charge of documenting the configuration decisions in MC Sandbox, the driver role shifted during the workshops depending on what method parts were in focus. MC Sandbox provides the possibility to store information about configuration decisions made by the system developers along with comments that explain why the decisions were made. Information about decisions is stored in a structured format while the comments are stored as free text. The latter made it possible to store, for example, citations from the configuration workshops.

The first author wrote the logbooks during the configuration workshops. The log books contained observations about the process such as, configuration decisions and arguments. Hence, when it came to configuration decisions and arguments they were possible to compare with the data in MC Sandbox, to reduce researcher bias. However, when documenting the configuration process such as, data about problematic areas and viewpoints on these areas was captured – data which are not found in MC Sandbox.

The interviews were semi-structured using the logbooks and data from MC Sandbox as a form of interview guide (Patton, 1990). The interviews focused the system developers' experiences of the method configuration process and how and to what extent the configuration decisions fulfilled their needs – that is if rationality resonance was achieved. According to Stolterman & Russo (1997), the appreciation of a method is related to how well the rationality of that method matches the rationality of the systems developer. They refer to this as a state of rationality resonance, where the public rationality, inherent in the method, is in harmony with the individual method user's private rationality. Hence, it is an indicator of how well the method fits the situation from the system developers' perspective.

## Analytical framework

The analytical framework used in this paper is based on the concept of method rationale as laid out by Ågerfalk & Fitzgerald (2006a). To facilitate analysis, previous work on the concept was extended by the explicit introduction of the actor as a framework component (see Figure 4). As we shall see below, the actor concept is key to facilitate analyses of rationality resonance.

The framework in Figure 4 consists of a number of components (depicted as UML classes): actor, value, goal, method component (in-concept and in-action) and a number of named associations between these. The framework draws on Weber's (1978) notion of practical rationality, which highlights that people, when acting socially, choose means in relation to ends, ends in relation to values and act in accordance with certain ethical principles. In terms of the framework, this means that

1. values dictate what goals are considered worthy of achieving;
2. method components prescribe how to achieve goals;
3. ethical principles dictate what method components are acceptable.

The term method component refers to a self-contained part of a method that is used in MMC during configuration work (see above). A method component is either 'in-concept' or 'in-action'. This is used to represent that a method component is either described as a prescription for action (in-concept), in, for example, a method handbook or is a result of a method following action in practice (in-action) (Lings & Lundell, 2004; Ågerfalk & Fitzgerald, 2006a).

As can be seen in Figure 4, *value rationale* corresponds to (1) in the list above, and *goal rationale* corresponds to (2). With respect to (2), there is a distinction between a goal that is the direct intention of a method component and higher-level goals that the intention is a way of supporting. For example, the goal (intention) of the method component 'User story' is probably 'to describe a path through the system.' This, however, is only a means to achieve a higher-level goal, such as 'customer requirements understood.' Hence, there is a *goal achievement* relationship between goals that represent how goals support other goals and thus form goal hierarchies, or networks. The same principle applies also to values. This is in line with previous work on goal-oriented requirements engineering (Mylopoulos *et al.*, 2001; Bresciani *et al.*, 2004; Gonzalez-Perez *et al.*, 2009).

The *value base* association represents the fact that not all values are operationalized as concrete goals but reside as guiding principles that the actor turns to as new situations emerge. Note that the value base is allowed to be empty, as it represents only known explicit values. Depending on the stage of analysis, these may still have to be elicited. Rationality resonance occurs when two actors share the same method rationale, that is, when they agree to a common set of values, goals and method components.

## Analysis

Data analysis was carried out in three steps, where Step 1 is reported in the next section and steps 2 and 3 are reported in the following section. Step 1 draws on the relationship between values and goals of a method component. This means tracing the goals and values of XP as expressed in-concept. This was achieved by taking the point of departure in the agile manifesto (see below), in which the XP method (Beck, 2000; Bresciani *et al.*, 2004) is anchored. These goals are thus seen as higher-level goals of the method and are used as input for the

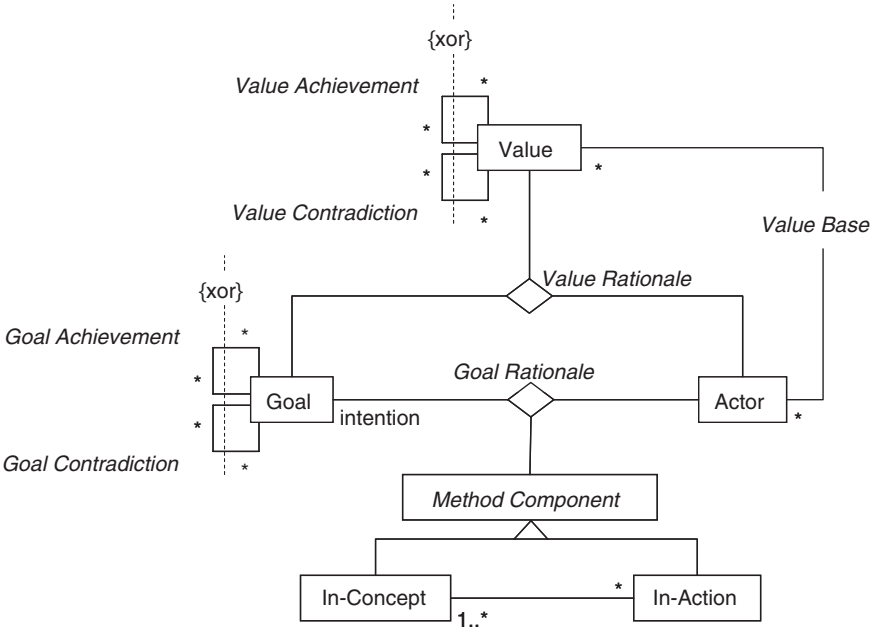


**Figure 4** Conceptual structure of method rationale (using UML notation).

second step. The essence of this analysis is adapted from a workshop paper by the second author (Ågerfalk, 2006).

The second step consists of an analysis of how the chosen configuration corresponds to the higher-level goals of the agile manifesto, which is the foundation of the XP method. This is done through the use of the method component's public rationale, which are the goals in the method component's interface. (See, for example, the User Story component in Figure 2). If it is possible to relate the goals to goals from the agile manifesto, these goals are presented. Otherwise, we argue that it is not possible to establish a relationship between the method component and the overall goals of the method. If these goals and values are in line with the higher-level goals and values of the base method, then the core of the method is preserved.

The third and final step focuses on the rationality resonance achieved between the situational method and the system developers. This analysis is based on the arguments provided in MC Sandbox and the interviews. If a system developer expresses rationality resonance with concern to a specific method component we recorded that this part of the situational method fits the situation. Hence, the situational method fit is traced to each actor in the system development projects studied.

## Agile method values

A number of leading agile method actors summarized the basic principles behind agile methods in what has become known as the 'Agile Manifesto' (see www.agilemanifesto.org, last accessed 15 June 2009). The agile manifesto is presented as a set of values and associated goals (referred to as principles), as shown in Table 3. Advocates of agile methods recognize the relevance of both sides of the value statements in Table 3 but choose to emphasize the first part of each statement more than the second. The values of the agile manifesto are all quite abstract, which is perhaps to expect of value statements that are to be used to judge the suitability of specific goals. The principles, on the other hand, are more specific and lend themselves to be treated as goals in the method rationale framework introduced above. The only principle that is indeed phrased more like a value statement than an achievable and measurable goal is that which refers to face-to-face conversations. Arguably, the first principle actually contains two goals with an achievement association.

When analysing the interrelationships between the goals, we find that there are two clusters of goals all contributing to their own highest-level (or root) goal: one cluster aiming for customer satisfaction and another aiming for sustainable development – hence one externally oriented and one internally oriented. These clusters are depicted in Figure 5 using plus signs to indicate goal contribution. We also find what is arguably a goal contradiction between $g_3$ and $g_7$ indicated by the minus sign.

The value rationale of the agile manifesto can be found by relating the identified goals to the identified values as shown by the graph in Figure 6. Apparently, all values are operationalized into specific goals, except $v_5$, and all goals are grounded in at least one value, except $g_1$. The case of $g_1$ seems to indicate that there is indeed a non-expressed value of the agile manifesto: satisfied customers over, what? In the case of $v_5$, it is unclear how the preference for face-to-face conversations is related to the other components of the agile manifesto. If it is treated as a value, there are no principles that operationalize it as

**Table 3 Agile values and goals (principles)**

| | | |
|---|---|---|
| Values | $v_1$ | Individuals and interactions over processes and tools. |
| | $v_2$ | Working software over comprehensive documentation. |
| | $v_3$ | Customer collaboration over contract negotiation. |
| | $v_4$ | Responding to change over following a plan. |
| | $v_5$ | The most efficient and effective method of conveying information to and within a development team is face-to-face conversation. |
| Goals (principles) | $g_1$ | Our highest priority is to satisfy the customer |
| | $g_2$ | Through early and continuous delivery of valuable software. |
| | $g_3$ | Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage. |
| | $g_4$ | Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale. |
| | $g_5$ | Business people and developers must work together daily throughout the project. |
| | $g_6$ | Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done. |
| | $g_7$ | Working software is the primary measure of progress. |
| | $g_8$ | Agile processes promote sustainable development. The sponsors, developers and users should be able to maintain a constant pace indefinitely. |
| | $g_9$ | Continuous attention to technical excellence and good design enhances agility. |
| | $g_{10}$ | Simplicity – the art of maximizing the amount of work not done – is essential. |
| | $g_{11}$ | The best architectures, requirements and designs emerge from self-organizing teams. |
| | $g_{12}$ | At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly. |

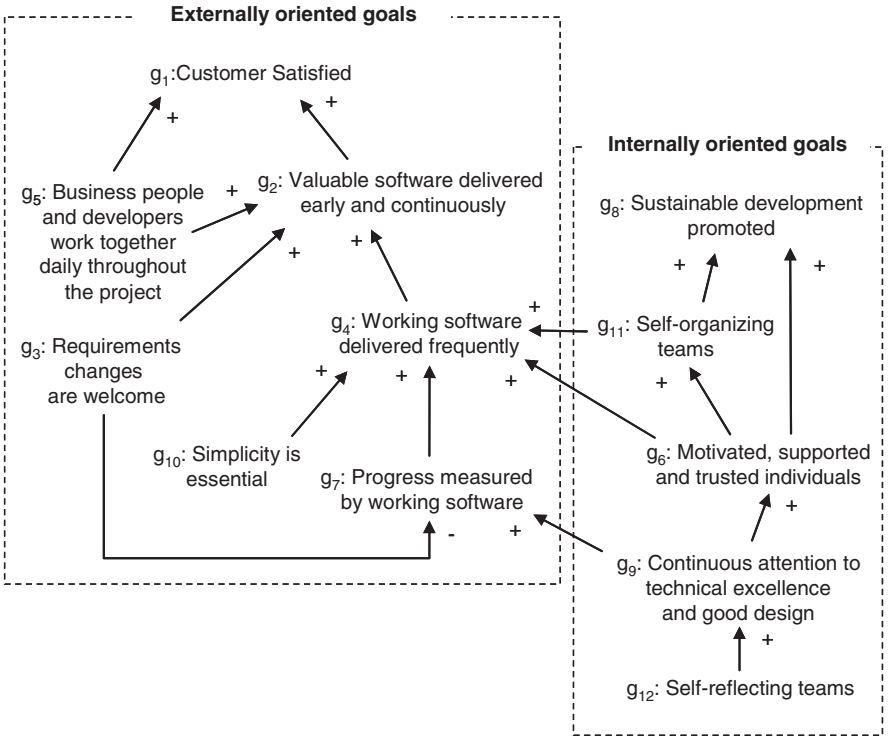


**Figure 5** Goal clusters in the agile manifesto (based on Ågerfalk, 2006).

a goal. If it is to be regarded as a goal, possibly grounded in $v_1$ it is singled out as the only goal that does not contribute to any other goal, besides $g_1$ and $g_8$. As such, it would thus be valued in its own right together with satisfied customers and sustainable development. The most obvious is perhaps to view it as a value with a value achievement association to $v_1$ – preferring face-to-face conversations can probably be traced back to valuing

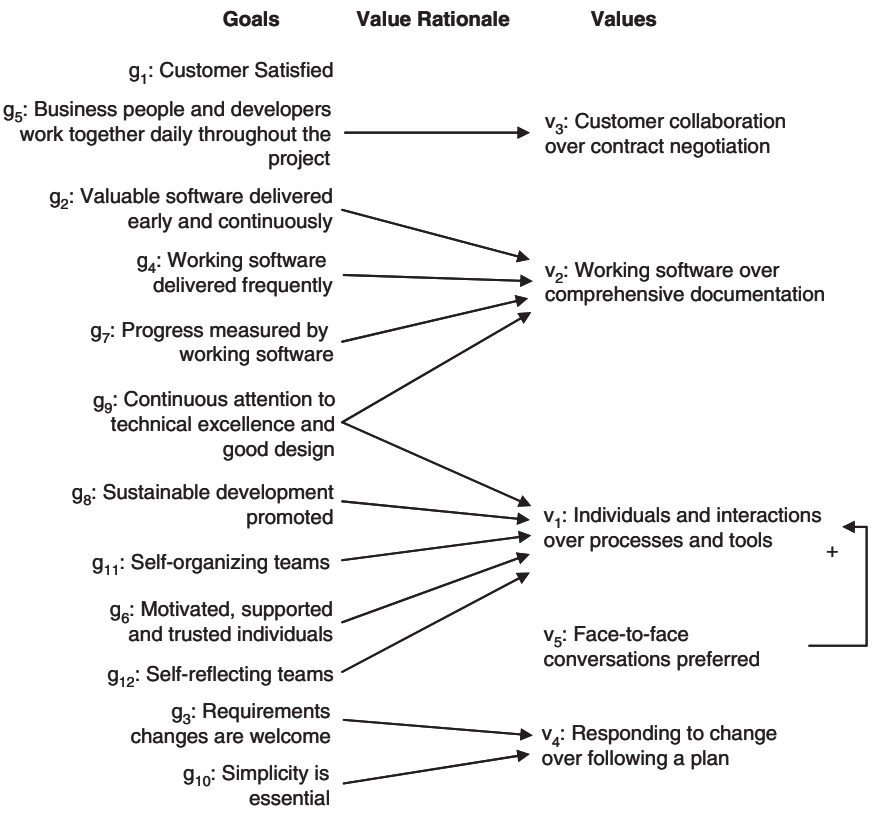


**Figure 6** Value rationale of the agile manifesto.

people and interaction over processes and tools. We also see that two separate value clusters mirror the two goal clusters identified above. This is not surprising and indicates that also the values are oriented mainly internally or externally. It is important to remember that these goals and values are those expressed by the group of people behind the agile manifesto, who thus constitute the actor to whom this method rationale belongs.

## Empirical experiences – examples

The empirical work included method configurations in three different projects. Due to space limitations it is impossible to provide a detailed description of all the method configurations. Accordingly, we make a brief overview of the three configurations and make a selection of examples based on three configuration packages covering the three different types of configuration situations where potential to achieve rationality resonance is satisfactory, unsatisfactory and missing (see Table 1).

## Configuration templates

One important aspect of MMC is the idea of reuse. The three projects shared a number of aspects, such as involving development of web applications. Table 4 contains the characteristics and configuration packages resulting from the method configuration work. The three rightmost columns in the table show the combination of configuration packages for each project; that is they illustrate the configuration templates used. From this table we find that the second and third project shared one configuration template. The first project differed in several respects, but shared the three configuration packages concerning project risk, degree of management commitment and modeling of 'web aspects,' with the other two projects.

## Configuration packages

The content of the configuration templates builds on the configuration packages. We chose to illustrate and exemplify the configuration content with three different configuration packages covering the three degrees of potential to achieve rationality resonance (see Table 1):

- 'Knowledge about business process = Low'
- 'On-site customer = No'
- 'Project risk = Normal (normal planning)'

The first configuration package is chosen to illustrate how the base method has been complemented with method support that is not present in XP – the potential for rationality resonance is missing. The second configuration package is chosen to illustrate how the developers needed to question the operationalization of a basic value in XP (to have a high degree of interaction with the end user) – the potential to achieve rationality resonance is unsatisfactory. The third configuration package illustrates when the potential to achieve rationality resonance is

**Table 4 Overview of configurations**

| *Characteristic* | *Configuration package* | *Project* | | |
|---|---|---|---|---|
| | | *1* | *2* | *3* |
| Knowledge about business process? | High | • | | |
| | Low | | • | • |
| On-site customer? | Yes | • | | |
| | No | | • | • |
| Co-located project team? | Yes | | • | • |
| | No | • | | |
| Project risk? | Normal (normal planning) | • | • | • |
| | High (extended planning) | | | |
| Degree of management commitment? | High | • | • | • |
| | Low | | | |
| Modeling of web aspects | Yes | • | • | • |
| | No | | | |
| Type of testing | Automated | | • | • |
| | Manual | • | | |

satisfactory. Note also that the last configuration package was reused in all three configuration templates.

**'Knowledge about business process = Low'**

This configuration package concerns the project teams plan to increase their knowledge of the business process. The developers concluded, based on their knowledge of the base method, that business modeling is not supported by XP. Consequently, they turned to additional methods to find complements. Since some of the developers had earlier experience of the Rational Unified Process they borrowed parts of its business modeling discipline.

Table 5 shows the method components that the developers chose to add to this configuration package. In total four components were added. The first method component is the Business vision. The developers expressed the importance of 'a high-level view' of the business. As one developer said 'it [the Business vision] can work much like the Vision [card], but for business modeling.' From the system developers' comments in MC Sandbox we find the following: 'Necessary to provide a comprehensive landmark, but still being fairly simple.' Consequently, it was in line with the overall way of working as suggested by XP.

The second component is the Business Use Case. The project members needed a way to capture business process details. Project members viewed the details necessary in order to know 'where to plug the [information] system into the business', or 'not to push a cube through a round whole.' Hence, they expressed a need to understand how the business was to be supported. The third added component is the Business Use Case Model. When examining the different voices from the workshop, we find that this component is closely associated with the Business Use Case component: 'it is needed to create an overview', 'you get that [the Business Use Case Model] into the bargain when working with Business Use Cases.'

Finally, the fourth method component was the Business Glossary. Here, opinions diverged. Two developers viewed this component as 'unnecessary.' However, other developers provided counter-arguments: 'concepts tend to become important' and 'when you are unskilled in a business, terms are difficult [to remember].' In the end the component was added.

Table 6 shows that three of the four added method components have support in the overall goals of XP. The Business Vision component is an operationalization of goal g3 (requirements changes are welcome). All changes in the business process and the requirements can always be discussed in light of the business vision. The second method component, Business Use Case, however, has weak support for the goals of XP. XP is not concerned with requirements for business processes. However, the system developers found the method component important in order to satisfy the customer (g1). The third component, Business Use Case Model, has support in goal g10 (simplicity is essential). The aim of this component is to provide an overall structure of the business use cases, to provide simplicity. Finally, the fourth method component, the Business Glossary, can be anchored in goal g1 (customer satisfied). The glossary is an artifact for creating a shared understanding of business-specific concepts, which facilitates communication both with the customer and within the project team.

When analysing rationality resonance, we find that it exists between the first three method components and all of the system developers in the two project teams that used the configuration packages. This is illustrated in the rightmost column in Table 6, where A2.1 denotes project member one in the second project, and so forth. When it comes to the last method component, the Business Glossary, however, rationality resonance is not achieved with two of the developers–one in the second team and one in the third team.

**Table 5 Configuration package: 'Knowledge about business process = Low'**

| *Method component* | *Method component's rationale* | *Classification* |
|---|---|---|
| c1: Business Vision | To capture and communicate very high-level objectives of a business effort | Added as is |
| c2: Business use case | To describe the value-added aspect of the business process | Added as is |
| c3: Business use case model | To provide an overview of the direction and intent of the business | Added as is |
| c4: Business glossary | To understand the terms that are specific to the project | Added as is |

**Table 6 Analyse of preserved goals and experienced rationality resonance: 'Knowledge about business process = Low'**

| *Agile Goals* | *Method component* | *Actor experience rationality resonance* |
|---|---|---|
| g3: Requirements changes are welcome | c1: Business vision | A2.1–A2.4, A3.1–A3.5 |
| g1: Customer satisfied | c2: Business use case | A2.1–A2.4, A3.1–A3.5 |
| g10: Simplicity is essential | c3: Business use case model | A2.1–A2.4, A3.1–A3.5 |
| g1: Customer satisfied | c4: Business glossary | A2.1, A2.2, A2.4, A3.2–A3.5 |

**'On-site customer = No'**

The 'On-site customer = No' configuration package focuses on how the project teams plan to handle the fact that no customer will be on-site. The developers demarcated the configuration package to requirements aspects of XP (the base method). Table 7 shows the method components and their classifications. The first component is the Vision card. The systems developers expressed that it was important for the customer to be able to 'depict the future system' and that the feasibility of this method component 'does not change with this kind of relationship to the customer.' Hence, the method component was to be performed as described in the base method.

The second method component, the Metaphor, was to be used on an informal basis. As one of the systems developer expressed during a method configuration workshop: 'we use them frequently but usually we do not document them.' According to the MMC classification scheme, such use is classified as 'Perform informal.' The last two method components in Table 7 illustrate the replacement of User stories with Use cases. The rationale for this exchange is captured in the statements: 'we need to compensate [the increased distance] with more details', 'one needs tools that are not that interaction intense', and 'I view it [the use case] as a more suitable way.' Consequently, the User story component was suppressed for the benefit of the Use case component.

Table 8 shows the analysis of preserved goals and experienced rationality resonance for this configuration package. Of the three method components found in the configuration package two of them are original method components from XP. These method components are operationalizations of the goals in the agile manifesto. The Use case component, however, is used instead of the User story component. This component is a different operationalization of goal g5.

Rationality resonance is found between the Vision Card component, the Metaphor component and the developers. Everyone agreed on these two choices. When it came to the choice to drop User Stories in favor of Use Cases, we found that this choice was not uncontroversial in the third project group. During the interviews two out of five system developers expressed that they did not agree on this modification. They expressed a feeling that this modification would not solve the problem with customers not being on-site.

**'Project risk = Normal (normal planning)'**

This configuration package is demarcated to the planning activities in XP. The developers associated this configuration package with the project risk characteristic, and cases where the risk is perceived as normal. Table 9 shows six components that were associated with planning in XP, all of them classified as 'Perform as is'. First, the team members of the different projects agreed that 'an initial assessment is necessary.' Second, they reflected on the necessity 'to discuss what actually fits into the [project] scope' or phrased differently 'to set the [project] scope.' Hence, the Prioritized user stories were included into the configuration package. Third, the team members argued that coordination with the customer organization and elaboration of stage goals were essential: 'planning is needed despite that we know it will change', 'we have to plan the content [for each release]' and 'if we do not plan how can we expect the customer to take responsibility for the delivery'. Consequently, both the Release plan and the Iteration plan were classified as 'Perform as is' in the configuration package. Finally, the Task and the Signed task component were classified as 'Perform as is'. The project members expressed that these component were needed to break down user stories into manageable tasks and to assign responsibility for these tasks. Hence, the developers intended to use this part of the method as prescribed in textbooks.

Table 10 presents how the method components' goals in the configuration package relate to the goals of XP. Overall this configuration package corresponds to XP since the components are used 'as is.' Consequently, the

**Table 7 Configuration package: 'On-site customer = No'**

| *Method component* | *Method component's rationale* | *Classification* |
|---|---|---|
| c5: Vision card | To capture the purpose of the system | Perform as is |
| c6: Metaphor | To describe the system's likeness | Perform informal |
| c7: User story | To describe a path through the system | Omit |
| c8: Use case | To understand the system's behavior | Exchanges as is |

**Table 8 Analyse of preserved goals and experienced rationality resonance: 'On-site customer = No'**

| *Agile Goals* | *Component* | *Actor* |
|---|---|---|
| g3: Requirements changes are welcome | c5: Vision card | A2.1–A2.4, A3.1–A3.5 |
| g10: Simplicity is essential | c6: Metaphor | A2.1–A2.4, A3.1–A3.5 |
| g5: Business people and developers work together daily throughout the project | c8: Use case | A2.1–A2.4, A3.1–A3.3 |

**Table 9 Configuration package: 'Project risk = Normal'**

| *Method component* | *Method component's rationale* | *Classification* |
|---|---|---|
| c9: Estimate | To estimate the amount of time required to implement a story | Perform as is |
| c10: Prioritized user stories | To prioritize functionality in the system | Perform as is |
| c11: Release plan | To schedule releases | Perform as is |
| c12: Iteration plan | To schedule the user stories for each iteration | Perform as is |
| c13: Task | To decompose user stories into tasks | Perform as is |
| c14: Signed task | To assign developers to tasks | Perform as is |

**Table 10 Preserved goals and experience rationality resonance: 'Project risk = Normal'**

| *Goals* | *Component* | *Actor* |
|---|---|---|
| g2: Valuable software delivered early and continuously | c9: Estimate | A1.1–A1.5, A2.1–A2.4, A3.1-A3.5 |
| g2: Valuable software delivered early and continuously | c10: Prioritized user stories | A1.1–A1.5, A2.1–A2.4, A3.1–A3.5 |
| g3: Requirements changes are welcome | c11: Release plan | A1.1–A1.5, A2.1–A2.4, A3.1–A3.5 |
| g3: Requirements changes are welcome | c12: Iteration plan | A1.1–A1.5, A2.1–A2.4, A3.1–A3.5 |
| g4: Working software delivered frequently | c11: Release plan | A1.1–A1.5, A2.1–A2.4, A3.1–A3.5 |
| g4: Working software delivered frequently | c12: Iteration Plan | A1.1–A1.5, A2.1–A2.4, A3.1–A3.5 |
| g10: Simplicity is essential | c13: Task | A1.1–A1.5, A2.1–A2.4, A3.1–A3.5 |
| g6: Motivated, supported and trusted individuals | c14: Signed task | A1.1–A1.5, A2.1–A2.4, A3.1–A3.5 |
| g11: Self-organizing teams | c14: Signed task | A1.1–A1.5, A2.1–A2.4, A3.1–A3.5 |

goals of XP are preserved in the configuration. In addition, all three project teams used this configuration package. When analysing experienced rationality resonance, we see that all project members agreed with the use of these method components.

## Discussion

The research objective of this study was to explore MMC as a tool for method tailoring with the intention to adhere to specific goals and values – in this case agile goals and values. With regard to the configurations made in the three cases some notable lessons can be made.

In the examined cases, MMC was used as quality assurance. Each method component contains the goals that it operationalizes. This conceptual design is anchored in the design principle that method rationale is important during method tailoring. In practice, this selection mechanism has worked well, both during this study and in previous ones (Karlsson & Ågerfalk, 2009). Irrespective of the kind of configuration decisions that have been made during the projects above, these changes have been mapped to agile goals. The following represents the voices of three developers: 'we have discussed our needs,' 'now everybody ought to know why these things [the selected method components] are important' and 'though we did not change it [the method] we discussed why.' Fitzgerald *et al*. (2006) have stated the importance of considering all principles in a method before discarding them, which according to them is rather uncommon. Consequently, MMC has actively

contributed with a discussion of the association between actors, method components and agile goals (see Figure 4). This is unique for method engineering approaches that incorporate method rationale. One important aspect in achieving this focus has been the use of interfaces. Henderson-Sellers *et al*. (2008) have questioned the interface/body model, stating that 'no information needs to be hidden'. However, our study as well as earlier accounts (Karlsson & Ågerfalk, 2005, 2009) indicate that an interface/body model is useful when method users take a lead role in the configuration process, which is the case when working with MC Sandbox.

An interesting distinction between this study and previous research on tailoring of agile methods is the project members' awareness of their tailoring decisions. Fitzgerald *et al*. (2003, 2006) and Henderson-Sellers & Serour (2005) provide reflections on extensions and alliances of agile methods, but these are not discussed at the level of method objectives. Fitzgerald *et al*. (2003) discuss the common problem of a focus on 'low-level method steps' instead of higher-level principles. MMC provides two types of tailoring decisions where extensions and alliances are used: when method parts are added to the base method and when method parts are exchanged. The empirical material shows that the developers made well-considered choices with regard to the higher-level principles – the agile goals and values – during both types of decisions.

The system developers made several extensions as laid out above. For example, XP does not contain a goal such as 'The value-added aspect of the business process described' and the base method could thus not meet the project members' intentions on a detailed goal level. One of the developers expressed that 'we needed to tackle this problem … to agree on a modelling technique' and in MC Sandbox developers expressed that 'each solution has to add [business] value' to keep the 'customer satisfied'. The developers made suggestions based on their need to add method rationale. Each suggestion was compared to the existing goals in the method and the goals of the project. 'Some of us had previously worked with business modeling using [business] use case models … they are a good complement.' Hence, MMC was valuable in the search for complementing method components because it forced the developers explicitly to state what goals they needed to achieve. Two voices of the system developers expressed this as 'a way to express our problems,' and 'MMC guided what to search for.' Furthermore, MMC explicated the existing goals of the method during these discussions. Consequently, the system developers created an association to the root goal in the external goal cluster (see Figure 5).

The cases also show that the system developers introduced goal contradictions into the tailored method. A method component with a business modeling focus are to some extent in conflict with the agile goal 'Valuable software delivered early and continuously,' which is a lower-level goal. However, the interviews revealed that these decisions were deliberate: 'the method has to be balanced,' 'a decisive addition in this case,' 'you have to consider how to keep the customer happy.' Such reflections by developers are not typically reported on in method tailoring research.

When it comes to situations when method parts were exchanged, we find that these changes were carefully crafted with regard to the agile goals. In the specific case above, where the customer was not on-site, the studied projects revealed difficulties in fulfilling the 'daily' part of the agile goal 'Business people and developers work together daily throughout the project.' The use of goals created an initiated discussion on how to choose a different operationalization, and what part of the goal that created the problem. This is supported by such voices from the developers as 'it [method rationale] created awareness,' 'we came to discuss how to mitigate this risk [increased distance],' and 'we were able to check against the different goals.'

Finally, the study shows that in practice, rationality resonance is not something that can be discussed at the project team level. The method rationale framework provided the possibility to trace rationality resonance to individual combinations of project members and method components. Based on mapping of configuration decisions, we found that rationality resonance was not achieved between all system developers and the tailored method. In other words, not all of the project members agreed that the choices made were important goals to achieve or a proper operationalization of a specific goal. One of the system developers that did not want to include a business glossary expressed: 'Though we disagree I understand what they want to achieve … I just do not find it that important.' Another system developer, who disagreed with exchanging user stories for the benefit of use cases, expressed that 'use cases do not compensate [for the increased distance] … I think it was an unnecessary exchange.' Consequently, MMC does not always generate consensus decisions, but it creates an awareness of different project members' different standpoints. Agree to disagree, as it were.

## Conclusion

In this paper, we have reported on system developers' experience from using MMC to manage method tailoring with the intention to promote agile goals and values. The meta-method was used during three system development projects to tailor the agile method XP. This study has been justified by (a) the need for method support in tailoring agile methods to particular development settings, and (b) the need to complement earlier evaluations of MMC with more conclusive tests to determine the effectiveness of the approach with regard to maintaining coherency with the overall goals of the base method during tailoring.

Through content examples of method configurations we have shown that it is possible to use MMC and its conceptual framework with XP. Furthermore, we have

identified reuse of both configuration templates and configuration packages between the three projects we have studied. This corroborates earlier evaluations of MMC. Moreover, three lessons have been learned about MMC's ability to maintain a coherency in the base method's network of goals.

The first lesson concerns MMC as a quality assurance tool. All project members participated in the configuration process and gave their view on how different parts of the base method matched their intentions. This discussion created an awareness of these goals and for whom they were important. However, we also learned that though awareness was created there was not always consensus. Developers sometimes disagreed on different modifications of the base method, but still showed an understanding for the modifications. The second lesson concerns the use of goals as a reference point when extending the base method. We can report on a valuable focus on how the added method components complemented or contradicted the base method. When the system developers introduced goal contradictions into the situational method these decisions were well informed. The system developers argued for anchoring the added functionality in the root goal of the base method, while creating conflicts with lower-level goals. The third lesson concerns support for finding new ways of operationalizing goals that exist in the base method. During two of the three projects the developers identified a project risk with customers not being on-site. Through the use of MMC this risk was traced to agile methods' confidence in daily contact between developers and business people. The developers performed a well thought through change to the operationalization of that goal. Consequently, MMC contributed to the discussion on why the goal was difficult to operationalize and how it could be implemented differently.

The use of method rationale and the chosen level of granularity (tailoring at the product/deliverable level) in this study show a way forward for method engineering in practice. This approach abstracts away from irrelevant detail and emphasizes the fulfilment of goals and values that the organization find important, such as agile ones. Among other things, this allows for domain-specific configurations. Such configurations are grounded in method rationale that is important to the domains of interest to an organization.

Earlier research on MMC and method configuration on plan-based methods has shown that the use of method rationale has been important when deciding what to exclude from the base method. The content serves as an encyclopaedia of possible paths to take. XP, being a lightweight method, provides less guidance when it comes to suggesting possible paths. Consequently, method configuration came to focus on extension rather than exclusion and the use of method rationale as a reference point came to play a more important role during extensions than in earlier studies of plan-based methods.

It is important to note that the concept of rationality resonance by definition depends on the system developers' pre-understanding. It thereby depends on an ongoing learning process, especially when a paradigmatically different approach to system development is introduced in an organization. Consequently, some of the results of this study may depend on a lack of understanding and appreciation of agile values and techniques (the disagreement regarding use cases and user stories could, for example, be a result of misaligned learning curves). It is therefore of interest to in the future conduct also longitudinal studies that investigate the impact of individual as well as organizational learning on method configuration and appreciation of different method constructs.

This study has been carried out on three small system development projects. However, we were interested to follow the tailoring decisions and the reasoning behind these decisions. Clearly, the sizes of the projects were large enough to reveal many tailoring decisions. What may have impact on our results is the choice of agile values. We have used the Agile Manifesto as a baseline for our analysis, but we recognize scholars promoting different sets of values and goals (see, for example, www.agilemodeling.com, last accessed 15 June 2009). Further research could thus broaden the scope to investigate also other notions of agility.

To promote system developer engagement and personal responsibility, the joint configuration decisions elaborated in this research build on the idea of participatory, or method-user-centred, method engineering (Karlsson & Ågerfalk, 2005). This way, system developers can become more aware of choices made than when a method engineer creates a tailored method for them to use. However, participation has at least two inherent and somewhat related problems: scalability and cost. The projects reported on in this research are fairly small projects with regard to the number of project members. Hence, participation was managed quite effortlessly. This may not be possible when projects grow larger and participation becomes more difficult to achieve, if it is to be meaningful for all team members. Also, larger project teams increase cost and there is likely a delicate balance between cost and gains in engagement and understanding. Hence, there is ample opportunity for more research to explore limitations and possible trade-offs in this respect.

## About the authors

**Fredrik Karlsson** is an assistant professor of Informatics at Örebro University. He received his Ph.D. in Information Systems Development from Linköping University. His research about tailoring of systems development

methods, system development methods as reusable assets, and CAME-tools has appeared in a number of IS journals and conferences. He is currently heading the Methodology Exploration Lab at Örebro University and is Deputy Head of the Swedish Business School at Örebro University.

**Pär J. Ågerfalk** is a professor at Uppsala University, Sweden, where he holds the Chair in Computer Science in Intersection with Social Sciences. He received his Ph.D. in Information Systems Development from Linköping University and has held full-time positions at Örebro University, University of Limerick, Jönköping International Business School, and Lero – The Irish Software Engineering Research Centre. His work on open source software, global software development, method engineering, information system design, and conceptual modeling has appeared in a number of leading information systems journals and conferences, including EJIS, MISQ, CACM, ICIS and ECIS.

## References


ÅGERFALK PJ (2006) Towards better understanding of agile values in global software development. In *Proceedings of the Eleventh International Workshop on Exploring Modeling Methods in Systems Analysis and Design* (KROGSTIE J, HALPIN T and PROPER E, Eds), Luxembourg.

ÅGERFALK PJ and FITZGERALD B (2006a) Exploring the concept of method rationale: a conceptual tool for method tailoring. In *Advanced Topics in Database Research* (SIAU K, Ed.), pp 63–78, Idea Group, Hershey, PA.

ÅGERFALK PJ and FITZGERALD B (2006b) Flexible and distributed software processes: old petunias in new bowls? *Communications of the ACM* **49(10),** 26–34.

AYDIN MN, HARMSEN F, VAN SLOOTEN K and STEGWEE RA (2005) On the adaptation of an agile information systems development method. *Journal of Database Management* **16(4),** 24–40.

BAJEC M, VAVPOTIČ D and KRISPER M (2006) Practice-driven approach for creating project-specific software development methods. *Information and Software Technology* **49(4),** 345–365.

BASILI VR and ROMBACH HD (1987) Tailoring the software process to project goals and environments. In *Proceedings of the 9th International Conference on Software Engineering*, pp 345–357, IEEE Computer Society Press, Los Alamitos, CA.

BECK K (2000) *Extreme Programming Explained: Embrace Change*. Addison-Wesley, Reading, MA.

BRESCIANI P, PERINI A, GIORGINI P, GIUNCHIGLIA F and MYLOPOULOS J (2004) Tropos: an agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems* **8(3),** 203–236.

BRINKKEMPER S (1996) Method engineering: engineering of information systems development methods and tools. *Information and Software Technology* **38(4),** 275–280.

BRINKKEMPER S, HARMSEN AF and HAN OEI JL (1994) Situational method engineering for informational system project approaches. In *Proceedings of the IFIP WG8.1 Working Conference on Methods and Associated Tools for the Information Systems Life Cycle* (OLLE TW and VERRIJN-STUART AA, Eds), pp 169–194, Elsevier, Maastricht, the Netherlands.

CAMERON J (2002) Configurable development processes. *Communications of the ACM* **45(3),** 72–77.

FITZGERALD B, HARTNETT G and CONBOY K (2006) Customising agile methods to software practices at Intel Shannon. *European Journal of Information Systems* **15(2),** 200–213.

FITZGERALD B, RUSSO NL and O'KANE T (2003) Software development method tailoring at motorola. *Communications of the ACM* **46(4),** 65–70.

GONZALEZ-PEREZ C, GIORGINI P and HENDERSON-SELLERS B (2009) Method construction by goal analysis. In *Proceedings of the 16th International Conference on Information Systems Development* (BARRY C, LANG M, WOJTKOWSKI W, WOJTKOWSKI G, WRYCZA S and ZUPANCIC J, Eds), pp 76–88, Springer-Verlag, Galway, Ireland.

HARMSEN AF (1997) *Situational Method Engineering*. University of Twente, Utrecht, the Netherlands.

HENDERSON-SELLERS B and SEROUR MK (2005) Creating a dual-agility method: the value of method engineering. *Journal of Database Management* **16(4),** 1–23.

HENDERSON-SELLERS B, SEROUR MK, GONZALEZ-PEREZ C and RALYTÉ J (2008) Comparison of method chunks and method fragments for situational method engineering. In *Proceedings of the 19th Australian Conference on Software Engineering* (HUSSAIN FK, Ed.), pp 479–488, IEEE Computer Society, Perth.

KARLSSON F and ÅGERFALK PJ (2004) Method configuration: adapting to situational characteristics while creating reusable assets. *Information and Software Technology* **46(9),** 619–633.

KARLSSON F and ÅGERFALK PJ (2005) Method-user-centred method configuration. In *Proceedings of the Situational Requirements Engineering Processes – Methods, Techniques and Tools to Support Situation-Specific Requirements Engineering Processes (SREP'05)* (RALYTÉ J, ÅGERFALK PJ and KRAIEM N, Eds), pp 31–43, University of Limerick, Paris, France.

KARLSSON F and ÅGERFALK PJ (2009) Towards structured flexibility in information systems development: devising a method for method configuration. *Journal of Database Management* **20(3),** 51–75.

KARLSSON F and WISTRAND K (2006) Combining method engineering with activity theory: theoretical grounding of the method component concept. *European Journal of Information Systems* **15(1),** 82–90.

KORPELA M, MURSU A, SORIYAN A, EEROLA A, HÄKKINEN H and TOIVANEN M (2004) Information systems research and development by activity analysis and development: dead horse or the next wave? In *IFIP International Federation for Information Processing* (KAPLAN B, TRUEX III D, WASTELL D, WOOD-HARPER A and DEGROSS J, Eds), pp 453–471, Springer, Boston.

KRUCHTEN P (2004) *The Rational Unified Process: An Introduction*. Addison-Wesley, Reading, MA.

KUMAR K and WELKE R (1992) Methodology engineering: a proposal for situation specific methodology construction. In *Challenges and Strategies for Research in Systems Development* (COTTERMAN WW and SEEN JA, Eds), pp 257–269, John Wiley & Sons, Washington, USA.

LEE AS (1989) A scientific methodology for MIS case studies. *MIS Quarterly* **13(1),** 33–51.

LINGS B and LUNDELL B (2004) Method-in-action and method-in-tool: some implications for CASE. In *Proceedings of the 6th International Conference on Enterprise Information Systems* (SERUCA I, CORDEIRO U, HAMMOUDI S and FILIPE J, Eds), pp 623–628, INSTICC Press, Porto.

MYLOPOULOS J, CHUNG L, LIAO S, WANG H and YU E (2001) Exploring alternatives during requirements analysis. *IEEE Software* **18(1),** 92–96.

NIKNAFS A and RAMSIN R (2008) Computer-aided method engineering: an analysis of existing environments. In *The 20th International Conference on Advanced Information Systems Engineering* (BELLAHSNE Z and LÉONARD M, Eds), pp 525–540, Springer, Montepellier, France.

NILSSON AG (1999) The business developer's toolbox: chains and alliances between established methods. In *Perspectives on Business Modelling: Understanding and Changing Organisations* (NILSSON AG, TOLIS C and NELLBORN C, Eds), pp 217–241, Springer-Verlag, Heidelberg.

ODELL JJ (1996) A primer to method engineering. In method engineering: principles of method construction and tool support. In *Proceedings of IFIP TC8, WG8.7/8.2 Working conference on method engineering* (BRINKKEMPER S, LYYTINEN K and WELKE RJ, Eds), pp 1–7, Springer, Atlanta, U.S.A..

PAIGE R and BROOKE P (2005) Agile formal method engineering. In *Proceedings of the Fifth International Conference on Integrated Formal Methods* (ROMIJN JMT, SMITH GP and VAN DE POL JC, Eds), pp 109–128, Springer, Eindhoven, the Netherlands.

PATTON MQ (1990) *Qualitative Evaluation and Research Methods*. Sage, Newbury Park, CA.

PLIHON V (1996) MENTOR: an environment supporting the construction of methods. In *Proceedings of the 3rd Asia-Pacific Softeware Engineering Conference*, p 384, IEEE Computer Society, Washington DC.

QUMER A and HENDERSON-SELLERS B (2006) Measuring agility and adaptability of agile methods: a 4-dimensional analytical tool. In *Proceedings of the IADIS International Conference Applied Computing 2006* (GUIMARÃES N, ISAIAS P and GIOIKOETXEA A, Eds), pp 503–507, IADIS Press, San Sebastian, Spain.

QUMER A and HENDERSON-SELLERS B (2008) An evaluation of the degree of agility in six agile methods and its applicability for method engineering. *Information and Software Technology* **50(4),** 280–295.

RALYTÉ J, DENECKÈRE R and ROLLAND C (2003) Towards a generic model for situational method engineering. In *Proceeedings of 15th International Conference on Advanced Information Systems Engineering* (EDER J and MISSIKOFF M, Eds), pp 95–110, Springer, Berlin.

ROLLAND C and PRAKASH N (1996) A proposal for context-specific method engineering. In *Proceedings of the IFIP TC8, WG8.1/8.2 Working Conference on Method Engineering on Method Engineering* (BRINKKEMPER S, LYYTINEN K and WELKE R, Eds), pp 191–208, Chapman & Hall, Atlanta, U.S.A.

ROSSI M, RAMESH B, LYYTINEN K and TOLVANEN J-P (2004) Managing evolutionary method engineering by method rationale. *Journal of Association of Information Systems* **5(9),** 356–391.

ROSSI M, TOLVANEN J-P, RAMESH B, LYYTINEN K and KAIPALA J (2000) Method rationale in method engineering. In *Proceedings of the 33rd Hawaii International Conference on System Sciences*, pp 1–10, IEEE Computer Society Press, Maui, U.S.A.

RUSSO N, WYNEKOOP J and WALZ DB (1995) The use and adaptation of system development methodologies. In *Proceedings of the International Conference of International Resources Management Association* (KHOSROWPOUR M, Ed), p 162, Idea Group Publishing, Atlanta, USA.

SEROUR MK and HENDERSON-SELLERS B (2004) Introducing agility: a case study of situational method engineering using the open process framework. In *Proceedings of the 28th Annual International Computer Software and Application Conference*, pp 50–57, IEEE Computer Society, Hong Kong.

STOLTERMAN E and RUSSO NL (1997) The paradox of information systems methods – public and private rationality. In *Proceedings of the British Computer Society 5th Annual Conference on Methodologies*. Lancaster, England.

SUNYAEV A, HANSEN M and KRCMAR H (2008) Method engineering: A formal description. In *Proceedings of the In Information Systems Development: Towards a Service Provision Society* (PAPADOPOULUS GA, WOJTOWSKI W, WOJTOWSKI WG, WRYCZA S and ZUPANIC J, Eds), Springer-Verlag, New York.

TOLVANEN J-P (1998) *Incremental Method Engineering with Modeling Tools: Theoretical Principles and Empirical Evidence*. University of Jyväskylä, Jyväskylä, Finland.

VAN SLOOTEN K and HODES B (1996) Characterizing IS development projects. In *Proceedings of the Proceedings of the IFIP TC8, WG8.1/8.2 Working Conference on Method Engineering on Method Engineering: Principles of Method Construction and Tool Support* (BRINKKEMPER S, LYYTINEN K and WELKE R, Eds), pp 29–44, Chapman & Hall, Atlanta, USA.

WEBER M (1978) *Economy and Society*. University of California Press, Berkeley, CA.

WISTRAND K and KARLSSON F (2004) Method components – rationale revealed. In *The 16th International Conference on Advanced Information Systems Engineering (CAISE 2004)* (PERSSON A and STIRNA J, Eds), Springer, Berlin.

WOOD J and SILVER D (1995) *Joint Application Development*. John Wiley & Sons Inc, New York.

YIN RK (1994) *Case Study Research: Design and Methods*. SAGE, Thousand Oaks, CA.