Supporting Collaborative Actions Through Method Engineering – Exploring the Potential of Activity Theory

Fredrik Karlsson[a] and Kai Wistrand[b]

[a] Dept. of Informatics (ESI), Methodology Exploration Lab, Research Network VITS, Örebro University, SE-701 82 Örebro, Sweden, fredrik.karlsson@esi.oru.se

[b] Dept. of Informatics (ESI), Methodology Exploration Lab, Research Network VITS, Örebro University, SE-701 82 Örebro, Sweden, kai.wistrand@esi.oru.se

Abstract

The complex and demanding business of developing information systems often involves the use of different systems development methods such as the Rational Unified Process or the Microsoft Solution Framework. Through these methods the development organisation can be viewed as a collective of actors following different rules in the form of prescribed actions in order to guide a work process in accord with Activity Theory (AT). Very often standardised systems development methods need tailoring for unique projects and strategies for this process have been labelled method engineering. Method configuration, a sub discipline to method engineering is applicable in situations where a single base method is used as a starting point for the engineering process. A meta-method (Method for Method Configuration) has been developed addressing these issues. A fundamental part of this meta-method is the method component construct as a means to facilitate efficient and rationally motivated modularisation of systems development methods. This paper is an exploration of the potential of the method component concept as a way to ensure that method engineering can be performed with an AT perspective.

Keywords: Activity Theory, Method Configuration, Method Component, Method Engineering, Situational Method

1 Introduction

The business of developing information systems can be characterized as complex and demanding. Large projects try to produce and deliver systems that meet the different customers' needs. In order to handle the complexity of the development project and a way to influence the quality of the project outcomes different System Development Methods (or SDMs for short) are often utilised. These SDMs are formalised suggestions how to perform when developing systems. They contain prescribed actions and very often define various artefacts, which are deemed beneficial in relation to a range of development goals.

These SDMs are also used in projects as a way to facilitate collaboration among project members and help them to focus on the projects' outcome, i.e. the system. In this context where one can discern a collective of actors following different rules in form of methods in order to guide a work process that can be likened to Activity Theory (AT) (Korpela et al., 2000). When we choose to view systems development in this light we can see how some of the complexity of this business can be handled through distribution of responsibility among project members. It also becomes obvious that systems development is a task carried out by people in collaboration in and between organisations.

Many developers tend to find support in the concrete instructions provided by SDMs but they also want to be able to change their plans when they feel a need for it. Fitzgerald et al. (2003) identify this as a need for tailoring SDMs. Strategies for solving such problems are commonly referred to as method engineering (Brinkkemper, 1996) and a number of suggested solutions as been presented (Odell, 1996; Ralyté et al., 2003; Harmsen, 1997; Brinkkemper et al., 1999). Much attention has been given to the conceptual harmonisation focus in these strategies. The common denominator seems to be the ambition to create situations where the project team can use a uniquely constructed method that meets their demands in every aspect. This seems valid, however no advice is given regarding how the project team should grasp the method, understand how the method fits their needs, and how the method itself really can support collaboration in a development process. Also, method engineering as a business can be regarded as cumbersome and often overly complex due to the many rules that one must adhere to when following different method engineering strategies (Brinkkemper et al., 1999). Method engineering from an AT perspective could be a way to clarify how collaboration aspects can be incorporated in the business of adapting SDMs to situational needs and perhaps also assist in making method engineering less cumbersome.

Much of the problems connected to the complexity of method engineering have been related to granularity issues. How should one modularise a SDM in a meaningful and efficient way? The concept of method components have been proposed to satisfy the need of a smallest meaningful concept during method engineering (Wistrand and Karlsson, 2004). The ambition is to combine the stability offered by methods in conjunction with pragmatic considerations and the knowledge dimension in methods. One obvious benefit is the possibility to see how the components can be related to each other from a collaboration perspective. This is achieved by focusing on individual goals during the development process and how these goals are related to the overall development process goals. The method components as such express method rationale (Ågerfalk and Wistrand, 2003) and this property is useful for the actors working with method engineering in order to make transparent and well informed choices.

The idea put forth and elaborated in this paper is to explore the potential of the method component concept as a way to ensure that method engineering can be performed with an activity theory perspective with beneficial results.

This paper is organized in six sections, including this opening section. In the following section we discuss this paper in a larger context, the meta-method research project, in order to present the research method adopted. Section three contains a presentation of a model of how to structure an activity according to activity theory. The fourth section describes the method component as a conceptual construct and its relationship to AT. We show in section five a small example of method engineering from an AT perspective utilising the method component construct. We end the paper with the sixth section, containing a short conclusion.

2 Research Method Adopted

This exercise is part of a larger project in the method engineering field. As a means to address the common situation of development organisations using large standard SDMs such as the Rational Unified Process (RUP) or the Microsoft Solution Frame-

work (MSF), a sub discipline to method engineering has been formulated and proposed (Karlsson and Ågerfalk, 2004). This discipline addresses situations where a single base method is the starting point for the method engineering tasks in an organisation and is referred to as method configuration.

In order to support the task of method configuration a meta-method, Method for Method Configuration (MMC), has been developed (Karlsson and Ågerfalk, 2004).One integral part of the MMC is the method component construct as a way to achieve efficient modularisation and explicate the rationality dimension of SDMs.

The exploration of possible benefits of combining AT and method engineering can be viewed as theoretical grounding of the method component concept in a larger research strategy of grounding the MMC. Seen in this larger context it is important, according to Goldkuhl and Cronholm (2003), to integrate the external theoretical grounding with internal and empirical grounding. Internal grounding is about reconstructing and articulating background knowledge, knowledge that might be taken for granted. Consequently, such work focuses on internal consistency of concepts, such as the method component, or the meta-method itself. Empirical grounding on the other hand focuses on design and application of these artefacts. In order to integrate these three grounding processes the development of the MMC has been carried out in a collaborative practice research project (Mathiassen, 2002). During this project the method component concept were developed together with practitioners with the ambition to combine method engineering and the view of methods as social actions.

Turning to the body of knowledge that exists outside the research project is putting the focus on external theoretical grounding. External theoretical grounding itself can be preformed with different focus. One subdivision is between external theoretical grounding with focus on theories within the corresponding research field, in this case method engineering, and grounding with focus on theories outside the research field. The former has been previously been discussed in for example Wistrand and Karlsson (2004), while little has been said about the second category.

In this paper we use the conceptual structure of AT as a starting point for discussing systems development, SDMs and method engineering. We relate the concepts found in a method component to the structure of an activity according to AT. The division of transformation into an activity network is translated to the division of methods into modules, where the focus is on conceptual similarities and differences.

3 Systems Development as an Activity

Spasser (1999) argues for using AT as a conceptual framework during studies in the field of information systems (IS). AT exists in several shapes or interpretations that has been operationalized in the field of IS research (Bødker and Petersen, 2000; Kuutti, 1999; Redmiles and Cleidson, 2003) and applied as a SDM for systems developers' day-to-day work (Korpela et al., 2000). This is quite natural since the original proposition (Vygotsky, 1978) was highly abstract. Consequently, AT could be fruitful to use when studying the concept of method and the business of systems development in order to tailor SDMs.

As a conceptual framework AT consists of several basic elements. In this article we use the framework from Korpela et al. (Korpela et al., 2000), which has earlier been used for analysing systems development. They regard work activity consisting of the elements illustrated in Figure 1. The motive of a collective activity is the trans-

formation of the shared object to an outcome. In a systems development project such a shared outcome is often an information system. The initial requirements are gradually transformed by a project team through shared objects of work, such as use cases, data models, and test scripts. In the project team there exists means for coordination and communication, which often include division of labour and rules. In systems development this type of coordination is addressed through SDMs, interpreting the concept of method in a broad sense including implicit ways of working. If a SDM has been explicitly documented there exists descriptions of the rules to use, otherwise these rules exist implicitly in varying degrees. Even though authors, within the information system field, define 'method' slightly differently (Brinkkemper, 1996; Checkland, 1981; Goldkuhl et al., 1998; Russo and Stolterman, 2000), there seems to be a common understanding that a method consists of three interrelated parts. First, there is a process to guide the performance of activities with sequence restrictions. The process tells a project member what actions to take during a specific phase of an information system development project, thus these are prescriptive actions. Second, there is some sort of notation used to document the results of the activities. For example, the notation for drawing state charts. Finally, we have a set of concepts used to describe the problem domain (i.e., modelling primitives) and the method itself.

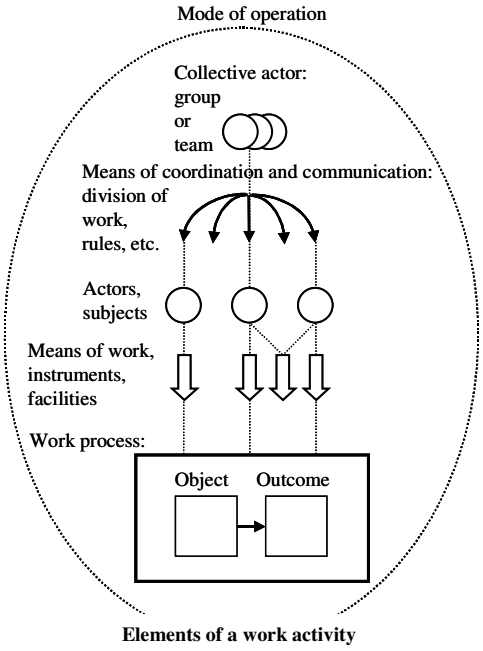

Figure 1: Collective work activity as a systemic entity (Korpela et al., 2000).

Actors perform individual actions based on the SDM in use. Consequently, the SDM is not only used for coordination of a project, it is also a means of work. For example, an actor having the role as business analyst in a project could use the SDM description itself when writing a business use case. For example, when using RUP the method provides templates, which guide the process of writing. Furthermore, SDMs

are nowadays often operationalized in CASE-tools, such as Rational Rose or Microsoft Visio that assist actors during system development work.

Korpela's et al. (2000) systematic view on activity also impose a relative fit between the elements, which is characterized by the mode of operation. It means that in order to make a collective activity such as systems development work the transition between activities must fit each other. For example, if a systems analysts uses the Unified Modelling Language (UML) when designing the data model it could cause problems if the implementer expects it to be delivered in Booch-notation (White, 1994). In this case a translation cost arises or in worse case, the data model may be useless for the implementer.

The mode of operation also point to the need of situational SDMs. In order to achieve a useful fit between activities and their elements, and in terms of methods they need to be tailored. For example, in order to guarantee delivered quality of the developed system, test scripts could be used. However, the writing of test scripts depends highly on supporting activities and their outcomes, such as use and test cases. Otherwise, there exists no explicit SDM support for writing test scripts. In MMC the methods' rationale is important, since it is used for guiding the tailoring process (Wistrand and Karlsson, 2004). The need of a development situation and the goal of the reusable building block, termed method component, determine if the method component should be included in the situational SDM.

4 The Method Component Concept

Korpela et al. (2000) discuss organisations as activity networks, where the outcome of an activity is consumed by another group of actors. A method could, in the same way, be viewed as an activity network. Making a distinction between methods in their ideal state and methods-in-action (Fitzgerald et al., 2002), the former can be viewed as a potential activity network for a project team. Korpela et al. (2000) divide the network of activities using organisational boundaries, where the outcomes are consumed by one or more activities. These boundaries embrace control of certain activities to organisational units, which resembles how control of activities is attributed to certain roles in projects. When discussing tailoring of SDMs according to MMC these boundaries decide what is to be included in a method component, and how an ideal method is transformed into a situational one. We choose to define a method component as:

A Method Component is a self-contained part of a systems development method expressing the transformation of one or several artefacts into a defined target artefact and the rationale for such a transformation.

Consequently, the method component concept shares the transformation view with AT, where objects are transformed to an outcome. However, compared to traditional method engineering, we give less attention to harmonisation of concepts and notations – instead we see a potential in using the collaboration view of systems development in AT when selecting method components, which is drafted below.

The method component construct consists of two basic views: the internal view and the external view. These two views are used in order to facilitate information

hiding and reduce the focus on details during method configuration. The content of a method component is sketched on a conceptual level in Figure 2, which at the same time represent the internal view. A method component consists of method elements, where the most important specialisation is the artefact class, since the method component is an artefact centric concept. A method component's input artefacts are used during the prescribed actions and are transformed to a deliverable, which is the outcome of each component. Subsequently, a method component is viewed as a self-contained module for producing the deliverable.

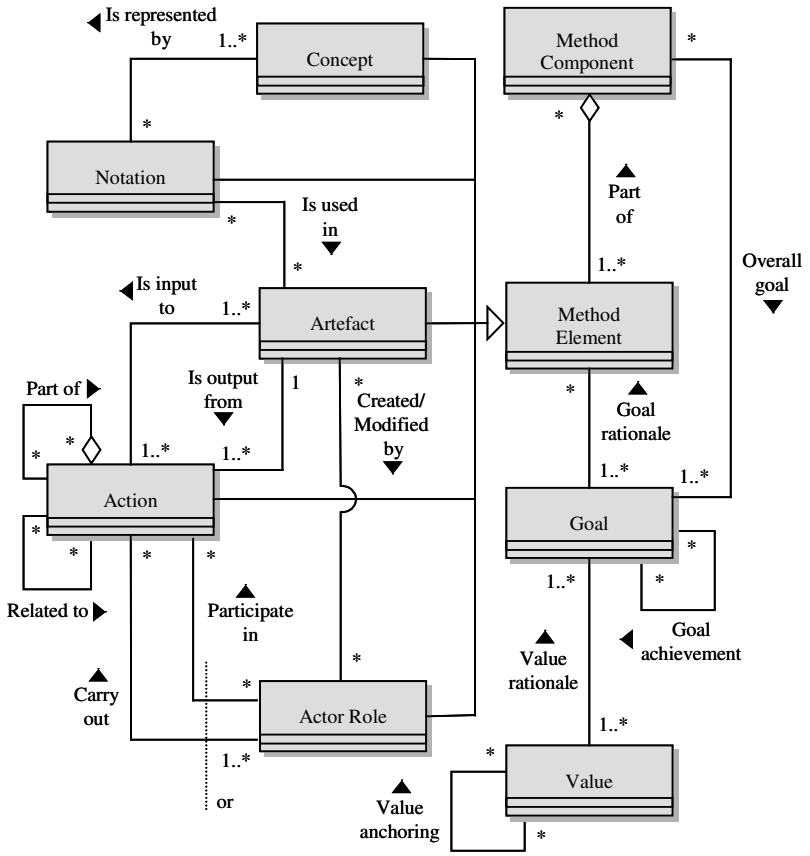

Figure 2: The Method Component Concept (Internal View).

The transformation process is determined by the prescribed actions needed to produce or update the deliverable, which each component inherits from the original SDM that it is part of. Subsequently, the transformation process determines the boundary of a method component, and which prescribed actions to include. For example, in order to transform Business Use Cases, Actors, and Business Rules to a Business Use Case Model using RUP, eleven steps are included in the created method component (Wistrand and Karlsson, 2004). Among these steps we find 'Present the Business

Use-Case Model in Use-Case Diagrams', and it is therefore included in a potential Business Use Case Model component.

The included prescribed actions determine the actor roles that need to be represented during the transformation process, since actors carrying out systems development play different roles during a project. From an AT perspective this determines which actors that are later involved in the actual development process, since actors are assigned to these roles. If we continue our example about a Business Use Case Model component, the business analyst role has to be included since it is the role that creates and modifies the business use case model. This work is carried out based on the input from other method components such as the Business Use Case and the Business Glossary components.

When conducting a SDM's prescribed action, concepts and notation are used to produce artefacts. UML could, for example, be the set of rules used during Business Use Case modelling. Hence, concepts are used to direct the attention to different focal areas during systems development while notation is used to standardise the content in artefacts. Hence, concept and notation are the two remaining method elements in Figure 2.

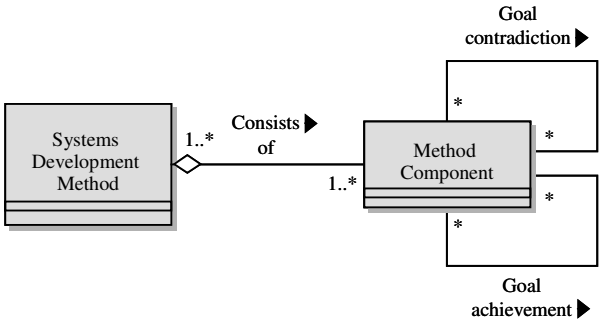

Figure 3: The Method Component Concept (External View).

All method elements in a method component exist for a reason, they contribute to achieving the overall goals of a component. Hence, method elements are anchored in goals that are possible to achieve through the transformation process, goals that are anchored in values of the method creator (Wistrand and Karlsson, 2004). Together goals and values are often considered important constituents of a method's perspective. When working with method configuration according to 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 for achieving collaborative actions. Subsequently, it is a different way of expressing the outcome of a method component, with a focus on why a method component from the base method should be used or why a method component from an additional SDM needs to be added. The aim is to achieve rationality resonance (Stolterman and Russo, 1997) between the system developers' private rationale and the method rationale.

The external view complement the method component's internal view and is used as the primary view during method configuration. In this view of the method

component only a sub-selection of the method elements is presented, together with the component's overall goals. The sub-selection of method elements consists of the recommended input to be consumed and the component's deliverable. Figure 3 illustrates the external view on a conceptual level. The external view of method components addresses the issues concerning how method components are connected to each other, and through their relationships constitute a SDM.

As illustrated in Figure 3 a SDM consists of one or more method components. Examining the association inversely we find that one instance of a method component can be part of several methods, however the component is at least part of one method. The reason is simple; the method component must have an origin somewhere, it was created as part of a SDM from the beginning. Through the external view we focus on how a specific method component contributes to a chain of goal achievements, while we try to reduce the number of possible goal contradictions in the situational SDM.

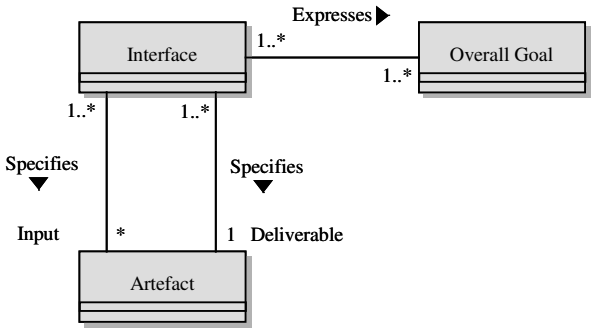

Figure 4: The Method Component's Interface.

The method components in a SDM are connected to each other through the interfaces of the method components. Figure 4 shows the interface on a conceptual level. The interface contains the sub-selection of method elements discussed above. Hence we emulate a feeling of components as black boxes during method engineering activities where the focus is on the overall goals and the component's artefacts. A method component needs to have at least one input artefact. Otherwise, the method component will not have any meaningful support in the method. One exception to this is method components that initiate new focal areas that are later intertwined with the result from other method components. Furthermore, an artefact can be classified both as input and as a deliverable. This is necessary in order to deal with the three generic actions that can be performed on an artefact during development: create, update and delete.

5 Selecting Method Components for Collaborative Actions

The potential of applying a collaborative view when working with method engineering activities, such as selecting method components, can be illustrated through an example using the external view of method components. We have used a demarcated part of MSF for this purpose, which contains six method components. In Figure 5 we find the components: Vision/Scope, Feature Proposal, Usage Scenario, Conceptual Design and Source Code as original MSF components and an additional Prototype

component that we have added. Each method component is illustrated as a box, where we distinguish the Prototype component with the use of broken lines. The arrows illustrate how each component is intended to complement each other in the SDM. Each box is divided into two sections; the top section contains the method component name while the bottom section contains the method component's rationale.

These method components have different actor roles that are responsible for completing the content of each deliverable. Consequently, when we decide which method components to perform, we also affect the possible input for the system development work performed by these actor roles. For example, the product management role is the driver of creating an approved Vision. This Vision is used when creating the Feature Proposal, and indirectly when creating the Usage Scenario. We can, in this example, identify that the development role is responsible for creating the usage scenario content.

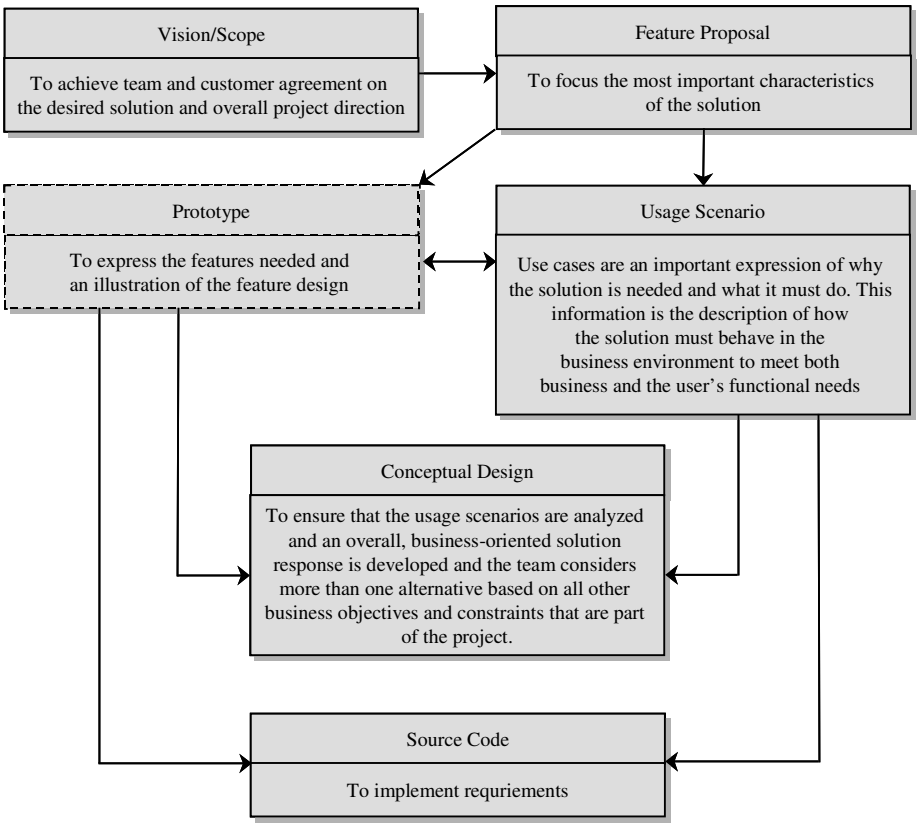

Figure 5: Content example of MSF.

However, the Feature Proposal component should not be performed in a development situation where the project members find the rationale of that component unimportant. For example, the Feature Proposal component might be found superfluous if we will be working with few and straightforward requirements. In such a case omitting

the component does not affect the development role, if the Vision is able to capture these parts.

If we shift focus to the Usage Scenario component we find that it is the key to several related components, in this example we have identified relationships to the Functional Specification, Conceptual Design and Source Code. Together with the latter component we can identify an additional role, the programmer. Omitting the Usage Scenario component would affect the possibilities for the programmer to transform the requirements into functionality. Hence, it would mean reducing the possibility for performing the collaborative action, since representing the functional requirements is central.

However, a method component that focuses on prototypes could share many of the characteristics that usage scenarios have. The rationale of the Usage Scenario component is expressed as 'Use cases are an important expression of why the solution is needed and what it must do. This information is the description of how the solution must behave in the business environment to meet both business and the user's functional needs.' (Microsoft, 2003) The rationale of a prototype can be expressed as 'To express the features needed and an illustration of the feature design' which compared to the usage scenario means making them more visible. Subsequently, if we exchange the Usage Scenario component with the Prototype component the exchange does not affect the collaborative work between the programmer and the developer. Furthermore it does not affect the developer's work with the conceptual design. However, the Prototype component can improve elicitation of requirements in cases where the project members believe that the users requirements will be difficult to externalise and where the use of usage scenario sometimes could have shortcomings. Hence we could improve the potential to achieve rationality resonance between the situational SDM and the project members, and hence support the collaborative process.

6 Conclusion

We see in this paper the potential to complement a method engineering construct for modularisation of systems development methods with the view of collaborative work that is found in Activity Theory (AT). Systems development is a social collaborative activity where the rationale of methods and its relationship to actor roles are important to take into consideration during method engineering activities, which is often a neglected aspect in current method engineering approaches. Through the method component construct we find a potential to focus on how the collaboration is affected through selection of method components, instead of a detailed focus on harmonisation of conceptual issues, which are common in the field of method engineering.

The method component construct has its origin in the method engineering field and is not derived from AT. Subsequently, we can conclude that this delimits the potential in the basic example presented in this paper. The method component construct shares the transformation view with AT. However, this does not mean it is not possible to find improvement areas in order to explore the potential of using a collaborative perspective during method engineering activities. In order to truly discuss the effect on collaborative work, for example, when omitting a method component the actor roles have to receive more attention. In the current design of the method component's interface, all actor roles are hidden inside the method component. Conse-

quently, it is not possible to see the actor role that is affected by such a method engineering decision and how the actor role's responsibilities are affected.

Acknowledgements

The Swedish Knowledge Foundation has financially supported this work.

References

BRINKKEMPER S (1996) Method Engineering: Engineering of Information Systems Development Methods and Tools, Information and Software Technology, 38(4) 275–280.

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

BØDKER S and PETERSEN MG (2000) Design for Learing in Use, Scandinavian Journal of Information Systems, 12.

CHECKLAND PB (1981) Systems Thinking, Systems Practice, Wiley, Chichester.

FITZGERALD B, RUSSO NL and O'KANE T (2003) Software Development Method Tailoring at Motorola, 46(4) 65-70.

FITZGERALD B, RUSSO NL and STOLTERMAN E (2002) Information Systems Development - Methods in Action, McGraw-Hill, London.

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

GOLDKUHL G, LIND M and SEIGERROTH U (1998) Method Integration: The Need for a Learning Perspective, IEE Proceedings - Software, 145(4) 113–118.

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

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

KORPELA M, SORIYAN HA and OLUFOKUMBI KC (2000) Activity Analysis as a Method for Information Systems Development, Scandinavian Journal of Information Systems, 12.

KUUTTI K (1999) Activity Theory, Transformation of Work, and Information Systems Design, In Perspectives on Activity Theory, (Engeström Y, et al.Eds), pp. 360–376, Cambridge University Press, Cambridge, UK.

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

MICROSOFT (2003) MSF Sample Project Lifecycle Deliverables, http://www.microsoft.com/downloads.

ODELL JJ (1996) A Primer to Method Engineering, In Method Engineering: Principles of method construction and tool support (IFIP TC8, WG8.7/8.2 Working conference on method engineering), Atlanta, USA.

RALYTÉ J, DENECKÈRE R and ROLLAND C (2003) Towards a Generic Model for Situational Method Engineering, In Advanced Information Systems Engineering, 15th International Conference, CAiSE 2003, (Johann Eder M M,Ed) Klagenfurt, Austria.

REDMILES DF and CLEIDSON RB (2003) Opportunities for Extending Activity Theory for Studying Collaborative Software Development, In Applying Activity Theory to CSCW research and practice, Helsinki, Finland.

RUSSO NL and STOLTERMAN E (2000) Exploring the Assumptions Underlying Information Systems Methodologies: Their Impact on Past, Present and Future Ism Research, Information Technology & People, 13(4) 313–327.

SPASSER MA (1999) Informing Information Science: The Case for Activity Theory, Journal of the American Society for Information Science, 50(12).

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.

WHITE I (1994) Using the Booch Method, The Benjamin/Cummings Publishing Company, Inc., Redwood City, US.

WISTRAND K and KARLSSON F (2004) Method Components - Rationale Revealed, In The 16th International Conference on Advanced Information Systems Engineering (CAiSE 2004), Riga, Latvia.

VYGOTSKY LS (1978) Mind on Society, Harvard University Press, Cambridge.

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.