



Combining method engineering with activity theory: theoretical grounding of the method component concept

Fredrik Karlsson¹ and
Kai Wistrand²

¹Department of Informatics (ESI), Methodology Exploration Laboratory, Research Network VITS, Örebro University, Örebro, Sweden;

²Department of Informatics (ESI), Methodology Exploration Lab, Research Network VITS, Örebro University, Örebro, Sweden

Correspondence:

Fredrik Karlsson, Department of Informatics (ESI), Methodology Exploration Laboratory, Research Network VITS, Örebro University, Örebro SE-701 82, Sweden.

Tel: +46 19 30 39 94;

E-mail: fredrik.karlsson@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. 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 possible benefits of combining activity theory and method engineering as theoretical grounding of the method component concept.

European Journal of Information Systems (2006) 15, 82–90.

doi:10.1057/palgrave.ejis.3000596

Keywords: activity theory; method configuration; method component; method engineering; situational method

Introduction

The business of developing information systems can be characterised as complex and demanding. Project team members try hard to deliver systems that meet the different customers' needs and requirements. Different systems development methods are often utilised to handle the complexity of development projects and as a way to influence the quality of the project outcomes. These methods are formalised suggestions of how to develop systems. They contain prescribed actions and very often define various artefacts, which are deemed beneficial in relation to a range of development goals. Systems development methods are in many cases based on lessons from earlier failures.

Systems development methods are also used in projects as a way to facilitate collaboration among project members and help them to focus on the projects' outcome, that is, the system. In this context, 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 systems in activity theory (Korpela *et al.*, 2002). When we choose to view systems development by this light we can see how some of the complexity of this business can be handled through distribution of responsibility and labour among

Received: 16 May 2005

Revised: 14 December 2005

Accepted: 3 January 2006

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 systems development methods 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 need for tailoring systems development methods. 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; Harmsen, 1997; Brinkkemper *et al.*, 1999; Ralyté *et al.*, 2003). These strategies give much attention to conceptual harmonisation seamless between method parts. 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). This tends to reduce the involvement of the method users in the process. Method engineering from an activity theory perspective could be a way to clarify how collaboration aspects can be incorporated in the business of adapting systems development methods to situational needs and perhaps also assist in making method engineering less cumbersome.

Much of the problems connected to the complexity of method engineering are related to granularity issues. How should one modularise a systems development method 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 & 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 (Wistrand & Karlsson, 2004) 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 possible benefits of combining activity theory and method engineering as theoretical grounding of the method component concept.

This paper is organised 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 activity theory. In section five we show an example of method engineering from an activity theory perspective utilising the method component construct. We end the paper with the sixth section, containing a discussion and a conclusion.

The research method adopted

The task undertaken in this paper is part of a larger project in the method engineering field. As a means to address the common situation of development organisations using standard systems development methods such as the Rational Unified Process (RUP) or the Microsoft Solution Framework (MSF), a sub-discipline to method engineering has been formulated and proposed (Karlsson & Ågerfalk, 2004). This discipline addresses situations where a single method, termed the base method, is the starting point for the method engineering tasks in an organisation; this sub-discipline is referred to as method configuration.

The meta-method Method For Method Configuration (MMC) (Karlsson & Ågerfalk, 2004) and the Computer-Aided Method Engineering (CAME) tool MC Sandbox (Karlsson & Ågerfalk, 2005) have been developed to support the task of method configuration. One integral part of the MMC is the method component construct as a way to achieve efficient modularisation and explicate the rationality dimension of systems development methods.

The exploration of possible benefits of combining activity theory and method engineering can be viewed as theoretical grounding of the method component concept in a larger research strategy of grounding the MMC and the MC Sandbox. Seen in this larger context it is important, according to Goldkuhl & 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, this kind of 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 during development of the MMC and the MC Sandbox the project have been based on a collaborative practice research strategy (Mathiassen, 2002). As part of this project, the method component concept was 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 performed with different focus. One possible sub-division 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 previously been discussed in, for example, Wistrand & Karlsson (2004), while little has been said about the second category.

In this paper, we use the conceptual structure of activity theory as a starting point for discussing systems development, systems development methods and method engineering. In addition, the existing method component's meta-model (Wistrand & Karlsson, 2004) has been used as input for the analysis. Its existing concepts and structure are the existing tool for creating focus during method engineering. The aim of the analysis has been to determine the limitations of the current design and what changes are needed in order to emphasise the activity theory perspective during work with method components. Subsequently, the analysis has focused on mapping of goals, concepts and relationships. First, goals have been important to analyse since they address the rationale for the model. Second, concepts are the operationalisations of goals and create focus during method-modelling tasks. Third, relationships are important since they determine how concepts are related to each other and what relationships that are addressed as important in the different constructs. Furthermore, together concepts and relationships determine the content of aggregated concepts.

The meta-model of method components is a significant building block in the MMC as well as the MC Sandbox and changes to this model affect prescribed actions in both artefacts. This is natural since prescribed actions on the meta layer is built around the concepts of the meta-method.

Systems development as an activity

Spasser (1999) argues for using activity theory as a conceptual framework during studies in the field of information systems. Activity theory exists in several shapes or interpretations that have been operationalised in the field of information system research (Kuutti, 1999; Bødker & Petersen, 2000; Redmiles & Cleidson, 2003) and applied as systems development method for systems developer in their everyday work (Korpela *et al.*, 2004). However, the existence of different versions is quite natural since the original proposition (Vygotsky, 1978) was highly abstract. Still we can conclude that activity theory can be fruitful to use when studying the concept of method and the business of systems development in order to tailor systems development methods.

As a conceptual framework activity theory consists of several basic elements. In this article we use the framework of Korpela *et al.* (2004), which has earlier been used for analysing systems development. According to them a work activity consists of the elements illustrated in Figure 1. The motive of a collective activity is the

transformation of shared objects to an outcome. In a systems development project such shared outcome is often an information system. The end users' 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 systems development methods, interpreting the concept of method in a broad sense that includes implicit ways of working. If a systems development method 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 (Checkland, 1981; Brinkkemper, 1996; Goldkuhl *et al.*, 1998; Russo & Stolterman, 2000), there seems to be a common understanding that a method consists of three inter-related parts. First, there is a process to guide the performance of activities with sequence restrictions. The process tells a project member what individual actions to take during a specific phase of an information system development project, thus these are prescriptive actions. For example, based on the way of working in RUP a team member choose to implement a use case as source code that is included in the implementation model. Second, there is some sort of notation used to document the results of the activities. For example, the Unified Modelling Language (UML) notation for drawing state charts. Finally, we have

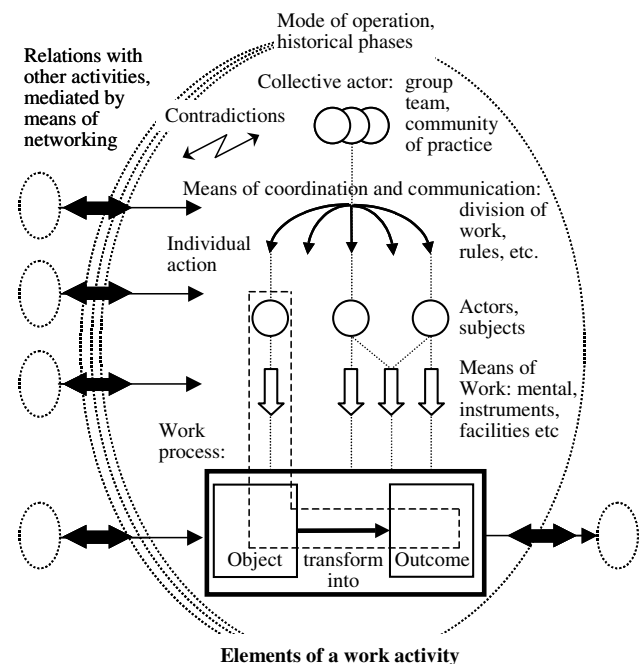


Figure 1 Collective work activity as a systemic entity (Korpela *et al.*, 2004).

a set of concepts used to describe the problem domain (i.e., modelling primitives) and the method itself. Concepts such as state, transition and event are important when drawing state charts.

Actors perform individual actions based on the chosen systems development method. Consequently, the systems development method 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 can use the systems development method description itself when writing a business use case. For example, when using RUP the method provides templates, which guide the process of writing. Furthermore, systems development methods are nowadays often operationalised in Computer-Aided Systems Engineering (CASE) tools, such as Rational Rose or Microsoft Visio that assist actors during system development work.

Korpela's *et al.* (2004) systematic view on activity also impose a relative fit between the elements, which is characterised 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 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 systems development methods. In order to achieve a useful fit between activities and their elements methods need to be tailored. Stolterman & Russo (1997) discuss this need as rationality resonance. For example, in order to guarantee delivered quality of the developed system, test scripts can 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 systems development method support for writing test scripts. In the MMC the methods' rationale is important, since it is used for guiding the tailoring process (Wistrand & Karlsson, 2004). The need of a development situation and the goal of the reusable building blocks, termed method components, determine if they should be included in the situational systems development method.

The method component concept

Korpela *et al.* (2004) discuss organisations as activity networks, where the outcome of an activity is consumed by another group of actors and their activities. A method can, 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.* (2004) 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 systems development methods according to the 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 activity theory, 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 which is found in the activity theory. Consequently, this view which is drafted below is used when selecting method components.

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. 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.

The transformation process is determined by the prescribed actions needed to produce or update the deliverable; prescribed actions that each component inherits from the original systems development method 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, 11 steps are included in the created method component (Wistrand & 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. From an activity theory 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

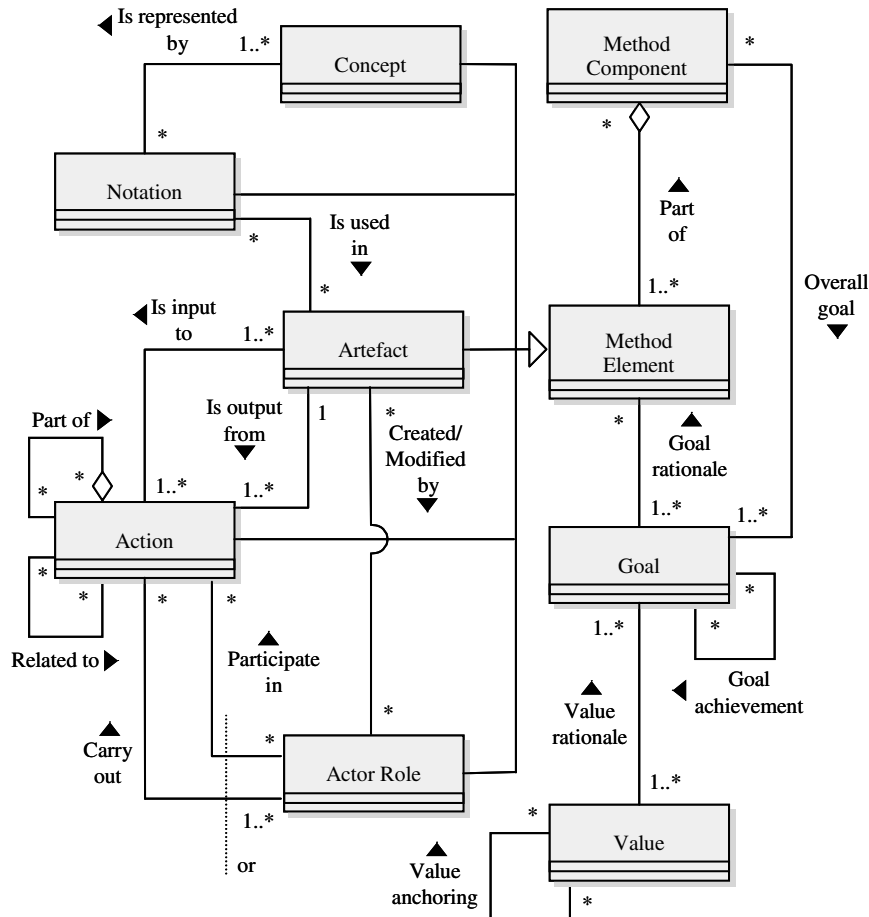


Figure 2 The method component concept (internal view).

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 systems development method'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 such as business actor and business entity are used to direct the attention to different focal areas during Business Use-Case modelling while the notation is used to standardise the content in the produced artefacts. Hence, concept and notation are the two remaining method elements in Figure 2.

All method elements in a method component exist for a reason, they contribute to achieving the overall goals of the 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 & Karlsson, 2004). Together goals and values are often considered important constituents of a method's perspective (Jayaratna, 1994). 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. Korpela & Mursu (2003) express it as the goal of action. This goal-centric view focuses on why a method component from the base method should be used or why a method component from an additional systems development method needs to be added. The aim is to achieve rationality resonance (Stolterman & Russo, 1997) between the system developers' private rationale and the method rationale.

The external view complements the method component's internal view and is used as the primary view during method configuration. In this view 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 systems development method, ideal or situational.

As illustrated in Figure 3 a systems development method 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 systems development method 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 systems development method.

The method components in a systems development method 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 systems development method. One exception to this rule are 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.

Selecting method components for collaborative actions

The potential of applying a collaborative view when working with method engineering activities, such as

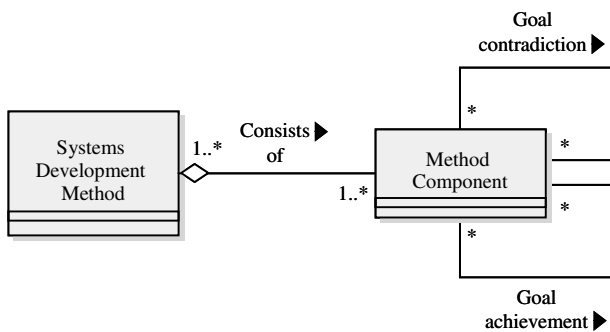


Figure 3 The method component concept (external view).

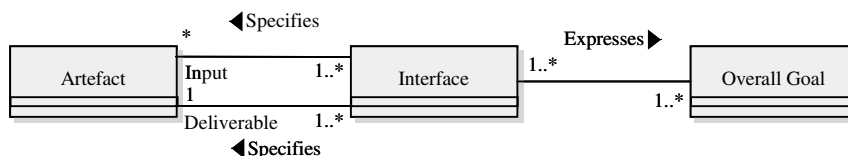


Figure 4 The method component's interface.

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 systems development method. 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.

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 team members believe that 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 Feature Proposal, 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 this 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

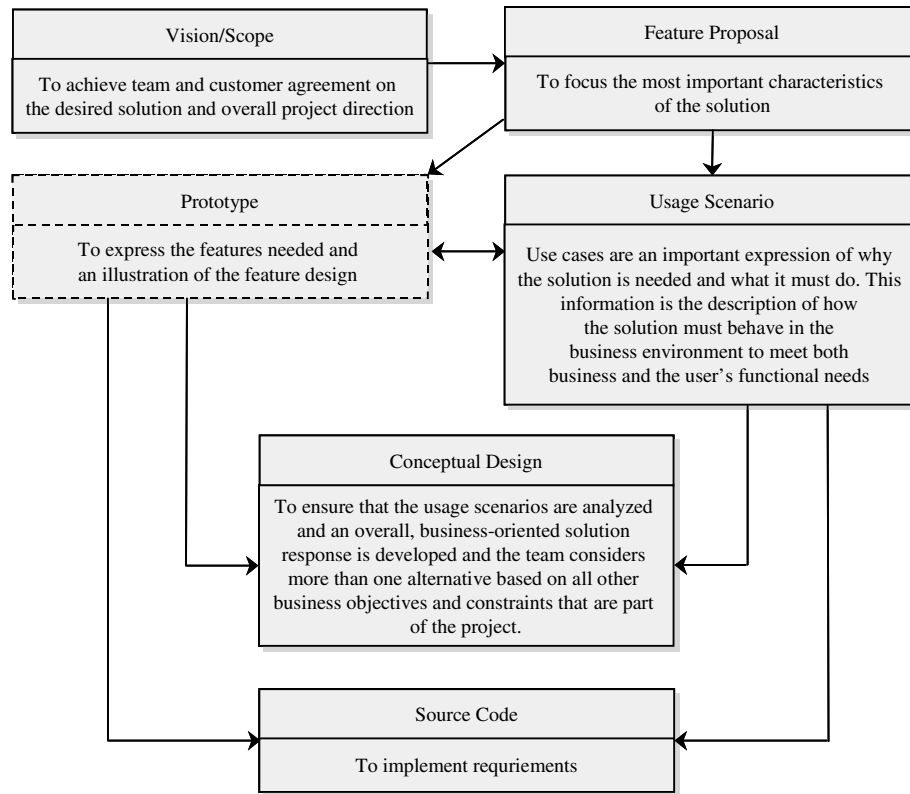


Figure 5 Content example of MSF.

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 the features more visible. Subsequently, if we use the Prototype component in exchange it 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 utilising usage scenario sometimes could have shortcomings. Hence we can improve the potential to achieve rationality resonance between the situational systems development method and the project members, and hence support the collaborative process.

Discussion and conclusion

The task undertaken to combine activity theory and method engineering has a potential to improve the practical work with tailoring of methods as well as bringing a different theoretical focus to the method-

engineering field. The main reason in both cases is the focus on methods as tools for collaborations.

Systems development is a social collaborative activity. However, this is often a neglected aspect in current method-engineering approaches. The method engineering is often criticised for the emphasised focus on the method artefact as such and for a rigid view on methods. For example, Nandhakumar & Avison (1999) criticise method engineering for the belief that it is possible to incorporate all required knowledge in a method. They argue for a softer approach towards methods based on background knowledge about the world instead of a specific method. We agree with Röstlinger & Goldkuhl (1996) that methods should not be in the forefront during systems development and that method should be viewed as heuristic procedures or heuristics (Langefors, 1995). However, through grounding in external theories it is possible to incorporate other aspects in existing method engineering theories and constructs and hence to address such criticism. The use of activity theory in this paper is one example and it directs the attention to improvements of an existing method-engineering construct, the method component concept and the line of reasoning about method configuration.

However, bringing activity theory into method engineering should not be viewed as a theoretical exercise. Rather the suggested improvements have practical implications since the meta-model has impact on the

available tools for method engineers, for example the MMC and the MC Sandbox. Viewing methods as tools for collaborative actions points at the importance of selecting method components that support such collaboration. Subsequently, it shifts the focus from conceptual issues to a more aggregated discussion about what different team members need as input in different phases of a project and what other team members have the possibility to provide. Emphasising a different selection of method elements in the method component can hence act as a bridge between the necessary focus on the method artefact and the project team during method engineering, which can improve the use of methods during projects. Karlsson & Ågerfalk (2005) have reported on the benefits from placing conceptual issues in the background using the rationale of method components as the starting point for method configuration. It has made it possible to involve team members in the method engineering process and where they pick out the components to utilise. Hence, viewing the selection of method components from a collaboration point of view may explain earlier empirical findings in for example Karlsson & Ågerfalk (2005). Team members have expressed the value of participation in the method-engineering process where a larger part of the team has jointly discussed their choices – and not the method engineer's choices. The emphasis of the actor role is another step in improving the possibilities to achieve rationality resonance between team members and the situational method. Furthermore, it brings both collective and individual aspects into work with situational methods.

Consequently, the external theoretical grounding in activity theory both confirms earlier design decisions and suggests new improvements. Among the most important design decisions in the first category we find the use of a method component interface. This design feature makes it possible to suppress details during method-engineering work and to emphasise a sub-selection of method elements. Related to that construct we also find support for promoting method rationale, the purpose of each component.

The second category, design improvements, also addresses the method component interface. 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

About the authors

Dr Fredrik Karlsson, Ph.D., is Assistant Professor (universitetslektor) in Informatics at Örebro University, Sweden. Dr. Karlsson's current research centres on development and tailoring of system development and computerised tool support for such processes. He is a member of the Methodology Exploration Lab (www.oru.se/esi/melab).

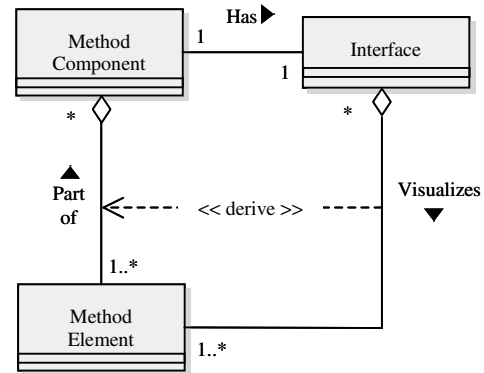


Figure 6 Generic interface.

roles are hidden inside the method component. Consequently, 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.

From a collaboration view on methods we suggest a design change of the method component interface. The most straightforward design change is to associate the actor role with the method component interface. Such a design change has been proposed in Karlsson (2005) but is an inflexible design solution when considering the CAME-tool implementation discussed in Karlsson & Ågerfalk (2005). The collaboration perspective of activity theory is only one possible source of inspiration and adding actor roles to the interface does not allow for more flexibility than the existing design, since it is a closed design. Therefore, we find a need to create a more generic interface where we can emphasise the selection of method elements that the method engineer find relevant. Figure 6 illustrates a modified version of the method component interface. The selection of method elements that is emphasised through the interface is derived from the method elements that are part of the method component. In the case of activity theory the method elements that we promote is: artefacts, overall goals, and the actor role that has an explicit responsibility for the method component's result.

Acknowledgements

The Swedish Knowledge Foundation (KK-stiftelsen) and Örebro University the Department of Informatics (ESI) have financially supported this work.

Mr Kai Wistrand is Ph.D. Student in Informatics at Örebro University, Sweden. Mr. Wistrand is currently working on his dissertation on method rationale in systems development methods. He is a member of the Methodology Exploration Lab (www.oru.se/esi/melab).

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 HARMSSEN 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 learning in use. *Scandinavian Journal of Information Systems* **12**, 61–80.
- 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 theoretic 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.
- HARMSSEN AF (1997) *Situational Method Engineering, Doctoral Dissertation, Moret Ernst & Young Management Consultants*. Utrecht, The Netherlands.
- JAYARATNA N (1994) *Understanding and Evaluating Methodologies*. McGraw-Hill, London.
- KARLSSON F (2005) Method components from the horizon of activity theory. In *Promote IT 2005* (BUBENKO Jr J, ERIKSSON O, FERNLUND H and LIND M, Eds), pp 269–278, Studentlitteratur, Borlänge, Sweden.
- 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 *Situational Requirements Engineering Processes – Methods, Techniques and Tools to Support Situation-Specific Requirements Engineering Processes (SREP’05)* (RALYTE J, KRAIEM N and ÅGERFALK PJ, Eds), pp 31–43, University of Limerick: Paris, France.
- KORPELA M and MURSU A (2003) *Means for Cooperative Work and Activity Networks: an Analytical Framework, in Applying Activity Theory to CSCW Research and Practice*. Helsinki, Finland.
- 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 *Systems Research. Relevant Theory and Informed Practice. IFIP TC8/WG 8.2 20th Year Retrospective: Relevant Theory and Informed Practice-Looking Forward from a 20-Year Perspective on IS Research* (KAPLAN B, TRUJEX III DP, WASTELL D, WOOD-HARPER AT and DEGROSS JI, Eds), pp 453–471, Kluwer Academic, Manchester, United Kingdom.
- KORPELA M, MURSU A, SORIYAN A and OLUFOKUNBI K (2002) Information systems development as an activity. *Computer Supported Cooperative Work* **11**(1–2), 111–128.
- KUUTTI K (1999) Activity theory, transformation of work, and information systems design. In *Perspectives on Activity Theory* (ENGSTRÖM Y, MIETTINEN R and PUNAMÄKI R-L, Eds), pp 360–376, Cambridge University Press, Cambridge, UK.
- LANGFORS B (1995) *Essays on Infology: Summing up and Planning for the Future*. Studentlitteratur, Lund.
- 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>.
- NANDHAKUMAR J and AVISON DE (1999) The fiction of methodological development: a field study of information systems development. *Information Technology & People* **12**(2), 176–191.
- 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)* (BRINKKEMPER S, LYTTINEN K, and WELKE RJ, Eds). 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 MM, 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.
- RÖSTLINGER A and GOLDKUHL G (1996) Generisk Flexibilitet – På Väg Mot En Komponentbaserad Metodsyn, Research Report in Swedish, Department of Computer and Information Science, Linköping University.
- RUSSO NL and STOLTERMAN E (2000) Exploring the assumptions underlying information systems methodologies: their impact on past, present and future IS 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), 1136–1138.
- STOLTERMAN E and RUSSO NL (1997) The paradox of information systems methods – public and private rationality. In *The British Computer Society Fifth 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)* (PERSSON A and STIRNA J, Eds), Riga, Latvia.
- VGOTSKY LS (1978) *Mind on Society*. Harvard University Press, Cambridge.