2007

Method Modifications in a Configuration Management Environment

Werner Esswein
Lehrstuhl für Wirtschaftsinformatik, insb. Systementwicklung, Technische Universität Dresden, werner.esswein@tu-dresden.de

Jens Weller
Technische Universität Dresden, IWM, jens.weller@tu-dresden.de

METHOD MODIFICATIONS IN A CONFIGURATION MANAGEMENT ENVIRONMENT

Esswein, Werner, Technische Universität Dresden, Chair of Information Systems, esp. Systems Engineering, 01062 Dresden, Germany, werner.esswein@tu-dresden.de

Weller, Jens, Technische Universität Dresden, Chair of Information Systems, esp. Systems Engineering, 01062 Dresden, Germany, jens.weller@tu-dresden.de

Abstract

Using configuration management (CM) within the method engineering discipline is a relatively new field of research (Greiffenberg, 2003; Saeki, 2006; Saeki & Oda, 2005). CM administrates revisions and variants of methods and provides a change process to control method modifications. Thus, method CM realizes the traceability of method changes, enables distributed method engineering and ensure a constantly high quality of engineered methods.

In the past years, research focused on the development of appropriate method CM systems. The consequences of method modifications in such systems, however, have attained only little attention or were discussed only in a superficial way. As method modifications can lead to several problems (Weller & Esswein, 2006), it is important to know the consequences of such modifications to forecast the costs of a method change. Furthermore, this knowledge is useful to create or enhance appropriate CAME tools supporting (situational) method engineering.

In this paper, we examine consequences of method modifications within configuration management. Thereby, we focus on the situational method engineering approach and the effects of method modifications on already instantiated models. We discuss critical modifications that lead to inconsistent models and analyze conditions for those problems.

Keywords: Configuration Management, Method Engineering, Method Modifications

1 INTRODUCTION

Conceptual models have been intensely used for the development and adaptation of software systems (Floyd, 2001; Dieste et al., 2000) in the past years. Moreover, they play an increasingly important role within business process reengineering activities to support the design and re-design of organizations and to document the company's knowledge (Rosemann et al., 2005).

To assist the modelling process and to define certain criteria of the resulting models, standardized methods, like UML (OMG, 2006b), ARIS (Scheer, 2000) or BPMN (OMG, 2006a) have been developed and enhanced over the last years. While those methods are very helpful in common projects, there are always situations, where they do not match the modellers needs. The situational method engineering approach has been developed to support the creation of methods, tailored to the requirements of a specific situation (Brinkkemper et al., 1998; Harmsen et al., 1994; Tolvanen, 1998). Thereby, methods are created especially for the use within a specific project and the methods are adjusted continuously to the changing needs within the project.

While situational method engineering increases the acceptance of (adjusted) methods (Fitzgerald, 1997), there are also some management problems arising within that context. The continuous adaptation of methods leads to an increasing number of method versions organizations must deal with (Saeki, 2006; Tolvanen, 1998). To avoid quality problems due to uncontrolled method modifications, a change process has to be installed. Therefore, method versions have to be administrated and modifications have to be documented.

As it addresses similar questions, configuration management (CM) can help to implement such a change process. CM deals with the administration of several versions of artefacts, controls their modifications and, therewith, supports the traceability of changes. The activities of configuration management and the design of corresponding CM systems have been intensely discussed in the software engineering literature (Conradi & Westfechtel, 1998; Estublier et al., 2002; Zeller, 1997). In the past years the positive findings were adopted to the IS discipline to support the maintenance of conceptual models (Braun et al., 2006; Thomas, 2006) and latterly CM was used also within the method engineering field (Greiffenberg, 2003; Saeki, 2006; Saeki & Oda, 2005). Method CM systems administrate meta-models representing methods. Additionally, such systems store the results of the method use – the models – as well as the relationships between models and their meta-models.

Most of the method CM literature focus on the development of adequate CM systems (Greiffenberg, 2003; Saeki, 2006). The consequences of method changes, however, have attained only little attention or were discussed only in a superficial way. Mostly, method modifications are described as being trivial (Greiffenberg, 2003), a detailed discussion about differences between abstract and concrete syntax, e.g., is missed in most of the cases (Saeki, 2006). Method modifications, however, can lead to several problems (Weller & Esswein, 2006) and thus, from an economical viewpoint, it is important to know the consequences of such a change.

In this paper, we classify the modifications of methods administrated by a configuration management system. Thereby we focus on methods created for a specific situation at hand and being modified frequently within a modelling project (situational method engineering). We discuss critical method modifications, analyze their effects on already instantiated models and identify conditions leading to inconsistent models. Our findings can help to forecast effects of method changes (business management viewpoint) and secondly, they can be used for the development of a CAME[1] tool with integrated method engineering CM system (IS viewpoint).

[1] CAME = computer aided method engineering

The paper is structured as follows. In section 2, we explain the fundamentals of conceptual models, method engineering and configuration management. Section 3 gives a survey of the fundamentals of configuration management of methods. Afterwards in section 4, we discuss the consequences arising from method modifications. The paper ends with conclusions and suggestions for further research.

2 BACKGROUND

2.1 Models in the Information Systems Discipline

A model is understood as the result of a construction process "... done by a modeller, who examines the elements of a system for a specific purpose ..." (Schuette & Rotthowe, 1998), which is defined by the user of the model. Depending on the kind of system that is examined during the modelling process (the problem domain), models in the information systems field can be classified as design models or conceptual models. While design models represent software systems or parts of it, conceptual models describe real world phenomena (Dieste et al., 2000; Evermann & Wand, 2005; Wand & Weber, 2002). The language used to express the model is called the modelling language. Different modelling languages are currently used, which comply with different problem domains as well as with different modelling purposes (Brinkkemper et al., 1998). Lastly, models are valid for a certain time interval leading to constant modifications of these models during their lifetime.

In the information systems discipline, models are used to redesign organizations or to develop software systems (Fettke & Loos, 2002; Rosemann et al., 2005). Therefore, graphical models have been established as "...a medium to foster communication with prospective users..." (Frank, 1999). For reasons of simplicity, complex models often consist of several views showing just a section of the whole model (Scheer, 2000; Strahringer, 1996). The different views, however, are not separated but integrated using the same model elements within different views (Greiffenberg, 2003; Schuette & Rotthowe, 1998). Thus, we can argue that a model always consists of its content and one ore more graphical representation, each of which illustrates a set of the model's elements and their relationships. Considering this aspect, modelling languages comprises the definition of a graphical representation (concrete syntax), the context (abstract syntax) and the meaning of the constructs (semantics) defined within the language (Clark et al., 2002).

To assist the modelling process, methods have been established within the IS discipline. IS methods consist of a specification of activities describing the steps of how to build up a model, and a description of the modelling language used for modelling (Harmsen et al., 1994; Jeusfeld et al., 1998; Tolvanen, 1998; Wand & Weber, 2002). The description of the modelling language is also called product description; the activity part is called process description (Brinkkemper, 1996).

2.2 Method Engineering

Method engineering is a discipline "… to design, construct and adapt methods, techniques and tools for the development of information systems." (Brinkkemper, 1996) Within the method engineering process, meta-models are used as a technique to describe the created or adapted methods. According to the above-mentioned parts of a method, meta-models can represent the product description as well as the process description of a method (Strahringer, 1996). Most of the standard methods provide their meta-models to support the modeller in using the method concepts (OMG, 2006b; Scheer, 2000).

Using a given method is sometimes difficult when creating models of a specific domain the language was not explicitly made for (Brinkkemper et al., 1998). In this case, modellers might want to add new concepts or rules that are not covered by the original modelling language. The situational method engineering approach addresses that problem (Harmsen et al., 1994; Tolvanen, 1998). Following that approach, methods used within a modelling project are adapted permanently according to the needs of

the modellers. An overview about the different types of (situational) method engineering is given in (Ralyté et al., 2004).

To support the modelling process with different, individual methods, CAME tools have been widely established. Those tools enable the user to define and modify methods that will be used within the modelling project. Afterwards, the methods can be used for modelling instantly (Cubetto, 2006) or for creating new modelling tools based on the defined method (Keller et al., 1996).

2.3 Configuration Management

Configuration management (CM) has been intensely discussed in theory and established in practice for many years. Norms like ISO 2000:9000 demand it to increase process and product quality and it is therefore also suggested by process maturity models (Kneuper, 2003; Paulk et al., 1993). In general, configuration management is defined as an "...activity that applies technical and administrative direction over the life cycle of a product, its configuration items, and related product configuration information." (ISO, 2003) While configuration management is an activity, a configuration management system is a socio-technical system consisting of people, organizational rules, tools and their relationships realising configuration management. Configuration management tools support one or more of the CM activities (Estublier, 2000).

The basic element in configuration management is the configuration item. Configuration items are the elements (source code files, requirement documents, models) being under control of the CM (Conradi & Westfechtel, 1998; ISO, 2003). To document all modifications of an item, every modification leads to a new state of the item which is called version (Braun et al., 2006; Zeller, 1997).

According to the purpose of the modification, versions can be characterized as follows: Versions that are created to replace an older version (e. g. for the purpose of development or maintenance) are called historical versions or revisions (Thomas, 2006; Zeller, 1997). In contrast, versions created with the intention to coexist are called variants. Variants are usually created to support different user purposes, e.g. a car component for different user groups (Thompson, 1997). Versions can also be created to support parallel working. Coexisting versions for the purpose of the development in different workspaces are called temporary variants (Braun et al., 2006).

A set of all versions of exactly one configuration item is called a version family. Version graphs are established to illustrate version families and the different kinds of versions existing within a version family (Conradi & Westfechtel, 1998). As the versions of different configuration items are independent from each other (Estublier, 2000), there has to be a container keeping them all together. This container is called the configuration. A configuration is a bundle of versions of configuration items, which represent a complex product (ISO, 2003). It is itself a configuration item. Thus, a new version of a configuration is created, when modifying an item that belongs to the configuration.

3 CONFIGURATION MANAGEMENT WITHIN METHOD ENGINEERING

3.1 Motivation

As the requirements of models change permanently, there is need to adapt appropriate methods the same way (Ralyté et al., 2004). As pointed out in the previous sections, the situational method engineering approach is based on that fact. Thus, methods – when following that approach – are modified constantly and are valid for a certain time interval only.

To ensure the quality of a method, however, the process of modifying methods must be regulated. Otherwise, the individual requirements of each method user can 'blow up' or 'destroy' the method

(Häggmark & Ågerfalk, 2006). To give an example: When user A never uses a specific concept, he or she might remove it from the meta-model. Another user B from within the same project team, however, might need it anymore. This conflict can be solved only by installing a regulated change request process (Dart, 1991). Additionally, all versions of a method should be stored, to enable the recovery of ineffective modifications. To use the knowledge of past decisions for or against a requested change, it is, furthermore, necessary to document the decisions. Thus, the knowledge generated in one project can be used for later decisions within similar projects as well.

To support the above-mentioned criteria, configuration management was adopted to the method-engineering field in the past years (Greiffenberg, 2003; Saeki, 2006; Saeki & Oda, 2005). The motivation for developing a CM for models and methods instead of using existing configuration management systems from the software engineering field are the differences between the configuration items administrated by these systems. In source code oriented CM systems configuration items are text files and differences between those files are recovered by comparing the files line by line. ”Since we use diagram documents..., we should manage the changes on the diagrams, not in the granularity of a line, but of a logical component...“ (Saeki & Oda, 2005), such as a model element or a method construct.

3.2 Administrating Meta-Models

Existing approaches of method CM systems base on model CM and, therefore, administrate meta-models representing methods (Greiffenberg, 2003; Saeki, 2006). Thereby, configuration items represent meta-model elements (method constructs) as well as relationships between those elements. All items are versioned what means that changes on a meta-model element leads to a new version of that element. To recognize different types of meta-model element modifications (e.g. add attribute, remove attribute), it is necessary to administrate the attributes of a meta-model as configuration items as well (Greiffenberg, 2003). Otherwise, from a technical perspective, changing the name of an attribute of a method construct (meta-model element) cannot be distinguished from changing the name of the construct itself.

As meta-models represent methods, CM systems ought to cover all aspects of a method. Unfortunately, it is difficult or even impossible to cover all aspects (product and process description) of a method by a meta-model. In general meta-models represent the concrete syntax (part of the product description) and the process description (Brinkkemper et al., 1998; Saeki, 2003; Strahringer, 1996). Depending on the meta-modelling language, also the graphical representation can be covered by meta-models (Greiffenberg, 2003), but usually the notation is held outside the meta-model as a simple table containing the construct name and its graphical representation (OMG, 2006b).

The semantics of the method constructs describes “…how to identify modelling language constructs in their application domain and what meaning they represent in this environment…” (Pfeiffer & Gehlert, 2005). In existing method descriptions this fact is also held outside the method in terms of natural language statements, either as simple text enclosed to the meta-model (OMG, 2006b), in terms of a domain language dictionary (Greiffenberg, 2003) or as method rational models (Ågerfalk & Wistrand, 2003).

Thus, when using a method CM system just administrating meta-models, information of the methods get lost. Additionally, such a configuration management system cannot automatically detect and react on modifications that lay outside the system. To overcome those deficits, a configuration management system has to administrate meta-models representing the abstract syntax of a method but has to administrate also the graphical representation and the semantics of the method constructs, to cover all aspects of a method.

3.3 Adding model CM to method CM

As modifications of a method affect appropriate models and modelling deficiencies leads to changes of a method, method CM literature usually considers both, configuration management of methods and of models (Greiffenberg, 2003; Saeki, 2006). Thereby, each model is assigned to exactly one method to indicate what method was used for its creation. Model elements point to exactly one method constructs, respectively.

Considering configuration management, models and methods are controlled by a configuration management system. As mentioned in section 2.3, changes on a configuration item do not overwrite older versions, but lead to a new state of the item (Dart, 1991). Thus, when changing a method, the old method version still exists and corresponding models are still consistent with that version. When modelling deficiencies lead to method modifications, however, it might be useful that existing models follow the modifications made on the corresponding method (Saeki, 2006; Saeki & Oda, 2005; Weller & Esswein, 2006).

From the technical viewpoint of a configuration management system, this means creating new versions for all affected models and creating new versions for all model elements whose method constructs have been changed. The created model element versions will refer to the new version of the method construct and the model version to the new method version respectively (Greiffenberg, 2003; Saeki, 2006).

4 METHOD MODIFICATIONS

Considering the effects on existing models, modifying a method can lead to three different results. Firstly, related models may be still consistent to the new method version. Secondly, one or more models are not consistent, but modifying the models (e.g., removing values) to remain consistency is uncritical. Thirdly, the model cannot be adapted automatically to the new method version (critical modification).

In the following sections, we will state critical modifications only. Thus, there might be some uncritical modifications, we do not consider. Modifications on a configuration item are indicated by the creation of a new version of that item. Removing an element means creating a new version of that element as well. In this case the new version will be marked as ‘removed’, but is still hold in the CM system (state-based versioning, see (Conradi & Westfechtel, 1998)).

Before starting the discussion on consequences of method modifications, we have to state some basic assumptions. Firstly, considering the abstract and concrete syntax, we assume that the configuration management system stores the method constructs, their relationships and their graphical representation. Thus, there are configuration items for constructs their properties and their graphical representation (Greiffenberg, 2003). Secondly, we assume that all method modifications are relevant for the user what means that existing models have to be adapted to remain consistent to the modified method. This is important as it is not useful to follow the modifications of a method in any case (Weller & Esswein, 2006).

For formal reasons, the kind of modifications we consider base on the research results of (Ralyté et al., 2004). Thereby, we focus on element changes and structural changes. Naming changes of method elements are trivial, as related models are not affected. Furthermore, we do not consider inter-model changes as we are not interested in effects of method modifications on related methods, but on consequences for instantiated models only. To cover modifications of the concrete syntax and on the semantics as well, we consider the configuration items given by (Greiffenberg, 2003).

4.1 Abstract syntax

4.1.1 Adding Elements

Adding a construct to a method is uncritical in general. If the meta-model prescribes that an existing construct has be connected with at least one instance of the added construct, however, existing models may not consistent to the new method (see Figure 1). This problem only occurs, if the related construct (Relationship-type in Figure 1) is used within the model. Otherwise, the add operation is uncritical.

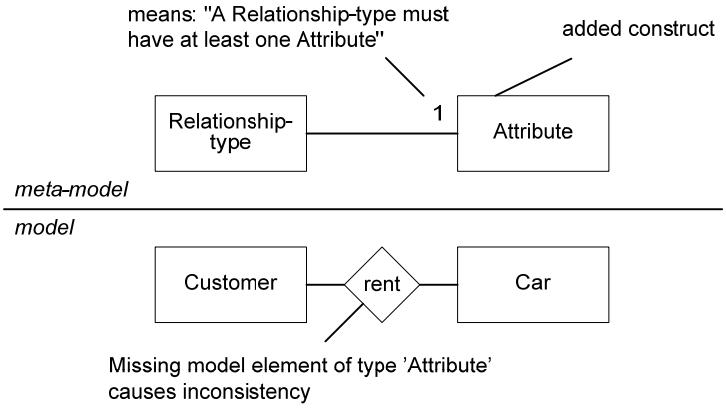

Figure 1. Inconsistency caused by adding method constructs

To solve this conflict, a new instance has to be created in the model for each instance of the related construct (for Relationship-type 'rent' in Figure 1) to ensure model consistency. A modelling tool could create model elements with default-values, but due to the nature of conceptual models, it is strongly recommend involving the modeller and asking him or her to revise the resulting model.

When adding a property to an existing construct, the same problems arise, when the added property is mandatory and when the construct is used within the model. To overcome the problem, a default-property can be added to all model elements related to the construct, but the modeller has to revise the model as well.

4.1.2 Removing elements

Removing a method construct means that this construct cannot be used in the related models anymore. Therefore, all instances of the construct must be deleted from the dependent models to ensure model consistency (Greiffenberg, 2003; Schuette & Rotthowe, 1998).[2] While this causes no problems in general, removing a method construct can also lead to inconsistencies when using generalisation within the meta-model.

[2] In practice, it might be interesting to indicate all instances of a construct as deprecated instead of removing them. While this leads to syntactical inconsistent models regarding the meta-model, we do not follow that idea.

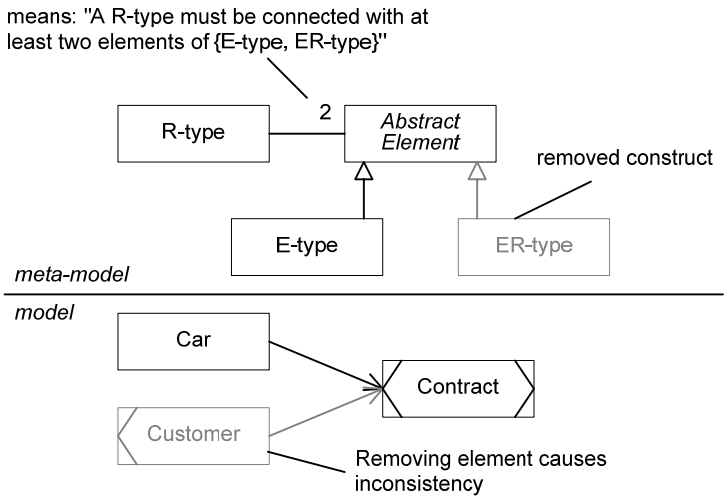

Figure 2. Inconsistency caused by removing method constructs

The upper part of Figure 2 shows a meta-model after removing the construct ER-type (gray). The model (lower part of Figure 2), however, becomes inconsistent with the method, because 'Contract' is related to one instance of Abstract Element only. This problem only occurs when the removed construct is used within the model and there is a relationship to the construct specifying a min-cardinality. The problem cannot be solved automatically, except by removing all instances of the meta-construct (abstract element in our example) or by adding necessary instances with default-values (see problems discussed in section 4.1.1 also). We demand, however, just to inform the user to correct the erroneous model.

4.1.3 Modifying elements

Modifying the properties of a construct means adding, removing or modifying the configuration item of the property (Greiffenberg, 2003). Thus, the construct itself is not affected in the configuration management system. As we already discussed add and remove operations, we focus on property modifications below.

When changing a property, we have to consider two critical modifications. First, the value range of the property might be changed. This has a strong effect on related models, as existing values may not be transferred to the new range. Thus, when changing the range from TEXT to BINARY existing values have to be removed. There might be migration strategies for that problem (e.g. "false" to false), but in general the modeller has to revise all affected values. This problem only occurs when the construct of the modified property is used within the model.

Another critical modification is changing the cardinality of a property. If the min-cardinality (specifying how many values have to exist at least) raises, there is need to add default-values to a model element (see again section 4.1.1) to remain consistency. If the max-cardinality (maximum number of values) lowers, values have to be removed. Because adding default-values or removing values change the content of a model, this has to be revised by the modeller. The problem only occurs when the construct of the modified property is used within the model.

4.1.4 Relationships between constructs

A Relationship (association, aggregation) between constructs can be represented by properties of the constructs pointing to the other construct. Thus, modifying a relationship means adding, removing or changing a property (see previous sections). A special kind of relationship is generalisation. Adding a generalisation between two existing constructs means adding all properties of the super-construct to

the sub-constructs (see problems described in section 4.1.1). Removing a generalisation means removing all super-construct properties from the sub-construct (section 4.1.2).

4.2 Concrete Syntax

4.2.1 Removing graphical representation of a method construct

Removing a graphical representation of a construct means removing all representations of model elements related to that construct representation. If there is more than one representation for the construct, the model element representation can be replaced by another representation of the construct (see next section). E.g. when removing the circle representation of the UML construct “interface”, the rectangle representation can be used instead. The modeller, however, should revise the result of such an operation.

As illustrated in Figure 3, removing the representation of a model element can lead to ‘broken’ models. Thus, graphical connections between model elements have to be removed and rearranged by the modeller. The problem occurs only, if the representation is used within the model. As removing a construct also means removing its graphical representation, this problem also occurs when a construct that is used within the model is removed.

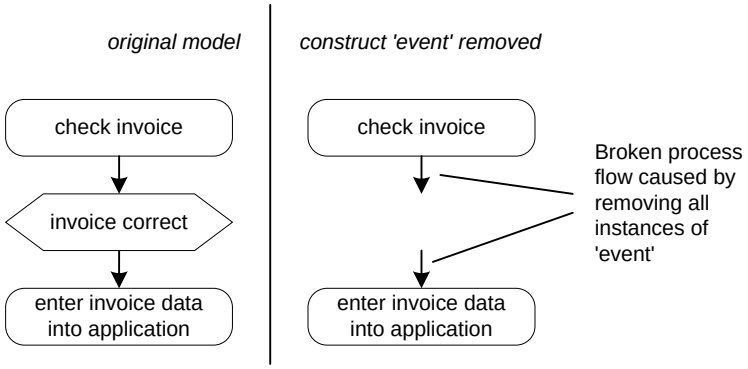

Figure 3. Visual problems caused by removing (graphical representation of) constructs

4.2.2 Changing graphical representation of construct

According to the purpose of modelling, it can be necessary to change the graphical representation of a construct (Becker et al., 2004). In this case, all instances of the construct have to be adapted according to the new graphic. Thereby, a different size or shape of the new representation can cause visual discrepancies that negatively effect the quality of resulting models (Schuette & Rotthowe, 1998). If the size of the graphic increased, other model elements might overlap with the new graphic. To avoid such side effects, we recommend adapting the size of the new model element’s representation according to the size of the old representation. Changing the representation of a construct is critical only when the modified representation is used within the model.

4.3 Semantics

Modifying the semantics of a method construct is indicated by the creation of a new version of the configuration item containing the construct description. As the semantics of methods in conceptual modelling cannot be formalized, all model elements of the modified construct have to be revised by the modeller. This problem only arises when the construct is used within the model.

5 CONCLUSIONS AND FURTHER RESEARCH

This paper has shown the problems arising when modifying a method in a configuration management environment. We discussed consequences of method modifications for existing models in the case it is necessary that the models follow the modifications of their related methods. Thereby, we have shown that these problems do not occur in any case, but under certain conditions. Table 4 summarizes our findings.

No.	Modification	Condition	Solution	Effects
1a	Add construct	Construct mandatory & used within model	Create default-model element & revise by user	
1b	Add construct property	Property mandatory & construct used within model	Create default-value & revise by user	
2	Remove construct	Construct is sub-construct of a generalisation & the super-construct is mandatory & construct is used within model	Remove model elements related to that construct & inform user to revise model	4
3a	Change value range of property	Automatic migration strategy not feasible	Inform user to revise model	
3b	Raise min-cardinality of property	Related construct used within the model	Add default values or Add relationship (when value range is construct)	1a, 1b
3c	Lower max-cardinality of property	Related construct used within the model	Remove values or relationships	2
4	Remove graphical representation of construct	Representation of construct used within the model	Inform user to revise model	
5	Change graphical representation of construct	Representation of construct used within the model & new graphic has different shape	Adjust size of new graphic to size of the old one & revise by user	
6	Change construct semantics	Construct used within the model	Inform user to revise model	

Table 4. Consequences of method modifications within a method CM system

Our findings can be used to forecast the consequences of method modifications. Within a tool-supported method engineering process, this knowledge is important for developing tools that can handle method modifications and inform the user about problems before the new method version will be published or handle model migrations after such a modification. This will help developing CAME tools with integrated CM support such as (Saeki, 2006) or (Cubetto, 2006).

Currently, we evaluate existing CAME tools according to their possibilities to administrate method versions and to handle modification problems as described above. Depending on the results of that evaluation, we will possibly start designing and implementing an appropriate tool or enhance an existing one. Our future research will focus the problem of merging method modifications made within different projects to build up or to enhance a company wide method, reflecting the methodology knowledge of the company. Thereby, we will use the positive findings made with configuration management of reference models (Braun et al., 2006). Additionally, we will discuss the advantages of using models and methods from the information systems discipline within managerial economics for business process improvement projects.

References

Ågerfalk, P. J. and Wistrand, K. (2003) Systems development method rationale: A conceptual framework for analysis. In *5th International Conference on Enterprise Information Systems (ECEIS '03)*, pp 185-190.

Becker, J., Delfmann, P., Dreiling, A., Knackstedt, R. and Kuropka, S. (2004) Configurative process modeling: Outlining an approach to increased business process model usability. *Innovations Through Information Technology*,

Braun, R., Esswein, W., Gehlert, A. and Weller, J. (2006) Configuration management for reference models. *Reference Modeling for Business Systems Analysis*, IDEA Group, Hershey.

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. (1998) Assembly techniques for method engineering. In *Proceedings of the Conference on advanced information systems engineering (CAiSE)*, pp 381-400.

Clark, T., Evans, A. and Kent, S. (2002) Engineering modelling languages: A precise metamodelling approach. In *Fundamental Approaches to Software Engineering* (Weber, H., Ed), pp 159-1732, Grenoble, France.

Conradi, R. and Westfechtel, B. (1998) Version models for software configuration management. *ACM Computing Surveys* 30 (2), 232-282.

Cubetto (2006) Cubetto toolset. http://wise.wiwi.tu-dresden.de/cubetto.

Dart, S. (1991) Concepts in configuration management systems. In *Proceedings of the 3rd international workshop on Software configuration management*, pp 1-18, ACM Press, Trondheim, Norway.

Dieste, O., Juristo, N., Moreno, A. M., Pazos, J. and Sierra, A. (2000) Conceptual modelling in software engineering and knowledge engineering: Concepts, techniques and trends. *Handbook of Software Engineering and Knowledge Engineering*, World Scientific Publishing Company, pp 733-766.

Estublier, J. (2000) Software configuration management: A roadmap. In *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*, pp 279-289, ACM Press, Limerick, Ireland.

Estublier, J., Leblang, D., Clemm, G., Conradi, R., Tichy, W., Hoek, A. V. D. and Wiborg-Weber, D. (2002) Impact of the research community on the field of software configuration management: Summary of an impact project report. *SIGSOFT Software Engineering Notes* 27 (5), 31-39.

Evermann, J. and Wand, Y. (2005) Ontology based object-oriented domain modelling: Fundamental concepts. *Requirements Engineering* 10 (2), 146-160.

Fettke, P. and Loos, P. (2002) Klassifikation von informationsmodellen - nutzenpotenziale, methode und anwendung am beispiel von referenzmodellen. Johannes Gutenberg-Universität Mainz, Lehrstuhl für Wirtschaftsinformatik und BWL, Mainz.

Fitzgerald, B. (1997) The use of systems development methodologies in practice: A field study. *Information Systems Journal* 7 (3), 201-212.

Floyd, C. (2001) Das mögliche ermöglichen: Zur praxis der realitätskonstruktion am beispiel der softwareentwicklung. *Konstruktivismus und Kognitionswissenschaft: Kulturelle Wurzeln und Ergebnisse*, 115-134.

Frank, U. (1999) Conceptual modelling as the core of the information systems discipline - perspectives and epistemological challenges. In *Proceedings of the Fifth America's Conference on Information Systems (AMCIS 99)* (Goodhue, D., Ed), pp 695-697, Milwaukee.

Greiffenberg, S. (2003) *Methodenentwicklung in wirtschaft und verwaltung*. Dr. Kovac, Hamburg.

Häggmark, M. and Ågerfalk, P. J. (2006) Why software engineers do not keep to the principle of separating business logic from display: A method rationale analysis. *Lecture Notes in Computer Science* 4001/2006, 399-413.

Harmsen, F., Brinkkemper, S. and Oei, J. L. H. (1994) Situational method engineering for information system project approaches. In *Methods and associated tools for the information systems life*

cycle, Proceedings of the IFIP Working Conference (Olle, T. W., Ed), pp 169-194, Elsevier Science B.V. (North-Holland).
Iso (2003) *Quality management systems: Guidelines for configuration management (iso 10007:2003).* Beuth Verlag GmbH, Berlin.
Jeusfeld, M. A., Jarke, M., Nissen, H. W. and Staudt, M. (1998) Conceptbase: Managing conceptual models about information systems. *Handbook on Architectures of Information Systems*, Springer, pp 265-285.
Keller, G., Kelly, S., Lyytinen, K. and Rossi, M. (1996) Metaedit+ a fully configurable multi-user and multi-tool case and came environment. *Lecture Notes in Computer Science* 1080, 1-21.
Kneuper, R. (2003) *Cmmi: Verbesserung von softwareprozessen mit capability maturity model integration.* dpunkt.
Omg (2006a) Business process modeling notation specification (bpmn 1.0). www.bpmn.org.
Omg (2006b) Unified modeling language specification (uml 2.0). www.uml.org.
Paulk, M. C., Curtis, B., Chrissis, M. B. and Weber, C. V. (1993) Capability maturity model, version 1.1. *IEEE Software* 10 (4), 18-27.
Pfeiffer, D. and Gehlert, A. (2005) A framework for comparing conceptual models.
Ralyté, J., Rolland, C. and Deneckère, R. (2004) Towards a meta-tool for change-centric method engineering: A typology of generic operators. *Lecture Notes in Computer Science* 3084, 202-218.
Rosemann, M., Schwegmann, A. and Delfmann, P. (2005) Vorbereitung der prozessmodellierung. *Prozessmanagement: Ein Leitfaden zur prozessorientierten Organisationsgestaltung*, 45-103.
Saeki, M. (2003) Came : The first step to automated method engineering. In *OOPSLA 2003: Workshop on Process Engineering for Object-Oriented and Component-Based Development*, pp 7-18.
Saeki, M. (2006) Configuration management in a method engineering context. *Lecture Notes in Computer Science* (4001), 384-398.
Saeki, M. and Oda, T. (2005) A conceptual model of version control in method engineering environment. In *CAiSE´05 Forum*.
Scheer, A.-W. (2000) *Aris - business process modeling.* Springer-Verlag New York, Inc., Secaucus, NJ, USA.
Schuette, R. and Rotthowe, T. (1998) The guidelines of modeling: An approach to enhance the quality in information models. *Lecture Notes in Computer Science* 1507, 240-254.
Strahringer, S. (1996) *Metamodellierung als instrument des methodenvergleichs: Eine evaluierung am beispiel objektorientierter analysemethoden.* Shaker, Aachen.
Thomas, O. (2006) Version management for reference models: Design and implementation. In *Multikonferenz Wirtschaftsinformatik 2006 (MKWI '06)*.
Thompson, S. M. (1997) Configuration management - keeping it all together. *BT Technology Journal* 15 (3), 48-60.
Tolvanen, J.-P. (1998) Incremental method engineering with modeling tools: Theoretical principles and empirical evidence. University of Jyväskylä.
Wand, Y. and Weber, R. (2002) Research commentary: Information systems and conceptual modeling--a research agenda. *Information Systems Research* 13 (4), 363-377.
Weller, J. and Esswein, W. (2006) Consequences of meta-model modifications within model configuration management. In *2nd Workshop on Meta-Modelling (WoMM)*.
Zeller, A. (1997) Configuration management with version sets: A unified software versioning model and its applications. Braunschweig Technical University.