Reuse Mechanisms in Situational Method Engineering

Jörg Becker, Christian Janiesch, Daniel Pfeiffer
European Research Center for Information Systems (ERCIS)
University of Münster, Leonardo-Campus 3, 48149 Münster, Germany
{becker,janiesch,pfeiffer}@ercis.de,
WWW home page: http://www.ercis.de

Abstract. Methods describe systematic procedures to overcome problems. It has been widely acknowledged that methods have to be adapted to the context of their application in order to maximize their impact. Since the original proposal of situational method engineering, numerous approaches have been introduced to tackle this problem. In order to efficiently design situation specific methods it is necessary to reuse existing knowledge. Reuse mechanisms have emerged in different research areas that can be transferred to method engineering. The objective of this paper is to identify relevant reuse mechanisms for method engineering and to review the literature for their usage. Thereof, we derive suggestions for the improvement of existing method engineering approaches and the design of new ones.

1 Introduction

Methods describe systematic procedures to overcome problems. Problems can be characterized as the discrepancy between an as-is and a to-be situation. It is widely accepted that a universal method which can be used without modification in all situations is not feasible [1-6]. Rather, appropriate methods for problem solving must be chosen, adapted, or designed depending on the specific characteristics of a situation, such as qualification, number of employees, or available time. In the method engineering community, terms like domain specific method engineering [7, 8] or situational method engineering [9-11] have been used to voice this special circumstance.

To design a method that meets the specific needs of a situation is very time-consuming and costly. Hence, it is not efficient to build situation specific methods from scratch. Rather, it makes sense to reuse existing knowledge to reduce the cost of construction and evaluation. For this purpose reuse approaches such as components, reference models, or patterns have evolved. These approaches have been successfully applied in diverse contexts. The underlying principles of these

Please use the following format when citing this chapter:

Becker, J., Janiesch, C., Pfeiffer, D., 2007, in IFIP International Federation for Information Processing, Volume 244, Situational Method Engineering: Fundamentals and Experiences, eds. Ralyté, J., Brinkkemper, S., Henderson-Sellers B., (Boston Springer), pp. 79-93.

approaches are so called reuse mechanisms. The aim of this paper is to explicate these mechanisms from a method engineering perspective and to propose directions for the improvement of existing and the design of new adaptable methods.

Accordingly, the paper is structured as follows: Following this motivation, a classification of reuse and adaptation approaches and their mechanisms is introduced to provide the basis for the analysis. In Section 3, approaches of situational method engineering are reviewed concerning their exploitation of reuse mechanisms. Section 4 includes a discussion of the literature review as well as a classification of the mechanisms. It concludes the analysis with an outlook and proposal for possible future directions of research on situational method adaptation. The paper closes with a short summary of the main results.

2 Reuse Approaches and Mechanisms

2.1 Reuse approaches

Reuse means to apply the experiences of a former projects to solve an actual problem [12]. This implies that an existing knowledge base is utilized to avoid starting from scratch. For the reuse of knowledge different approaches have been developed that can be found in a similar form in software engineering, conceptual modeling, or method engineering:

Patterns (P): A pattern defines a template to solve a commonly occurring problem [13]. It contains a problem description and the abstract structure of a possible solution. It is necessary to specialize the pattern and to fill the abstract solution with additional information in order to meet the specific conditions of the actual case. A pattern can be applied, when the issue at hand maps with the general problem specification in the pattern. Examples are analysis patterns which contain the knowledge on how to appropriately represent a certain fact in systems analysis or requirements engineering [14]. Patterns are also used to guide model-based design of software [15].

Components (CO): Components are independent items that can be aggregated to form a new artifact [16, 17]. They have been derived from recurrently occurring elements or they are formulated to reach compliance with a certain standard. Components provide a partial solution to a defined problem. Compared with patterns they are more concrete as they can be used without modification. They act as building blocks that can be assembled based on certain rules to achieve an intended solution. Process building blocks are an example for model components [18]. It can be argued that components may also be configured or specialized before or after aggregation. For the sake of selectiveness of mechanisms we have not included this in the overview.

Modules (M): Modules or generic packages are abstract objects which have to be instantiated to be of concrete use. The idea originates from the need in software engineering to know data types before run-time [19]. By implementing a non-type specific package, it can be reused, i.e. instantiated, for various data types. The concept of generic packages [20] carries the idea on to offer unique data structures

that can be reused for various data types. Lately, the idea to instantiate reference models emerged [26].

Reference model (RM): A reference model is a robust yet flexible model which comprises universal information that suits more than one situation [22]. Reference models contain information which is relevant for a class of modeling scenarios. The information within these models is applicable to several organizations in different domains. Reference models can be used as-is, but commonly, they are adapted to the specific conditions of a situation. They represent common or best practices and often offer a normative suggestion to solve a certain problem. Compared with components, reference models do not just provide a small part of a solution but they are more comprehensive. Similar to patterns reference models normally also have to be adapted to meet the specific conditions of an organization. Examples for reference models are the Y-CIM model [23] or the Retail-H [24]. Reference models are used by enterprise systems vendors to specify the functionality of their systems [21].

These reuse approaches are rather complementary then competing. For example a reference model can be split up into components and later be aggregated. A reference model can also be part of a pattern. Reference models imply a top down approach. Components help to construct a solution bottom up. Patterns can, depending on their granularity, address both ways. In the next section we want to analyze what basic mechanisms are employed by the reuse approaches.

2.3 Reuse Mechanisms

All reuse approaches are based on a common set of reuse mechanisms [25, 26], which enable their definition and application.

Analogy Construction (AC): An analogy implies the transfer of information from one subject to another. This mechanism is very flexible as it can be drawn from any aspect of an artifact. It is for example used by patterns (P/AC). Patterns employ this mechanism in order to be applicable in domains they were not specifically constructed for. The application of a pattern requires a conclusion by analogy to establish a fit between the problem description in the pattern and the actual situation. Also, reference models or their parts can be the basis of an analogy construction (RM/AC), e.g. as proposed by the ebXML initiative [27]. By the annotation of relevant parts of the reference model, its elements can be reused in different situations.

Aggregation (A): Aggregation assembles independent parts to form a composite. This mechanism is applied by components (CO/A) (cf. e.g. [16, 28]). Interface descriptions of model components offer information on the possibility to combine or integrate the different components and their general compatibility [29]. One might also argue that there are also reference models which support aggregation. These models are not available as monolithic blocks but rather as independent elements that can be assembled [25]. This is rather a special case which does not conform well with our definition of a reference model as one universal model. Hence, we do not consider aggregation a relevant mechanism for reference models.

Configuration (C): Configuration means to modify certain elements of an artifact based on predefined rules that refer to specific project situations. Reference models

can be designed as configurable artifacts (RM/C). They are provided with explicit configuration points, which specify model variants regarding purpose-specific characteristics [21, 26]. Based on the specific values assigned to configuration parameters a reference model is projected into a company-specific model. Model elements are removed or modified depending on the parameters. The actual procedure of model projection can be automated based on the prior annotation of configuration parameters to the model.

Specialization (S): Through specialization a particular artifact is derived from a more general artifact by adopting, extending and/or partially modifying the more general one [30]. Reference models can support specialization (RM/S). These reference models have a higher level of abstraction than their organization-specific counterparts. They offer only a relatively low number of model elements. Patterns also use the specialization mechanism in order to transform the solution structure into a concrete solution (P/S).

Instantiation (I): Instantiation selects a specific value or object from a predefined domain with multiple possible occurrences. Instantiation can be applied on modules (M/I). In order to prepare them for this mechanism they must be annotated with placeholders [19, 20]. The placeholders are added during the construction of the module. When a specific module is created, the placeholders are filled with valid occurrences with respect to the particular circumstances. Depending on the properties of the domain, numeric or alphanumeric attributes, distinct elements, or even composed clusters can be defined as placeholders.

Table 1 maps the different reuse mechanisms to their corresponding reuse approaches.

Table 1. Mapping of Identified Mechanisms to Approaches

Mechanism / Approach	Pattern	Component	Module	Reference Model
Analogy Construction	P/AC			RM/AC
Aggregation		CO/A		
Configuration				RM/C
Specialization	P/S			RM/S
Instantiation			M/I	

3 Utilization of Reuse Mechanisms in (Situational) Method Engineering

Since the establishment of the method engineering as an own research field within the IS discipline in the 1990's many suggestions for the design of methods and the combination of their components have been published. In this section the current state-of-the-art of (situational) method engineering research is examined and compared.

In IS literature a method is not considered as a single monolithic block but rather consisting of a set of fragments [e.g. 11, 31, 32, 33], also called chunks [e.g. 34] or components [e.g. 35, 36]. These fragments can have a very different granularity and

they can describe the product (what is created) as well as the process aspect (how is it created) of a method. The fragments can comprise a single activity or construct but they can also contain a complete method. Hence, a method engineering project can start with a set of atomic method fragments which must be assembled as well as an existing method which has to be modified [37]. Method engineering research has mainly focused on the first strategy so far [38]. Contrary to that, reuse mechanisms primarily focus on an existing artifact and only additionally consider its design by the aggregation of predefined components. Both strategies are viable approaches to (situational) method engineering as the corresponding mechanisms can be widely found in the method engineering literature as the following review shows (cf. Table 2).

Table 2. Overview of Reuse Mechanisms in Method Engineering Approaches

No.	Reference	Reuse Mechanism	Denotation of the Reuse Mechanism	Objects of the Reuse Mechanism
1	Baskerville and Stage [39]	Aggregation	Accommodation	Method Fragments
		Specialization	Accommodation	Method Fragments
2	Bajec et al. [38]	Configuration	Process Configuration	Base Method, Configuration Rules, Project Characteristics
3	Becker et al. [40]	Configuration	Method Configuration	Configurable Method
4	Brinkkemper et al. [10, 11, 31]	Aggregation	Method Assembly	Method Fragments
5	Cameron [41]	Aggregation	Tailoring	Work Product Descriptions, Work Breakdown Structures
6	Fitzgerald [4]	Aggregation	Method Tailoring	Original Formalized Methodologies
7	Greiffenberg [42]	Aggregation	Creation of Meta Model	Concepts
		Specialization	Choice of Reference Meta Model Scope	Reference Meta Models
		Analogy Construction	Creation of Meta Model	Typing Patterns
8	Gupta and Prakash [16]	Aggregation	Method Assembly	Method Components
9	Henninger [43]	Specialization	Refinement and Tailoring	Software Development Resources
10	Karlsson et al. [36, 44, 45]	Configuration	Configuration Framework	Base Method, Configuration Package, Configuration Template
11	Kumar and Welke [9]	Aggregation	Methodology Engineering	Methodology Components
12	Leppänen [46]	Aggregation	Method Engineering Methodical Skeleton	Contextual Method Components
13	Nuseibeh [47]	Aggregation Instantiation	Template Reuse Instantiation	Viewpoint Templates Viewpoint Templates

No.	Reference	Reuse Mechanism	Denotation of the Reuse Mechanism	Objects of the Reuse Mechanism
		Specialization	Inheritance	Super Templates
14	Odell [48]	Specialization	Single Framework Modeling	Kernel Meta Model
15	Patel et al. [49]	Aggregation	Selection and Assembly	Method Fragments
		Specialization	Method Tailoring	Method Fragments
16	Punter and Lemmen [32]	Aggregation	Assembly	Method Fragments
17	Ralyté et al. [30, 34, 37, 50]	Aggregation	Assembly-based strategy	Method Chunks
		Specialization	Extension-based strategy, Paradigm-based strategy	Method Chunks, Meta Models
		Analogy Construction	Paradigm-based strategy	Meta Models
18	Rossi et al. [51, 52]	Aggregation	Method Construction	Method Components
		Specialization	Method Refinement	Meta Models
19	Saeki and Wenyin [33]	Aggregation	Method Integration	Meta Models
20	van Offenbeek and Koopman [53]	Specialization/ Analogy Construction	Fit	Scenarios

Harmsen [11], Brinkemper [10] and Brinkkemper et al. [31] focus in their situational method engineering approach in particular on the recombination of method fragments and, thus, are using the mechanism of aggregation. They describe rules in the context of the aggregation to guide the assembly of method fragments. Comparable approaches were for example published by Fitzgerald et al. [4], Gupta and Prakash [16], Punter and Lemmen [32], and Saeki and Wenyin [33]. Kumar and Welke [9] handle methodology components similarly but also stress the disassembly of old methods prior to the assembly of new methods. Cameron [41] puts so called work products together and chooses their temporal order to define a specific development process. Baskerville and Stage [39] as well as Patel et al. [49] emphasize the need to adapt a method after the aggregation by means of deletion, addition and/or modification. This so called method accommodation is considered an application of the specialization mechanism.

Greiffenberg [42] has published a comprehensive approach for the development of modeling languages. Greiffenberg specifies a meta modeling language, a reference meta model, a set of meta modeling patterns as well as a process model for method engineering. Following Greiffenberg, the construction of a meta models is based on concepts. For this purpose the mechanism of aggregation is used. Furthermore, Greiffenberg includes the mechanism of specialization for selecting from the reference model. Analogy construction is taken into consideration by the application of meta modeling patterns.

Henninger et al. [43] and van Offenbeek and Koopman [53] base their methodology on existing scenarios or available resources. They gather contextual factors such as risk to guide the adaptation process of a method. The procedure

results ultimately in a refined method that is a specialized version of the original or that is analog to the original. The refinement process is strongly depending on the fit of original model to the specific problem. A configuration in terms of specific adaptation points or parameters is not in focus.

Karlsson and Ågerfalk [45], Karlsson [44], and Karlsson and Wistrand [36] are one of the few authors in method engineering who directly address the mechanism of configuration. Adaptations of methods are performed by the use of configuration packages which rest upon a base method. To manage complex situations with a number of characteristics, configuration packages are combined to configuration templates. Based on the characteristics of a project, an acceptable configuration template is chosen and applied on the base method. Thus, the base method is configured according to the project needs. The configuration of the method focuses only the procedure model. Due to this fact, the modeling language and the resulting products are only indirectly taken into consideration. A comparable approach that also focuses the process part of the method is suggested by Bajec et al. [38]. Becker et al. [40] transfer the mechanisms of configurative reference modeling to the domain of method engineering.

Leppänen [46] also makes use of method components that can be aggregated to form situation-specific methods. He, however, focuses on forming an ontology that assists the selection and combination, i.e. integration, of these components. He provides comprehensive interfaces that explicate the compatibility of method components. However, his ontology is not intended to be the basis for any further configuration and, thus, only provides a means to perform method aggregation.

The approach of Nuseibeh [47] focuses on the multi-perspectivity of software development with the ViewPoint Framework. Due to the abstraction of viewpoints to viewpoint templates, this can be understood as a method engineering approach. New methods can be designed by the combination of viewpoint templates. On the basis of an abstract super template, specialization (inheritance) results in a more specific sub template. By applying the instantiation mechanisms on a viewpoint template, a viewpoint is created.

Odell [48] suggests a basic vocabulary for describing modeling languages, which is called kernel meta model. With the aid of the kernel meta model, which is based on the mechanism of specialization, it is possible to derive specific concepts of meta models. For example, the concept *relation* from the kernel meta model can be specialized as *is super type of.*

Ralyté and Rolland [37] provide a generic process model for situational method engineering. With a map notation they describe different strategies to construct a method that meets the contingencies of a project situation. The *assembly-based strategy* reuses method chunks from a repository and compiles them by applying the aggregation mechanism [30]. The *extension-based approach* uses the specialization mechanism on an existing method and employs patterns to provide novel additions to it [34]. The *paradigm-based approach* takes a meta-model that belongs to a certain theoretical framework as starting point. By analogy construction and specialization the meta model is adapted to the specific needs. Mirbel and Ralyté [50] include the a detailed refinement of a project specific method by each project member. They describe the aggregation of method chunks for the development of project specific methods. For adapting the method with respect to the necessity of every project

member, they suggest the adaptation of the procedure (roadmap) of the project specific method by the mechanism of specialization; e.g. the user is able to choose between a prosaic way and use case diagrams to document a requirements analysis depending on his expertise.

Tolvanen [51] and Rossi et al. [52] are highlighting the iterative, incremental aspects of method engineering. They assume that due to the inadequate acknowledgement of the application domain at the beginning of the method engineering project, only a part of the language specification is possible. Thus, an iterative process for evaluation and refinement of the method is necessary to reach an adequate description level for the method. This includes adaptation (specialization) and addition of missed constructs (aggregation). Becker et al. [54] also argue that the feedback of situational adaptations has to be considered in the evolutionary development of a method.

In the next section the results of this literature review are analyzed and implications for method engineering research are derived.

4 Discussion of the Literature Review

Reuse mechanisms can be classified according to different dimensions. One possibility of segmentation is to take the costs of preparation, i.e. the engineering of the situational method, as the first dimension and the cost of utilization, i.e. the reciprocal value of the degree of guidance in adaptation, as second dimension [25, 55].

The costs of preparation depend on how much effort is necessary before a certain mechanism can be used. To be able to apply the mechanisms of *configuration*, rules must be defined and the model elements must be annotated according to the rules. This process is very time-consuming. It is necessary to define the domains of valid values to be able to apply an *instantiation* of the corresponding placeholders. *Aggregation* can specify constraints which restrain the possible combinations of components but such rules are not obligatory. *Specialization* can exclude certain sorts of modification and allows the general adaptation of models. An *analogy construction* can always be applied and does not require any preparation.

The costs of utilization vary on how much the modeler is instructed when a certain mechanism is employed. The guidance in the case of *configuration* is high and consequently, the costs of utilization are low. When the parameters are filled with values the model can be automatically configured. Interactions with the user are only necessary to resolve possible conflicts. *Instantiation* specifies the domain of possible values but gives no hints what values to choose in a certain situation. This leads to higher efforts and thus, higher costs of utilization. The guidance of *aggregations* and *specialization* depends on whether any restrictions have been specified. Aggregation can be directed by interface definitions and specialization can be supported by detailed descriptions of the required actions. *Analogy construction* offers no instructions at all on how to proceed and, hence, it results in the highest cost of utilization. In Fig. 1 the different mechanisms are arranged in a portfolio.

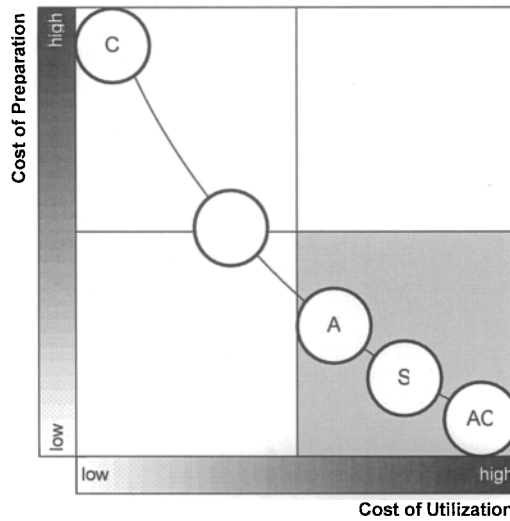

Fig. 1. Cost of Preparation and Cost of Utilization of Reuse Mechanisms [25, 56]

A different possibility to assess reuse mechanisms is by the complexity of the reuse situation and the rate of repetition of the corresponding conditions. In Fig. 2 the different mechanisms are juxtaposed.

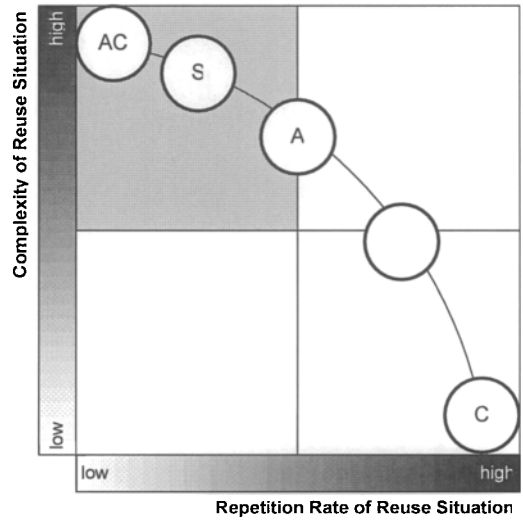

Fig. 2. Applicability of Mechanisms Concerning the Complexity of the Situation and the Repetition Rate of Reuse

The complexity of the reuse situation describes how many contingency factors influence the suitability of a solution. *Configuration* requires anticipating all

circumstances that may affect the result. As all these factors can be mutually dependent, the number of necessary configuration rules increases strongly with the complexity of the potential reuse situations. Therefore, configuration is only applicable in situations with a fair degree of complexity. *Instantiation* makes it necessary to define an instantiation domain. Therefore, knowledge about the relevant aspects that influences the corresponding instantiation values is needed. However, this knowledge is only available in medium complex potential situations. Aggregation and specialization can also be used in situations with a higher complexity as they provide flexible means to adapt the solution to the specific needs of a situation. *Aggregation* takes predefined artifacts to assemble a solution; with *specialization* an existing solution is modified with loose guidance. *Analogy construction* provides the highest degree of freedom to shape the result and can, therefore, also meet the requirements of highly complex situations.

The repetition rate measures whether the specific conditions of a reuse situation are unique or rather regularly reoccurring. *Configuration* is based on rules that fit to a number of predefined situations. The efforts to construct these configurations rules only pay off when the repetition rate of the situation is high and, thus, the rules can be applied in more than one situation. *Instantiation* also leads to relatively high cost of preparation but is less strongly coupled with a set of specific situations. *Aggregation* and *specialization* can be used in a much wider variety of situations. Hence, both of them do not depend on a high repetition rate of each single reuse situation. As *analogy construction* does not induce high costs of preparation but can help to construct a solution that meets the specific needs of a project, it can also be applied when the reuse situation is unique. On the downside, its repeatability is very low so that it cannot be ensured that results can be reproduced.

Table 3 gives an overview about the utilization of the reuse mechanisms in situational method engineering literature. Every approach analyzed employs one or more mechanisms. Thus, the sum of mechanism utilization does not add up with the sum of the reviewed method engineering approaches above.

Table 3. Utilization of Reuse Mechanisms in Method Engineering

Reuse Mechanism	Number of Utilizations	Percentage
Analogy Construction	3 / 20	15 %
Aggregation	14 / 20	70 %
Configuration	3 / 20	15 %
Specialization	9 / 20	45 %
Instantiation	1 / 20	5 %

The analysis shows that especially aggregation and specialization are included in many approaches. Often both approaches appear in conjunction (6 out of 20, i.e. 30 %). Analogy construction is related to specialization as a specialization with a very high degree of freedom can lead to similar results as an analogy construction [26]. Furthermore, analogy construction is rarely employed; often it is only used in combination with specialization [37, 42, 53].

It can also be deducted that the configuration and instantiation of methods has not been sufficiently addressed yet. While instantiation was used in only one

approach in conjunction with two others, configuration only appeared thrice. Furthermore, when configuration was used, it was always the only mechanism employed in the approach.

Hence, so far method engineers have focused mainly on mechanisms with low costs of preparation but rather high costs of utilization (cf. gray box in Fig. 1). This means that the costs of designing method engineering approaches are comparably low, but only little guidance is given on how to construct a situation specific method. Consequently, method engineering research has mainly focused on complex and singular situations where extensive preparations are not feasible (cf. gray box in Fig. 2). Hence, future method engineering research should also look at less complex but repetitive situations. With domain specific methods [52, 57] the first results from this research stream can be observed.

Ultimately, however, the goal has to be to engineer well-balanced methods, which utilize the most applicable reuse mechanisms for their intended reuse scenario. Since there are as many scenarios, i.e. situations, as there are possible ways to adapt a method no universal answer can be derived from this analysis. However, it is to assume that certain general assumptions hold true:

- If a (part of the) method is used and adapted more often than others and this adaptation can be explicated beforehand, it should be a configurable method.
- If a (part of the) method's general path of adaptation can be foreseen but is not as clearly laid out as with configuration, it should be a method that can be adapted by instantiation.
- If a (part of the) method is very extensive and used in heterogeneous environments so that only a small amount of the originally intended method is used, but this part is used as is, it should be a component-based method.
- If a (part of the) method is used and adapted in a diverse way and only limited adaptations have to take place to create variants, specialization should be used.
- If a (part of the) method is used only seldom and adapted in a very diverse way, the method should be adapted by analogy construction.

Naturally, a method engineering approach can combine multiple of these mechanisms to form an adaptable situational method. For example, a situational method can be aggregated from adaptable components. Some of the components can be configurable, some might only be instantiable or specializable. Some might not even exist and have to be engineered by analogy construction on demand.

Preliminary quantitative analysis hints at the fact that making only a limited but integral part of a method configurable or instantiable, eases the adaptation of the overall method considerably. Cf. for evidence concerning model component configuration of business documents [58]. In this case it was observed that certain central components of a method have been used more often than others. Making only them configurable already allows the method engineering to reduce the adaptation effort of the overall method considerably without reengineering the whole method. This observation goes along with the Pareto principle observed in other areas.

5 Conclusion

It has been widely acknowledged that methods have to be adapted to the context of their application in order to maximize their impact. Situational method engineering is considered to be a reasonable approach to support this adaptation process to reuse existing knowledge.

As preliminary analysis has shown that there are at least three commonly accepted approaches to knowledge reuse that are relevant to method engineering: patterns, components, and reference models. Each of these approaches uses one or more of five distinct reuse mechanisms which facilitate employing existing knowledge to a new situation. A subsequent literature review revealed that only two of these mechanisms are used frequently within situational method engineering: aggregation and specialization. The other three, configuration, analogy construction, and instantiation, all have a specific purpose in method engineering. Their application, however, involves different costs in preparation and utilization compared with the other two mechanisms.

The results at hand, therefore, suggest that a *mechanism mix* should be used when engineering methods that are to be situationally adapted. Central method parts, e.g. aggregateable components, which are used often, should be configurable or at least instantiable, parts of lesser detail and specificity should be specializable or available to analogy construction. It should be avoided to solve all adaptation problems with lesser structured mechanisms as reproducibility and lack of guidance cannot be qualitatively compensated by the cost savings in the preparation of a method.

This entails for method engineers to rethink their method proposals and to consider the herein described mechanisms to enrich and refine their methods with configuration, instantiation, aggregation, specialization, and if necessary analogy construction. It is by no means prescriptive to use more than one of the mechanisms. However, situations may be very diverse – why should not method adaptation be diverse as well?

Future research on this topic will have to deal with an analysis of the actual utilization of the mechanisms of situational methods in case studies. This analysis may point out which mechanisms are considered to be more efficacious than others by method users and can give hints on fruitful future research directions. Furthermore, it will allow validating the five general assumptions we explicated in the previous section.

References

1. F.P. Brooks, Essence and Accidents of Software Engineering, *IEEE Computer* **20**(4), 10-19 (1987).
2. M. Lindvall and I. Rus, Process Diversity in Software Development, *IEEE Software* **17**(4), 14-18 (2000).
3. K. Kautz, The Enactment of Methodology: The Case of Developing a Multimedia Information System, in: Proc. 25th International Conference on Information Systems (ICIS 2004) (Washington, D.C., 2004), pp. 671-683.

4. B. Fitzgerald, N.L. Russo, and T. O'Kane, Software Development: Method Tailoring at Motorola, *Communications of the ACM* **46**(4), 65-70 (2003).
5. K. Wistrand and F. Karlsson, Method Components – Rationale Revealed, in: Proc. 16th International Conference on Advanced Information Systems Engineering (CAiSE 2004) (Riga, 2004), pp. 189-201.
6. A.H.M. ter Hofstede and T.F. Verhoef, On the Feasibility of Situational Method Engineering, *Information Systems* **22**(6/7), 401-422 (1997).
7. S. Kelly, M. Rossi, and J.-P. Tolvanen, What is Needed in a MetaCASE Environment?, *Enterprise Modelling and Information Systems Architectures* **1**(1), 25-35 (2005).
8. J. Luoma, S. Kelly, and J.-P. Tolvanen, Defining Domain-Specific Modeling Languages - Collected Experiences, in: Proc. 4th Object-Oriented Programming Systems, Languages, and Applications Workshop on Domain-Specific Modeling (OOPSLA 2004) (Vancouver, 2004).
9. K. Kumar and R.J. Welke, Methodology Engineering: A Proposal for Situation-specific Methodology Construction, in: Challenges and Strategies for Research in Systems Development, edited by W. W. Cottermann and J. A. Senn (John Wiley & Sons Ltd., Chichester, 1992), pp. 257-269.
10. S. Brinkkemper, Method Engineering - Engineering of Information Systems Development Methods and Tools, *Information and Software Technology* **38**(4), 275-280 (1996).
11. A.F. Harmsen, *Situational Method Engineering* (Twente, Utrecht, 1997).
12. K. Wimmer and N. Wimmer, Conceptual modeling based on ontological principles, *Knowledge Acquisition* **4**(4), 387-406 (1992).
13. C. Alexander, *A Pattern Language: Towns, Buildings, Constructions* (Oxford Univ. Press, New York, 1977).
14. M. Fowler, *Analysis Patterns: Reusable Object Models* (Addison-Wesley, Menlo Park, 1996).
15. E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software* (Addison-Wesley, Reading, 2005).
16. D. Gupta and N. Prakash, Engineering Methods from Method Requirements Specifications, *Requirements Engineering* **6**(3), 135-160 (2001).
17. C. Szyperski, D. Gruntz, and S. Murer, *Component Software: Beyond Object-Oriented Programming* (Addison-Wesley, London, 2003).
18. J. Becker, L. Algermissen, T. Falk, D. Pfeiffer, and P. Fuchs, Model Based Identification and Measurement of Reorganization Potential in Public Administrations – the PICTURE-Approach, in: Proc. 10th Pacific Asia Conference on Information Systems (PACIS) (Kuala Lumpur, 2006), pp. 860-875.
19. P. Slater, Output from generic packages, *ACM SIGAda Ada Letters* **XV**(3), 76-79 (1995).
20. T.C. Jones, Reusability in Programming: A Survey of the State of the Art, *IEEE Transactions on Software Engineering* **10**(5), 488-493 (1984).
21. M. Rosemann and W.M.P. van der Aalst, A Configurable Reference Modelling Language, *Information Systems* **32**(1), 1-23 (2007).
22. J. Becker, M. Kugeler, and M. Rosemann, *Process Management: A Guide for the Design of Business Processes* (Springer, Berlin, 2007).
23. A.-W. Scheer, *Business Process Engineering: Reference Models for Industrial Enterprises* (Springer, Berlin et al., 2002).
24. J. Becker and R. Schütte, *Handelsinformationssysteme* (Redline Wirtschaft, Frankfurt am Main, 2004).
25. J. vom Brocke, Design Principles for Reference Modelling – Reusing Information Models by Means of Aggregation, Specialisation, Instantiation, and Analogy, in: Reference Modeling for Business Systems Analysis, edited by P. Fettke and P. Loos (Idea Group Publishing, Hershey, 2007), pp. 47-75.

26. J. Becker, P. Delfmann, and R. Knackstedt, Adaptive Reference Modeling: Integrating Configurative and Generic Adaptation Techniques for Information Models, in: Proc. Reference Modeling Conference (RefMod) (Passau, 2006).
27. C. Crawford: Core Components Technical Specification - Part 8 of the ebXML Framework. Version 2.01. UN/CEFACT (2003)
28. S. Brinkkemper, M. Saeki, and F. Harmsen, Meta-modelling Based Assembly Techniques for Situational Method Engineering, *Information Systems* **24**(3), 209-228 (1999).
29. M. Leppänen, Contextual Method Integration, in: Advances in Information System Development, edited by G. Knapp, G. Wojtkowski, J. Zupancic, and S. Wrycza (Springer, 2007).
30. J. Ralyté and C. Rolland, An Assembly Process Model for Method Engineering, in: Proc. 13th International Conference on Advanced Information Systems Engineering (CAiSE 2001). Lecture Notes in Computer Science. Vol 2068 (Interlaken, 2001), pp. 267-283.
31. S. Brinkkemper, M. Saeki, and F. Harmsen, Assembly Techniques for Method Engineering, in: Proc. 10th International Conference on Advanced Information Systems Engineering (CAiSE 1998). Lecture Notes in Computer Science (Pisa, 1998), pp. 381-400.
32. T. Punter and K. Lemmen, The MEMA-model: towards a new approach for Method Engineering, *Information and Software Technology* **38**(4), 295-300 (1996).
33. M. Saeki and K. Wenyin, Specifying Software Specification & Design Methods, in: Proc. 6th International Conference on Advanced Information Systems Engineering (CAiSE 1994) (Utrecht, 1994).
34. J. Ralyté and C. Rolland, An Approach for Method Reengineering, in: Proc. 20th International Conference on Conceptual Modeling (ER 2001) (Yokohama, 2001), pp. 471-484.
35. X. Song, Systematic Integration of Design Methods, *IEEE Software* **14**(2), 107-117 (1997).
36. F. Karlsson and K. Wistrand, Combining Method Engineering with Activity Theory: Theoretical Grounding of the Method Component Concept, *European Journal of Information Systems* **15**(1), 82-90 (2006).
37. J. Ralyté, R. Deneckère, and C. Rolland, Towards a Generic Model for Situational Method Engineering, in: Proc. 15th International Conference on Advanced Information Systems Engineering (CAiSE 2003) (Klagenfurt, 2003), pp. 95-110.
38. M. Bajec, D. Vavpotič, and M. Krisper, Practice-driven Approach for Creating Project-specific Software Development Methods, *Information and Software Technology* **49**(4), 345-365 (2007).
39. R. Baskerville and J. Stage, Accommodating Emergent Work Practices: Ethnographic Choice of Method Fragements, in: Proc. IFIP TC8/WG8.2 Working Conference on Realigning Research and Practice in IS Development: The Social and Organisational Perspective (Boise, ID, 2001), pp. 12-28.
40. J. Becker, R. Knackstedt, D. Pfeiffer, and C. Janiesch, Configurative Method Engineering: On the Applicability of Reference Modeling Mechanisms in Method Engineering, in: Proc. 13th Americas Conference on Information Systems (AMCIS 2007) (Keystone, CO, 2007).
41. J. Cameron, Configurable Development Processes, *Communications of the ACM* **45**(3), 72-77 (2002).
42. S. Greiffenberg, *Methodenentwicklung in Wirtschaft und Verwaltung* (Verlag Dr. Kovac, Hamburg, 2003).
43. S. Henninger, A. Ivaturi, K. Nuli, and A. Thirunavukkaras, Supporting Adaptable Methodologies to Meet Evolving Project Needs, in: Proc. Joint Conference on XP Universe and Agile Universe (Chicago, IL, 2002), pp. 33-44.
44. F. Karlsson: Method Configuration: Method and Computerized Tool Support. Linköping (2005)

45. F. Karlsson and P.J. Ågerfalk, Method Configuration: Adapting to Situational Characteristics While Creating Reusable Assets, *Information and Software Technology* **46**(9), 619-633 (2004).
46. M. Leppänen: An Ontological Framework and a Methodical Skeleton for Method Engineering: A Contextual Approach. Jyväskylä (2005)
47. B.A. Nuseibeh: A Multi-Perspective Framework for Method Integration. London (1994)
48. J. Odell, Meta-modelling, in: Proc. OOPSLA'95 Workshop on Metamodelling in OO (Austin, TX, 1995).
49. C. Patel, S. de Cesare, N. Iacovelli, and A. Merico, A Framework for Method Tailoring: A Case Study, in: Proc. 2nd OOPSLA Workshop on Method Engineering for Object-Oriented and Component-Based Development (Vancouver, 2004).
50. I. Mirbel and J. Ralyté, Situational Method Engineering: Combining Assembly-based and Roadmap-driven Approaches, *Requirements Engineering* **11**(1), 58-78 (2006).
51. J.-P. Tolvanen: Incremental Method Engineering with Modeling Tools: Theoretical Principles and Empirical Evidence. Jyväskylä (1998)
52. M. Rossi, B. Ramesh, K. Lyytinen, and J.-P. Tolvanen, Managing Evolutionary Method Engineering by Method Rationale, *Journal of the Association for Information Systems* **5**(9), 356-391 (2004).
53. M.A.G. van Offenbeek and P.L. Koopman, Scenarios for System Development: Matching Context and Strategy, *Behaviour & Information Technology* **15**(4), 250-265 (1996).
54. J. Becker, C. Janiesch, S. Seidel, and C. Brelage, A Framework for Situational and Evolutionary Language Adaptation in Information Systems Development, in: Advances in Information System Development, edited by G. Knapp, G. Wojtkowski, J. Zupancic, and S. Wrycza (Springer, 2007).
55. A. Mili, S.F. Chmiel, R. Gottumukkala, and L. Zhang, An Integrated Cost Model for Software Reuse, in: Proc. 22nd International Conference on Software Engineering (Limerick, Ireland, 2000), pp. 157-166.
56. J. Becker, C. Janiesch, and D. Pfeiffer, Towards more Reuse in Conceptual Modeling: A Combined Approach using Contexts, in: Proc. 19th International Conference on Advanced Information Systems Engineering (CAiSE 2007) Forum (Trondheim 2007).
57. G. Guizzardi, L.F. Pires, and M.J.v. Sinderen, On the Role of Domain Ontologies in the Design of Domain-Specific Visual Modeling Languages, in: Proc. 17th ACM Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA 2002) (Seattle, WA, 2002).
58. C. Janiesch, Implementing Views on Business Semantics: Model-based Configuration of Business Documents, in: Proc. 15th European Conference on Information Systems (ECIS 2007) (St. Gallen, 2007).