Modularization Constructs in Method Engineering: Towards Common Ground?

Pär J. Ågerfalk[1], Sjaak Brinkkemper[2], Cesar Gonzalez-Perez[3], Brian Henderson-Sellers[4], Fredrik Karlsson[5], Steven Kelly[6] and Jolita Ralyté[7]
1 Lero – The Irish Software Engineering Research Centre, University of Limerick, Ireland, and Uppsala University, Sweden, par.agerfalk@lero.ie
2 Institute for Information and Computing Sciences, Utrecht University, Netherlands, S.Brinkkemper@cs.uu.nl
3 European Software Institute, cesargon@verdewek.com
4 University of Technology, Sydney, Australia, brian@it.uts.edu.au
5 Methodology Exploration Lab, Dept. of Informatics (ESI), Örebro University, Sweden, fredrik.karlsson@esi.oru.se
6 MetaCase, Finland, stevek@metacase.com
7 CUI, University of Geneva, Switzerland, jolita.ralyte@cui.unige.ch

Abstract. Although the Method Engineering (ME) research community has reached considerable maturity, it has not yet been able to agree on the granularity and definition of the configurable parts of methods. This state of affairs is causing unnecessary confusion, especially with an ever increasing number of people contributing to ME research. There are several competing notions around, most significantly 'method fragments' and 'method chunks', but also 'method components' and 'process components' are used in some quarters and have also been widely published. Sometimes these terms are used interchangeably, but there appears to be important semantic and pragmatic differences. If the differences are unimportant, we should be able to come to an agreement on what construct to promote. Alternatively, the different constructs may serve different purposes and there is a need for them to coexist. If this is the case, it should be possible to pinpoint exactly how they are related and which are useful in what contexts. This panel is a step towards finding common ground in this area, which arguably is at the very core of ME.

1 Introduction

Since its inception in the early to mid 1990s, the Method Engineering (ME) research community has reached considerable maturity. Nonetheless, there is still

Please use the following format when citing this chapter:

Ågerfalk, P. J., Brinkkemper, S., Gonzalez-Perez, C., Henderson-Sellers, B., Karlsson, F., Kelly, S., Ralyté, J., 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. 359-368.

some ambiguity with regards to fundamental concepts and terminology. Since situational ME is fundamentally concerned with the assembly and configuration of information systems engineering methods, understanding the basic building blocks of methods is arguably core to the discipline. In order to devise appropriate ME processes and tools, we need to understand what building blocks those processes and tools are to handle. To date, a number of different such 'modularization constructs' have been suggested. Among the most cited are 'method fragments', 'method chunks', 'method components' and 'process components'. Along with these constructs come certain interpretations of related concepts such as method, technique, notation, process, deliverable, work product, tool etc. Sometimes the constructs are used interchangeably, but there appears to be important semantic and pragmatic differences. If the differences are unimportant, we should be able to come to an agreement on what construct to promote. Alternatively, the different constructs may serve different purposes and there is a need for them to coexist. If this is the case, it should be possible to pinpoint exactly how they are related and which are useful in what contexts. This panel is a significant step towards finding common ground in this area.

The remainder of this panel introduction consists of a brief description of the four modularization constructs mentioned above, followed by a brief introduction of the panellists. The aim of this document is to provide some background and context for the panel. The actual discussion and its outcome will be reported elsewhere.

2 Method Fragments

One of the earliest and arguably most important modularization construct in ME is that of the *method fragment*. It was first proposed and elaborated by Brinkkemper and colleagues [1–4] and has since been widely adopted in ME research. Essentially, method fragments are standardized building blocks based on a coherent part of a method [1]: '... a description of an IS engineering method, or any coherent part thereof'. A complete method, such as 'OMT', is a method fragment and so is any single concept used within that method, such as 'object'. A method fragment thus resides on a certain so-called *layer of granularity*, of which five are possible: method, stage, model, diagram, or concept [4]. Consequently, 'object' resides on the concept layer while 'OMT' resides on the method layer. Furthermore, a method fragment is either a *process fragment* or a *product fragment*. Process fragments represent the activities, stages etc that are to be carried out and product fragments represent deliverables, diagrams etc that are to be produced, or that are required, during development. Method fragments are stored in a method base from which they can be retrieved using a query language, such as the Method Engineering Language (MEL) [5]. This way, a situational method can be constructed by combining a number of method fragments into a situational method. To be meaningful and useful, such a combination must follow certain assembly rules that adhere to the construction principles in the process perspective on the one hand and in the product perspective on the other hand. This has been explored by Brinkkemper *et al.* [4].

Currently the team of Brinkkemper at Utrecht University is focussing on the methodological support for product software companies, i.e. companies that develop and market software products for a particular market. About 10% of the total ICT spending is spend on software products and examples of such companies are Microsoft, SAP, Oracle, and Business Objects [6]. As these companies keep the ownership of the software code and all auxiliary materials belonging to the software product, these companies create and maintain a proprietary software development method. From the start-up phase where they begin with bug tracing to a more consolidated company with all kinds of quality engineering processes in place. The gradual growth of the product software company calls for a more incremental growth from simple method fragments to more complex fragments at a later stage [7, 8]. The evolution from simple to complex processes properly supported with development tools while keeping the historical documentation and the methodological context in place are a significant scientific challenge for the coming years.

3 Method Chunks

The *method chunk* concept was proposed by Rolland and colleagues [9–13] as a way to capture more of the situational aspects in ME and to appropriately support the retrieval process. The concept was introduced together with a contextual ME approach using scenarios [10] and suggests an organization of the method base in two levels, one method knowledge level and one method meta-knowledge level [9]. The former level is represented by the method chunk body and the latter captures the situational and intentional aspect of method chunks in the method chunk descriptor. In [9] the method knowledge level is operationalized in a three level abstraction model and method chunks are classified into component, pattern or framework. A method component is a complete method description. A pattern is, for example, a set of generic guidelines for writing test scripts. Finally, a framework is a meta-method that guides the construction of a way-of-working within a specific method.

In the latest work [11–13] the concept of method chunk is defined as autonomous, cohesive and coherent part of a method providing guidelines and related concepts to support the realisation of some specific system engineering activity (e.g. business modelling, requirements specification, design etc). In fact, the method knowledge is captured in the method chunk *body* and *interface*. The interface defines pre and post conditions of chunk application formalised by a couple <*situation, intention*>. The situation specifies the required input product part(s) while the intention defines the goal that the chunk helps to achieve. For example, the interface of the method chunk supporting identification of Business Actors and Use Cases within the RUP could be modelled as <(Business knowledge, problem description, interview results), Identify and describe business actor(s) and use case(s)>.

The body of the method chunk includes two kinds of knowledge: product and process. The product knowledge defines the work products (input and output) used by the method chunk (e.g. the definitions of the concept "actor" and "use case and their relationships). This knowledge is generally expressed in terms of meta-models.

The process knowledge captured in a method chunk provides guidelines how to obtain target product(s) from input product(s) (e.g. the guidelines how to identify system actors and their business use cases). The guideline can be represented as an informal description or expressed by using different process modelling formalisms such as NATURE context trees [14] or MAP graphs [15] depending on how rich and complex it is. The fact that a guideline can be complex (i.e. composed of a set of sub-guidelines) means that the corresponding method chunk can be an aggregate of a collection of smaller chunks.

The *descriptor* (i.e. method meta-knowledge) of the method chunk extends the contextual knowledge defined in the *interface* with a set of criteria that help to better locate the engineering situation in which the method chunk is useful. A detailed classification of these criteria related to the information systems development, named Reuse Frame, is proposed in [13].

A method chunk is selected for a specific situation based on the characterization of that situation and how relevant it is to achieve the intention of the method chunk. Hence, the intention of a method chunk, the goal that can be achieved through application of the way of working prescribed by the method chunk, is central.

Method chunks are retrieved from the method base through the use of meta-knowledge. Based on the structure of the method base, where method chunks have been clustered and described using interfaces and descriptors, it is possible to query the method base using a query language. For example, it is possible to select a chunk from the RUP if it has a representation in the method base. Hence, a method chunk query language has similarities with MEL when using method fragments.

Some initial comparisons of method fragments and method chunks are to be found in [13] and [16].

4 Method Components

First introduced by Goldkuhl and colleagues [17, 18], the *method component* concept has recently been further developed by Karlsson and others [19–22]. The basic idea is to view methods as constituted by exchangeable and reusable components. Fundamentally, each component consists of descriptions for ways of working (a process), notations, and concepts [17]. A process describes rules and recommendations for and informs the method (component) user what actions to perform and in what order. Notation means semantic, syntactic and symbolic rules for documentation. Concepts are categories included in the process and the notation. Concepts and notation together constitute what is sometimes referred to as a modelling language, such as the UML. A method component can also be used separately and independently from other components. Each method component addresses a certain aspect of the problem at hand.

Building further on this original method component concept, Karlsson [21] defines it as ‘a self-contained part of a method expressing the transformation of one or several artifacts into a defined target artifact and the rationale for such a transformation.’ The method component construct thus draws significantly on the idea of method rationale – the systematic treatment of the arguments and reasons behind a particular method [20, 23, 24, 25]. While the intention of a method chunk is typically expressed in terms of the action that immediately satisfies the intention, method rationale aims to direct method engineers' attention to the underlying assumptions of those actions and promote a critical attitude towards the different parts of a method.

A method component consists of two parts: its content and the rationale expressing why the content is designed as it is and what it can bring about. The content of a method component is an aggregate of method elements [21]: A method element is a part of a method that manifests a method component's target state or facilitates the transformation from one defined state to another. The concept of method element can be specialized into five categories. Firstly, there are three interrelated parts of prescribed action, concept and notation. These categories are complemented with artefact and actor role as two further sub-types of method element. Artefacts act as deliverables from the transformation process as well as input to this process. Methods are here viewed as heuristic procedures (heurithms) and consequently specified inputs are considered to be recommended inputs. However, a method component needs to have at least one input. Otherwise the method component will not have any meaningful support in the method. One exception to this is method components that initiate new activities that are later integrated with the result from other method components. The selection of actor roles are determined by the prescribed actions that need to be part of the transformation process. Actor roles are played either as drivers of the prescribed actions in the method component or as participants.

The rationale part of the method component concept consists of two parts: goals and values. Method elements exist for reasons, which are made explicit by means of associating method elements to the goals. These goals are anchored in values of the method creator [18, 25]. Taken together, goals and values are often considered important constituents of a methods underlying perspective [18] or 'philosophy' [26]. In method engineering, method rationale is more important than the deliverable as such. Through the method rationale it is possible to address the goals that are essential in order to fulfil the overall goal of a specific project. Prescribed actions and artefacts are only means to achieve something and method rationale can thus help developers not to lose sight of that ultimate result, and also help them find alternative ways forward.

It is important to point out that in our current understanding, method components always reside on the 'artefact layer of granularity' and represent a non-hierarchal concept. This is to reflect the notion that method components are the smallest coherent parts of a method that are practically useful. This design choice is based on two empirical observations [21]: The first, and most important, is that systems developers' tend to focus on the artefacts (a.k.a. deliverables) when discussing situational methods, and these are viewed as non-hierarchal patterns. Second, it has proven difficult to balance precision and cost with hierarchal concepts in situational method engineering.

5 OPF Method/Process Components

The OPEN Process Framework [27, 28] also utilizes the concept of a method fragment but stresses that each fragment needs to be generated from an element in a prescribed underpinning metamodel. This metamodel has recently been upgraded with the recent availability of the International Standard ISO/IEC 24744 'Software Engineering Metamodel for Development Methodologies' [29]. While many of the OPF fragments focus on 'process' there are also significant numbers for products and producers (people and tools involved in software development). These are the three acknowledged top-level meta-elements for methodologies leading to: process-focussed fragments (e.g. a kind of task or technique), product-focussed fragments (a kind of diagram, document or other work product) and producer-focussed fragments (e.g. a role played by a member of the software development team, a testing tool) – the last of which (producers) is not represented in other SME approaches. In the OPF, these method fragments are defined separately and then linked together using instances of metamodel classes such as *ActionKind*, representing a single usage event that a given process fragment exerts upon a given product fragment. This class contains an attribute, *Type*, that specifies what kind of action the process part is exerting on the product part. For example, imagine a methodology that contains a requirements validation task. This task takes a draft requirements document as input and modifies it accordingly through the validation process, creating, as well, a requirements defect list. Modelling this task plus the two involved products (one of which is both an input *and* an output) can be easily modelled by using two actions: one action would map the requirements validation task to the requirements document, specifying a type 'modify', and a second action would map the same requirements validation task to the requirements defect list, specifying the type as 'create'. The relationships between process- and product-oriented fragments are thus clearly specified. (It must be noted that the actions are lightweight entities in the methodology that act as mappings between heavyweight process- and product-oriented fragments. Actions are not containers, as are chunks.).

6 The MetaEdit Experience

Research in the MetaPHOR project, object-oriented ideas in the implementation of MetaEdit+, and experience with customers, led MetaCase largely to avoid the question of the size or definition of 'chunks' or 'fragments'. Rather they are able to reuse anything, from a single Property type (e.g. the 'Actor Name' field of the Actor type in UML Use Case diagrams) through Object types (e.g. Actor) to Graph types (e.g. Use Case Diagram) and interlinked sets of Graph types (e.g. UML). Accompanying these central and clearly identifiable elements go various rules that map to the 'harder' end of the process scale, generators that form the operational semantics, along with 'softer' parts of processes and things like problem domain semantics. Mainly, though, the focus has been on support for creating entirely new modelling languages, and how reuse and linking of types in the metamodel allows reuse and linking on the model level.

7 About the Panellists

Prof. Pär J. Ågerfalk (panel moderator) is a Senior Researcher at Lero – The Irish Software Engineering Research Centre and holds the Chair in Computer Science in Intersection with Social Sciences at Uppsala University. He received his PhD in Information Systems Development from Linköping University and has held fulltime positions at Örebro University, University of Limerick, and Jönköping International Business School. His current research centres on open source software development, globally distributed and flexible development methods and how IS development can be informed by language/action theory. His work has appeared in a number of leading IS journals and conferences and he is currently an associate editor of the *European Journal of Information Systems* and a senior associate editor for a special issue of *Information Systems Research* on Flexible and Distributed IS Development.

Prof. Sjaak Brinkkemper is professor of Organisation and Information at the Institute of Information and Computing Sciences of the Utrecht University, the Netherlands. Before he was a consultant at the Vanenburg Group and a Chief Architect at Baan. Before Baan he held academic positions at the University of Twente and the University of Nijmegen, both in the Netherlands. He holds a MSc and a PhD in of the University of Nijmegen. He has published five books and more than hundred papers on his research interests: software product development, information systems methodology, meta-modelling, and method engineering.

Dr. Cesar Gonzalez-Perez has been a research project leader at the European Software Institute until last June, where he led research efforts in the areas of method engineering, metamodelling and conceptual modelling. Previously, he worked over 3 years at the Faculty of IT of the University of Technology, Sydney, from where he co-edited the standardisation projects that resulted in the standard metamodels AS4651 and ISO/IEC 24744. Cesar is also the founder and former technical director of Neco, a company based in Spain specialising in software development support services, which include the deployment and use of OPEN/Metis at small and mid-sized organisations. Cesar has also worked for the University of Santiago de Compostela in Spain as a researcher in computing and archaeology, and got his PhD in this topic in 2000.

Dr. Fredrik Karlsson received his PhD in Information Systems Development from Linköping University and is currently a Senior Lecturer at Örebro University. His research focuses on tailoring of systems development methods, systems development methods as reusable assets, and CAME tools. He has developed the CAME tool MC Sandbox that supports method configuration. At Örebro University he heads the Methodology Exploration Lab and is an active member of the Swedish research network VITS. His work has appeared in, for example, *European Journal of Information Systems* and *Information and Software Technology*.

Dr. Steven Kelly is the CTO of MetaCase and co-founder of the DSM Forum. He has over a dozen years of experience of building metaCASE environments and acting as a consultant on their use in Domain-Specific Modelling. He is architect and lead developer of MetaEdit+, MetaCase's domain-specific modelling tool. Ever present on the program committee of the OOPSLA workshops on Domain-Specific Modelling, he co-organized the first workshop in 2001. He is author of over 20 articles in both academic and industry publications, and is a member of IFIP WG 8.1 and the editorial board for the Journal of Database Management. Steven has an M.A. (Hons.) in Mathematics and Computer Science from the University of Cambridge, and a Ph.D. from the University of Jyväskylä.

Dr. Jolita Ralyté is currently a senior researcher and lecturer at the University of Geneva, Department of Information Systems. She obtained a PhD in Computer Science from the University of Paris 1 – Sorbonne in 2001. The research areas of Dr. Ralyté include situational method engineering, requirement engineering, information systems evolution and interoperability and distributed information systems development. She is in charge of the International Method Engineering Task Group within the IFIP WG 8.1 and the task group TG6 dealing with methods and method engineering techniques supporting various systems interoperability issues within the European NoE INTEROP. Her work has been published in various international conferences and journals. Dr Ralyté has been involved in the organisation of a number of international conferences and workshops (ME'07, OOIS'03, EMSISE'03, Interop-ESA'05, SREP'05, SREP'07 and Doctoral Symposium at I-ESA'06) and co-edited a special issue of *SPIP* with revised best papers from SREP'05.

References

1. Harmsen, F., Brinkkemper, S., and Oei, H. (1994). Situational Method Engineering for Information System Project Approaches. In: A.A. Verrijn Stuart and T.W. Olle (Eds.), Methods and Associated Tools for the Information Systems Life Cycle. *Proceedings of the IFIP WG 8.1 Working Conference*, Maastricht, Netherlands, September 1994, IFIP Transactions A-55, North-Holland, 1994, ISBN 0-444-82074-4, pp. 169-194. Also in: *Memoranda Informatica* 94-03, ISSN 0924-3755, 34 pages, January 1994.
2. Brinkkemper, S. (1996). Method engineering: Engineering of information systems development methods and tools. *Information and Software Technology, 38*(4), 275–280.
3. Harmsen, A.F. (1997). *Situational method engineering.* Doctoral dissertation, Moret Ernst & Young Management Consultants, Utrecht, The Netherlands.
4. Brinkkemper S., Saeki, M., and Harmsen, F. (1999). Meta-Modelling Based Assembly Techniques for Situational Method Engineering, *Information Systems,* 24(3), pp. 209–228.
5. Brinkkemper S., Saeki M., and Harmsen, F. (2001). A Method Engineering Language for the Description of Systems Development Methods (Extended Abstract). In: K.R. Dittrich, A Geppert, and M.C. Norrie (eds.), *Proceedings of the 13th International Conference CAiSE'01*, pp. 173-179, Interlaken, Switzerland, 2001, Lecture Notes in Computer Science, Springer Verlag. ISBN 3-540-42215-3.
6. Xu, L. and Brinkkemper, S. (2007). Concepts for Product Software. To appear in *European Journal of Information Systems.*
7. Weerd, I. van de, Brinkkemper, S., Souer, J., and Versendaal, J. (2006). A Situational Implementation Method for Web-based Content Management System-applications: Method Engineering and Validation in Practice. *Software Process: Improvement and Practice* 11(5), 521-538.
8. Weerd, I. van de, Brinkkemper, S., Versendaal J. (2007). Concepts for Incremental Method Evolution: Empirical Exploration and Validation in Requirements Management. In *Proceedings of the 19th International Conference on Advanced Information Systems Engineering*, LNCS 4495, 469–484.

9. Rolland, C. and Prakash, N. (1996). A proposal for context-specific method engineering. In S. Brinkkemper, K. Lyytinen & R. Welke (Eds.), *Method Engineering: Principles of method construction and tool support* (Vol. 191–208): Chapman & Hall.
10. Rolland, C., Plihon, V. and Ralyté, J. (1998). Specifying the Reuse Context of Scenario Method Chunks. *Proceedings of the 10th International Conference on Advanced Information System Engineering (CAISE'98),* Pisa, Italy, June 1998. B. Pernici, C. Thanos (Eds), LNCS 1413, Springer-Verlag, pp. 191-218.
11. Ralyté, J. and Rolland, C. (2001). An Approach for Method Reengineering. Proceedings of the 20th International Conference on Conceptual Modeling (ER2001), LNCS 2224, Springer-Verlag, pp.471-484.
12. Ralyté, J., Deneckère, R., and Rolland, C. (2003). Towards a Generic Model for Situational Method Engineering, In *Proceedings of 15th International Conference on Advanced Information Systems Engineering (CAiSE 2003), Klagenfurt, Austria, June 16–18, 2003,* (Eds, Eder J, *et al.*) Heidelberg, Germany: Springer-Verlag, pp. 95–110.
13. Mirbel, I. and Ralyté, J. (2006). Situational method engineering: combining assembly-based and roadmap-driven approaches, *Requirements Engineering,* 11(1), pp. 58–78.
14. Jarke, M., Rolland, C., Sutcliffe, A., and Domges, R. (1999). *The NATURE requirements Engineering.* Shaker Verlag, Aachen.
15. Rolland, C., Prakash, N., and Benjamen, A. (1999). A multi-model view of process modelling. *Requirements Engineering, 4*(4), 169–187.
16. Henderson-Sellers, B., Gonzalez-Perez, C., and Ralyté, J. (2007). Situational method engineering: chunks or fragments? *CAiSE Forum, Trondheim, 11-15 June 2007,* 89-92
17. Röstlinger, A., and Goldkuhl, G. (1996). *Generisk flexibilitet: På väg mot en komponentbaserad metodsyn,* In Swedish: "Generic flexibility: Towards a component-based view of methods", Technical Report LiTH-IDA-R-96-15, Dept. of Computer and Information Science, Linköping University. Originally presented at VITS Höstseminarium 1994.
18. Goldkuhl, G., Lind, M., and Seigerroth, U. (1998). Method integration: The need for a learning perspective. *IEE Proceedings Software,* 145, 113–118.
19. Ågerfalk, P.J. (2003). *Information Systems Actability: Understanding Information Technology as a Tool for Business Action and Communication.* Doctoral dissertation. Dept. of Computer and Information Science, Linköping University, 2003.
20. Wistrand, K. and Karlsson, F. (2004). Method Components: Rationale Revealed. In Persson, A. and Stirna, J. (eds.) *Proceedings of the 16th International Conference on Advanced Information Systems Engineering (CAiSE 2004),* Riga, Latvia, June 7-11, 2004. Heidelberg, Springer-Verlag.
21. Karlsson, F. (2005) *Method Configuration: Method and Computerized Tool Supp*ort. Doctoral dissertation. Dept. of Computer and Information Science, Linköping University.
22. Karlsson, F. and Wistrand, K. (2006). Combining method engineering with activity theory: theoretical grounding of the method component concept. *European Journal of Information Systems,* 15, 82-90.
23. Ågerfalk, P.J. and Wistrand, K. (2003). Systems Development Method Rationale: A Conceptual Framework for Analysis. In Camp, O., Filipe, J., Hammoudi, S. & Piattini, M. (Eds.) *Proceedings of the 5th International Conference on Enterprise Information Systems (ICEIS 2003).* Angers, France.
24. Rossi, M., Ramesh, B., Lyytinen, K., and Tolvanen, J.-P. (2004). Managing evolutionary method engineering by method rationale. *Journal of the Association for Information Systems, 5*(9), 356–391.
25. Ågerfalk, P.J. and Fitzgerald, B. (2006). Exploring the Concept of Method Rationale: A Conceptual Tool for Method Tailoring. In Siau, K. (Ed.) *Advanced Topics in Database Research Vol 5.* Hershey, PA, Idea Group.

26. Fitzgerald, B., Russo, N. L., and Stolterman, E. (2002). *Information systems development - methods in action.* London: McGraw-Hill.
27. Henderson-Sellers, B. and Graham, I.M. (1996). OPEN: toward method convergence? *IEEE Computer,* 29(4), 86-89
28. Firesmith, D.G. and Henderson-Sellers, B. (2002). *The OPEN Process Framework. An Introduction,* Addison-Wesley, 330pp
29. ISO/IEC (2007). *Software Engineering. Metamodel for Development Methodologies. ISO/IEC 24744*: International Standards Organization / International Electrotechnical Commission, Geneva.

Printed in the USA