# Requirements for Software Process Modeling Tools

**Rodrigo M. Pagliares**

Computer Science Department – Federal University of Minas Gerais (UFMG)
Av. Antônio Carlos, 6627 – CEP 31.270-010 – Belo Horizonte – MG - Brazil

pagliares@gmail.com

***Abstract.*** *Software development is all about methods. So, a clear understanding of the way these methods are constituted, organized and documented becomes vital to successful organizations. One well investigated way of achieve reasoning about methods is through their modeling. Method Engineers are the people responsible for defining, maintaining and improving these models and this is done by the use of automated tools. This work presents a catalog with requirements of such tools aiding team members, instructors, project managers and method engineers to compare different implementations on the market and helping tools providers to direct future developments.*

## 1. Introduction

Software development is all about methods. In the last years, several software development methods (SDM) were developed and proposed [Boehm 1996], [Krutchen 2003], [Jacobson+ 1999], [Humphrey 1995], [Schwaber 2004], [Beck 2004]. A portion of these tried to impose some form of rigor in the way which software is conceived, documented, developed and tested [Pfleeger and Atlee 2005, page: 59]. The other ones were formulated, in the late 1990s, by some developers who had resisted to this rigor and elaborated their own principles, trying to highlight the roles that flexibility could play in producing software quickly and capably. They codified their thinking in "the agile manifesto" [Agile 2009] that focuses on alternative ways of thinking about software development.

Several methods and SDM definitions can be found on literature. Accordingly to Brinkkemper, a method is *"an approach to perform a systems development project, based on a specific way of thinking, consisting of directions and rule, structured in a systematic way in development activities with corresponding development product"* [Brinkkemper, 1996]. More specifically, Ian Sommerville defines software engineering methods as *"Structured approaches to software development which include system models, notations, rules, design advice and process guidance"* [Sommerville 2006, pages: pages: 11-12]. Roger S. Pressman states that *"Software engineering methods provide the technical how-to's for building software. Methods encompass a broad array of tasks that include requirements analysis, design, program construction, testing, and support"* [Pressman 2001, page: 21]. All three definitions dictate a method having a broad range of constituent parts. This way, an adequate language to describe all these parts of a method in a consistent and useful manner is needed.

SDM can be expressed or modeled in natural language, with Unified Modeling Language (UML) [OMG 2009] or domain-specific languages [Nardini+ 2008], [Combemale+ 2006]. Natural language is ambiguous and UML is too much generic. In that sense both aren´t adequate to describe methods. A domain-specific language, which can be represented by an underpinning metamodel, on the other hand, is not as ambiguous as natural language and is constituted from major concepts of the subjects being modeled, not being so generic as UML. [Gonzalez-Perez+ 2008, pages:11-12], [ISO/IEC 2007, page: vi].

Having justified a domain-specific language as a foundation for SDM modeling, we could naively argue that the direct use of a metamodel for modeling methods would be a natural choice. However, as stated by Wilson de Pádua Paula Filho, in the context of the Software & Systems Process Engineering Metamodel (SPEM) [OMG 2008], a metamodel is of huge complexity and hard to be used directly by method engineers [Paula Filho 2009, page:108]. We are assuming in this paper that the same level of difficulty would appear in other metamodels aiming at the expression of SDM. To ease this complexity, a software development modeling tool becomes necessary. Until now, few of these were developed but, with the advent of the already cited SPEM and others such as ISO/IEC 24744 - Software Engineering Metamodel for Development Methodologies (SEMDM) [ISO/IEC 2007], is our belief that a plethora of them will be available in a near future.

Several people would benefit from using or constructing these tools: method engineers could more easily describe and improve the existing company´s software development methods; software development team members would have, for instance, some level of consistency and shared language across the organization; software engineering instructors could use these tools to teach current best practices in their curriculums and also bring the state-of-the-art software best practices to the software industry.

As so many people would benefit from modeling tools for software method development, a catalog with requirements of such ones would be of great help, aiding team members, instructors, project managers and method engineers to compare different implementations on the market and helping tools providers to direct future developments. Matthias Hoffmann, Nikolaus Kuh, Matthias Weber and Margot Bittner presented a catalog of requirements for requirements management tools in the area of automotive as well as aircraft and defense systems [Hoffmann+ 2004]. The objective of this work is to provide a similar catalog in structure but with focus on software method modeling tools, instead.

The remainder of this paper is organized as follows: in section 2, we present models and features of software processes modeling tools. Section 3 describes the requirements for such tools. Practical and operational aspects are presented in Section 4. Section 5 lists some tools, already on market, and how well they adhere to the requirements presented at Section 3. Finally, in Section 6, conclusions are made.

## 2. Models and features of modeling tools for software method development

Situational methods are customized information systems development methods defined in the discipline of Method Engineering [Brinkkemper, 1996]. One way to build such

methods is to reuse method portions stored in a method base, a.k.a. repository and assemble them in a project specific method [Ralyte+, 2001] , [Henderson-Sellers+, 2008]. This reuse is aided by the use of automated tools known as Computer Aided Method Engineering (CAME).

In this work we used three Came Tools, based on concepts of metamodels, aiming at the validation of the requirements catalog proposed in section 3. This was done through a cross-reference between each requirement presented on the catalog and its corresponded realization, or not, in the considered tool.

Eclipse Process Framework Composer (EPF-Composer) [Eclipse 2009] and IBM Rational Method Composer (RMC) [Rational 2009] were chosen because, although they were build upon the same kernel, they represent different worlds, open-source and proprietary, respectively. Osellus-Iris [Osellus 2009], another proprietary product, also cited by [Mäkilä 2006] was the third tool chosen.

## 3. Requirements

The catalog of requirements presented here has a structure similar to the one used by Neto et al. [Neto+ 2007].They used a, similar, hierarchical structure of requirements for software products as described in ISO/IEC Standard for Evaluation of Software Products [ISO/IEC 1991]. Their structure is comprised of three levels. The top level corresponds to the stakeholder with interest on the requirement. The second level corresponds to a subsection's title expressing a goal, need, expectation or constraint that must be attended. The third level corresponds to a detailed text discussing several aspects related to the second level. We also enumerate the requirements but, differently of Neto et.al, we provide a separated section (section 5 of this paper) to discuss the implementation of the identified requirements by tools already on market. All requirements are prioritized in the second level. The priorities "Essential", "High-Value" and "Follow-On" are indicated by "(+++)", "(++)" and "(+)" respectively.

Several communities aforementioned would benefit from the CAME tools. However, in order to develop a cohesive catalog, we merged roles such as team members and project manager in a unique one called *method users*. This generalization seems viable since all these roles share common expectations and is also the abstraction used by Henderson-Selez [Gonzalez-Perez+ 2008, page: 7]. In this paper, since roles are not always disjunctives in expectations, if there is more than one stakeholder being benefited from the requirement, the most prominent is mentioned. In that sense, two primary stakeholders are described in this work, *method users* that use methods in order to create software systems and *method engineers* who create and maintain methods according to the method users needs.

### 3.1. Requirements of the Method Engineer

**Requirement # 1 – The tool must be independent of the underpinning metamodel (++)**

Since the 80s, there have been various methods metamodels and finally Object Management Group (OMG) and International Organization for Standardization/International Electrotechnical Commission (ISO/IEC), decided to create, respectively, a specification (SPEM) and a standard (ISO/IEC 24744) in this area.

SPEM, already on version 2.0, is a standardized way of expressing any SDM. The specification was developed especially to address the unique and complex nature of software development. It is vendor, framework and methodology neutral and leverages the expressiveness and popularity of UML. SPEM is described through a meta-modeling language called Meta Object Facility (MOF) [OMG 2006] and reuses other OMG specifications [OMG 2008, page: 1].

The ISO/IEC standard defines SEMDM as *"a formal framework for the definition and extension of development methodologies for information-based domains (IBD), such as software, business or systems, including three major aspects: the process to follow, the work products to use and generate, and the people and tools involved"*. [ISO/IEC 2007, page: 1].

With two important organizations behind the formalism of SDM, the method engineer could not be restricted to one specific way of modeling. So, it is desirable that a method modeling tool allows the method engineer experiments with any SDM metamodel besides offering all necessary elements for modeling software methods and support the modeling efforts in order to make it as easy as possible.

**Requirement # 2 – The tool must allow the authoring of methods** (+++)

Before delving on the details of this requirement, it's worth mention a subtle difference between the meaning of *authoring* and *tailoring* and position ourselves for the remainder of this paper. Some authors must agree that defining a new method from a base one, modifying certain portions of it, with the objective to produce a situational method as result, would be some kind of method authoring. That is not the case here. In this paper, we consider authoring, the methods constructed from scratch or through reuse where the method fragment or chunk had been reused "as is", without modifications. When, through method reuse, we decide to modify, extend or reduce some aspect of the reusable chunk, we are *tailoring*. Another way to perceive this subtle difference is through a timeline since tailoring depends on existed method content while authoring could be done from scratch. That is, authoring must happen before tailoring.

High quality SDM can best be authored by means of situational methods. The idea is that methods can be dynamically assembled from existing components according to the user's needs. These components can be stored in a repository to be selected and incorporated into a method as necessary. Besides this *assembly approach* for getting a situational method, other approaches are possible, e.g. *extension based approach*, *paradigm based approach*, a.k.a. *evolution based approach* and *method for method configuration* (MMC) [Ralyte+ 2003], [Bajec+, 2007, page: 348].

No matter which approach used for situational method authoring, the tool must allow the method engineer capture and model the company´s best practices, making them available in the repository for future reuse.

Albeit situational methods are becoming more and more common nowadays, the market of out-of- the-box methods also called "of-the-shelf methods" or "pre-packaged" methods, e.g. Rational Unified Process (RUP), cannot be neglected and these kinds of methods play and probably will continue to play an important role on software development worldwide in the near future. For this reason, these kinds of methods also need be represented in the authoring tool.

**Requirement # 3 – The tool must allow the tailoring of methods** (+++)

Tailoring means adaptation or customization. As discussed before this presumes that some method content must exist before hand, from which the final, purpose-fit method is obtained [Gonzalez-Perez+ 2008, page: 5].

As noted by Ågerfalk & Fitzgerald [Ågerfalk+ 2006], Baskerville & Stage [Baskerville+ 2001] and Fredrik Karlsson [Karlsson 2008], there are two schools of thought involved with research on method tailoring (method engineering and method-in-action schools). The first one based on the positivist view and the last one on socio-organizational [Aydin+ 2005]. Aydin et al. classifies method engineering in either static or dynamic [Aydin+ 2005]. Static method engineering doesn't care too much about the method evolution during enactment while dynamic method engineering focus specifically on how methods are enacted in practice.

In order to obtain a high customized method, the method authoring tool must prescribe tailoring not only for each project, but also in a continuously way throughout the project lifecycle.

Requirement # 4 – The tool must allow the publication of methods (+++)

… To be done

### 3.2. Requirements from the Method Users

Requirement # 5 – The tool must enable the enactment of methods (++)

… To be done

Requirement # 6 – The tool must allow the verification and validation of the Authored or Tailored method (++)

… To be done

Requirement # 7 – The tool must support continuous method improvement (++)

… To be done

## 4. Practical and Operational Aspects

Below, we list practical and operational aspects related to the catalog of requirements:

- The tool must free the method engineer from tasks not related to method authoring and tailoring;
- The process of creation and tailoring of methods must be as easy and pleasant as possible using, for instance, forms and rich-text editors to configure and generate the situation method;
- The tool must provide in any time, a preview of the work being done by the method engineer.
- To be improved….

## 5. Tool Examples

This section cross-references the requirements described at section 3. For each requirement presented, we establish a link to the three tools chosen in this paper, investigating if the requirement is or not implemented on such tools as well the way it was or not implemented. For the sake of clarity and to make the reading as easily as possible, we decided to replicate in this section the description of the requirement presented in the requirements section. The tools evaluated comes right after the requirement replication.

**Requirement # 1 – The tool must be independent of the underpinning metamodel** (++)

Tool # 1 - EPF-Composer

EPF-Composer is attached to a SPEM Metamodel and doesn't provide ways to substitute it.

Tool # 2 - IBM Rational Method Composer (RMC)

As EPF-Composer, RMC is also SPEM dependent and cannot be configured with another metamodel.

Tool # 3 - Osellus Iris

To be done…

**Requirement # 2 – The tool must allow the authoring of methods** (+++)

Tool # 1 - EPF-Composer

With respect to method authoring, EPF-Composer lets the method engineer define a completely new content from scratch (including companies' best practices), or through reuse.

In order to create situational methods, EPF-Composer uses the *paradigm based approach*, cited on requirement #2, through the reuse of components organized in a method library (repository), called plug-ins. Within a method plug-in, content is separated into method content, e.g. *tasks, roles, work products* and processes, e.g. *capability patterns*. This idea of method separation, sometimes referred to as the static versus dynamic structure, was used, among others, by Ivar Jacobson [Jacobson 2004].

Both method content and process content can be reused. Process contents are organized into reusable pieces of process called capability patterns. Here we can make an analogy to the way reuse is accomplished in EPF-Composer and object orientation: method content reuse is similar to object reuse while capability patterns are like reuse through design patterns.

You select the process and underlying method content that fits your needs by browsing the method library. Once you find one that matches your need, you start configuring the situational method by selecting or deselecting the components.

**5.3. Requirement # 3 – The tool must allow the tailoring of methods** (+++)

Tool # 1 - EPF-Composer

EPF-Composer provides some pre-defined methods, e.g. OpenUP/Basic, Extreme Programming, and Scrum that can be used "as is" or as a base for tailoring to individual needs. In many situations, all that a method engineer needs is selecting, configuring, and tailoring the existing content to produce a situational method without the need of authoring method content.

Through the concept of variability, we can change existing content without directly modifying the original. There are four types of method content variability in EPF-Composer (contributes, extends, replaces and extends & replace) and each one, in some way, overrides the content from the method base maintaining the original content intact.

## 6. Conclusions

In this paper, we have presented a catalog of requirements for software process modeling tools. This catalog can be used, not only, by method engineers but also for software development teams, e.g. project manager, architects, instructors, software developers. This catalog is not a definite one. We suggest the review and update of it from time to time as computer technology trends evolves.

To be continued and improved…

## 7. References


Ågerfalk, P. J. and Fitzgerald, B. (2006) "Exploring the Concept of Method Rationale: A Conceptual Tool for Method Tailoring", In: Advanced Topics in Database Research, pages: 63-78.

Agile (2009) "Manifesto for Agile Software Development", http://agilemanifesto.org , Last access on march, 2009.

Aydin, M. N., Harmsen, F., Slooten, K. V. and Stegwee, R. A. (2005). "On the Adaptation of an Agile Information Systems Development Method", In: Journal of Database Management. 16, 4, pages: 24-40.

Bajec, M., Vavpotic, D. and Krisper, M. (2007) "Practice-driven approach for creating project-specific software development methods", In: Information and Software Technology. pages: 345–365.

Baskerville, R. and Stage, J. (2001) "Accommodating emergent work practices: Ethnographic choice of method fragments", In: Realigning Research and Practice in IS Development: The Social and Organisational Perspective. Pages:11-27.

Beck, K. (2004), Extreme Programming Explained: Embrace the Change, 2nd Edition. Addison-Wesley.

Boehm, B. (1996) "Anchoring the Software Process", In: IEEE Software 13(4), Pages: 73-82

Brinkkemper, S. (1996) "Method Engineering: Engineering of Information Systems Development Methods and Tools", In: Information and Software Technology, 38, pages: 275-280.

Combemale, B., Crégut, X., Caplain, A. and Coulette, B. (2006) "Towards a rigorous process modeling with SPEM", In: 8th International Conference on Enterprise Information Systems: Databases and Information Systems. Pages: 530-533.

Eclipse(2009) "Eclipse Process Framework Home Page", http://www.eclipse.org/epf/. Accessed on March 22, 2009.

Gonzalez-Perez, C. and Henderson-Sellers, B. (2008), Metamodelling for Software Engineering, publisher: Wiley.

Henderson-Sellers, B., Serour, M., McBride, T., Gonzalez-Perez C. and Dagher L. (2004) "Process Construction and Customization", In: Journal of Universal Computer Science, vol. 10, no. 4, Pages: 326-358.

Henderson-Sellers, B., Gonzalez-Perez, C. and Ralyté, J. (2008), "Comparison of Method Chunks and Method Fragments for Situational Method Engineering", In: Proceedings of the 19th Australian Conference on Software Engineering. Pages: 479-488.

Hoffmann, M., Kuhn, N., Weber, M. and Bittner, M. (2004) "Requirements for Requirements Management Tools", In: Proceedings of the 12th IEEE International Requirements Engineering Conference. Pages: 301-308.

Humphrey, W. S. (1995), A discipline for Software Engineering. Addison-Wesley.

Jacobson, I., Rumbaugh, J. and Booch, G. (1999), The Unified Software Development Process, Addison-Wesley.

Jacobson, I. (2004), Object-Oriented Software Engineering: A Use Case Driven Approach, Addison-Wesley.

ISO/IEC (2007) "ISO/IEC 24744 - Software Engineering Metamodel for Development Methodologies".

ISO/IEC (1991) "ISO/IEC 9126: Information Technology - Software Product Evaluation - Quality Characteristics and Guidelines for Their Use".

Karlsson, F. (2008) "A wiki-based approach to method tailoring", In: Proceedings of the 3rd International Conference on the Pragmatic Web: Innovating the Interactive Society, pages : 13-22.

Krutchen, P. (2003), The Rational Unified Process – An Introduction, 2nd Edition. Addison-Wesley.

Mäkilä, T., Järvi, A. (2006) "Spemmet–A Tool for Modeling Software Processes with SPEM", In: Proceedings of the 9th International Conference on Information Systems Implementation and Modelling.

Nardini, E., Molesini, A., Omicini, A. and Denti, E. (2008) "SPEM on test: the SODA case study", In: Proceedings of the 2008 ACM Symposium on Applied Computing. Pages: 700-706

Neto, P. S., Resende, R. and Pádua, C. (2007) "Requirements for Information Systems Model-Based Testing", In: Proceedings of the 2007 ACM symposium on Applied computing. Pages: 1409–1415.

OMG (2006) “Meta Object Facility (MOF) Core Specification”, v2.0. Formal/2006-01-01, http://www.omg.org/docs/formal/06-01-01.pdf .

OMG (2008) “Software & Systems Process Engineering Meta-Model Specification”, v2.0. Formal/2008-04-01, http://www.omg.org/docs/formal/08-04-01.pdf.

OMG (2009) “Unified Modeling Language: Superstructure”, Version 2.2. Formal /2009-02-02, http://www.omg.org/docs/formal/09-02-02.pdf

Osellus (2009) “Osellus Home Page.” http://www.osellus.com/. Accessed on March 26, 2009.

Paula Filho, W. P. (2009), Engenharia de Software: Fundamentos, Métodos e Padrões, LTC, 3th edition. Rio de Janeiro, Brazil (in Portuguese).

Pfleeger, S.L. and Atlee, J. M. (2005), Software Engineering: Theory and Practice, Prentice Hall, 3th edition.

Pressman, R. S. (2001), Software Engineering: A practitioner´s Approach, McGraw-Hill, 5th edition.

Ralyte, J., Deneckere, R. and Rolland, C. (2001) “An assembly process model for method engineering”, In: Advanced Information Systems Engineering, pages: 267-283.

Ralyte, J., Deneckere, R. and Rolland, C. (2003) “Towards a Generic Model for Situational Method Engineering”, In: Lecture Notes in Computer Science (Proc. Of CAiSE’2003), volume 2681, pages: 95–110.

Rational(2009) “Rational Method Composer Home Page;” http://www-01.ibm.com/software/awdtools/rmc/index.html. Accessed on March 22, 2009.

Schwaber, K. (2004), Agile Project Management with Scrum, Microsoft Press.

Sommerville, I. (2006), Software Engineering, Addison-Wesley, 8th edition.