

Software Process Modeling with
Eclipse Process Framework and SPEM 2.0

Henrik Terävä

hemite@utu.fi, 55954

Master's Thesis
October 3rd, 2007

Supervisors

Tuomas Mäkilä, University of Turku
Juha Jääskinen, SYSOPENDIGIA

UNIVERSITY OF TURKU
DEPARTMENT OF INFORMATION TECHNOLOGY
FI-20014 TURKU
FINLAND

UNIVERSITY OF TURKU
Department of Information Technology

TERÄVÄ, HENRIK: Software Process Modeling with Eclipse
Process Framework and SPEM 2.0

Master's Thesis, 94 pages
Software Engineering
October 2007

The software development industry is constantly evolving. The rise of the agile methodologies in the late 1990s, and new development tools and technologies require growing attention for everybody working within this industry. The organizations have, however, had a mixture of various processes and different process languages since a standard software development process language has not been available.

A promising process meta-model called Software & Systems Process Engineering Meta-Model (SPEM) 2.0 has been released recently. This is applied by tools such as Eclipse Process Framework Composer, which is designed for implementing and maintaining processes and method content. Its aim is to support a broad variety of project types and development styles.

This thesis presents the concepts of software processes, models, traditional and agile approaches, method engineering, and software process improvement. Some of the most well-known methodologies (RUP, OpenUP, OpenMethod, XP and Scrum) are also introduced with a comparison provided between them. The main focus is on the Eclipse Process Framework and SPEM 2.0, their capabilities, usage and modeling.

As a proof of concept, I present a case study of modeling OpenMethod with EPF Composer and SPEM 2.0. The results show that the new meta-model and tool have made it possible to easily manage method content, publish versions with customized content, and connect project tools (such as MS Project) with the process content. The software process modeling also acts as a process improvement activity.

Keywords: Software Process Modeling, Software Processes, Method Engineering, Software Process Improvement, Methodologies, RUP, SPEM, Eclipse Process Framework

TURUN YLIOPISTO
Informaatioteknologian laitos

TERÄVÄ, HENRIK: Ohjelmistoprosessin mallinnus Eclipse Process Frameworkilla
ja SPEM 2.0 metamallilla

Diplomityö, 94 sivua
Ohjelmistotekniikka
Lokakuu 2007

Ohjelmistot ja ohjelmistoteollisuus kehittyvät jatkuvasti. Ketterien menetelmien tulo 1990-luvun loppupuolella, uudet kehitystyökalut ja teknologiat vaativat yhä enemmän huomiota alalla työskenteleviltä ihmisiltä. Organisaatioilla on kuitenkin ollut sekalainen kirjo prosesseja ja erilaisia prosessikuvauskieliä, koska standardia kuvauskieltä ei ole ollut saatavilla.

Prosessimetamalli SPEM 2.0 julkaistiin hiljattain. Tätä mallia hyödyntää mm. Eclipse Process Framework Composer (EPFC) –työkalu, joka on suunniteltu prosessien ja menetelmäsisällön kehittämiseen ja ylläpitoon. Työkalun tavoitteena on tukea useita erilaisia projektityyppejä ja kehitystyyliä.

Tässä työssä esitellään seuraavat aiheet ja käsitteet: ohjelmistoprosessit, mallit, perinteiset ja ketterät lähestymistavat, metoditekniikkaa sekä prosessien kehittäminen. Lisäksi tutustutaan muutama tunnetuimmista metodologioista (RUP, OpenUP, OpenMethod, XP ja Scrum) ja vertaillaan näitä. Työssä tutkitaan tarkemmin Eclipse Process Framework Composer –työkalua, SPEM 2.0 metamallia, näiden ominaisuuksia, käyttöä sekä mallintamista.

Esitän tutkimustulokset ja tutkimuksenkulun OpenMethodin mallintamisesta EPFC –työkalulla sekä SPEM 2.0 -metamallilla. Tulokset osoittavat, että uusi metamalli ja työkalu helpottavat prosessin ja menetelmäsisällön hallintaa, mahdollistavat räätälöityjen julkaisujen teon sisällöstä, sekä yhdistävät prosessin projektityökaluihin kuten MS Projectiin. Mallinnus voidaan lisäksi ymmärtää osana prosessin kehittämistä.

Avainsanat: Ohjelmistoprosessin mallinnus, ohjelmistoprosessit, metodikehitys, ohjelmistoprosessien parantaminen, metodologiat, SPEM, Eclipse Process Framework

List of Figures

Figure 2.1.1 – The role of the software process [Elv06]	4
Figure 2.1.2 – Two conceptual models of processes: A specific process and a generalized one with serial and parallel activities [Joh04]	5
Figure 2.1.3 – The relationships between Role, WorkProduct and Task [SPEM2.0]	7
Figure 2.2.1 – The V Model [Gra02].....	9
Figure 3.1.1 – The method engineering process [Elv06].....	16
Figure 3.3.1 – The IDEAL model – a traditional SPI cycle with five phases [SEI07].....	20
Figure 3.3.2 – Productivity vs. time in process improvement [Wie05].....	21
Figure 3.3.3.1 – Relationships in Process Assessment [SPICE]	24
Figure 3.3.3.2 – The framework for process assessment [Ele98].....	25
Figure 4.1.1 – The Rational Unified Process – Phases and Disciplines [San07]	28
Figure 4.4.1 – A general overview of a typical XP process [Ram06]	34
Figure 4.5.1 – The Scrum process [Ram06]	36
Figure 4.6.1 – A comparison of methodologies in flexibility and iterativity [Hai07].....	37
Figure 5.1.1 – Process modeling layers [Jär05, Kos99, SPEM 2.0].....	42
Figure 5.1.2 – Structure of the SPEM 2.0 Meta-Model [SPEM2.0].....	43
Figure 6.3.1 – EPF with Wiki technology [Kro07b]	54
Figure 7.1.1 – Modeling approach modeled with EPF Composer 1.2 [Mäk07].....	61
Figure 8.2.1 – Working hours spent on thesis (2007).....	70
Figure 8.3.1 – The original OpenMethod opening page (in Finnish).....	71
Figure 8.3.2 – OpenMethod opening page modeled with EPF Composer (Finnish).....	71
Figure 8.3.3 – A part of the EPFC library view for the OpenMethod content	72
Figure 8.3.4 – EPFC library view of the Standard and Custom Categories	73
Figure 8.3.5 – EPFC library view of the Processes and Configurations	74

List of Tables

Table 2.4.1 – A comparison of traditional and agile development [Ner05]	12
Table 4.6.1 – A comparison of methodologies [Abr02, SD07, Ram06, Lar03, Ter06]	39
Table 5.1.3 – Some essential SPEM 2.0 icons with short explanations	44

Acronyms and Abbreviations

BPEL	Business Process Execution Language
BUP	Basic Unified Process
CASE	Computer Aided Software Engineering
CAME	Computer Aided Method Engineering
CMMI	Capability Maturity Model Integration
CVS	Concurrent Versions System
DSDM	Dynamic Systems Development Method
EPF	Eclipse Process Framework
EPFC	Eclipse Process Framework Composer
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Organization for Standardization
ISM	Issue Management System
ITIL	Information Technology Infrastructure Library
KM	Knowledge Management
MDA	Model Driven Architecture
MDE	Model Driven Engineering
ME	Method Engineering
MSF	Microsoft Solutions Framework
NLS	Native Language Support
OMG	Object Management Group
OPENUP	Open Unified Process
OPF	Open Process Framework
PMBOK	Project Management Body of Knowledge
RMC	Rational Method Composer
RUP	Rational Unified Process
SME	Situational Method Engineering
SPEM	Software & Systems Process Engineering Meta-Model
SPI	Software Process Improvement
SPICE	Software Process Improvement and Capability dEtermination
UMA	Unified Method Architecture
UML	Unified Modeling Language
UPM	Unified Process Model
XP	Extreme Programming

Definitions

CAME tool, software which aids in method engineering.

CASE tool, software which helps in software development, used for standardization and normalization by developers.

Framework, provides the underlying structure for development, used when exact instructions on how to do things are not given. A framework can be used to develop methodologies and it should be instantiated to yield a process [Rub05]. Often used interchangeably with the term ‘methodology’. [Ram06, Kni07]

Method, a systematic process or technique that is used to aid in the creation of a satisfactory software product or model [Elv06]. Method prescribes a way of performing activity within a process, in order to properly produce a specific output starting from a specific input [Mol06]. For example a recipe is a method for cooking a specific dish.

Method Engineering, the engineering discipline to design, construct, and adapt methods, techniques and tools for the development of information systems [Ram06].

Software Process Engineering, consists of modeling, authoring, tailoring and enacting processes [Jär05].

Methodology, a body and collection of methods meant to support all software development phases. [Elv06] It is a prescriptive guidance which covers and connects different stages in a process. The purpose of a methodology is to prescribe a certain coherent approach to solving a problem in the context of a software process. [Mol06] Methodology offers not only indications of what to do, but includes actual recipes of how to do [Rub05]. This term is often considered too long and formal, and the term ‘method’ is used instead. Examples include OpenMethodTM and XP.

Process, a sequence of actions leading to some result [Elv06]. A set of related activities which together transform inputs into outputs to achieve a given objective. The process is defined in more detail in section 2.1.

Software Development Process, a coherent set of policies, organizational structures, technologies, procedures and deliverables that are needed to conceive, develop, deploy and maintain a software product [Mol06]. Software development process is often used to refer the term ‘software development process model’.

Software Development Process Model, prescribes a process organization around phases, in which order phases should be executed and when interactions and coordination between the work of the different phases should occur. In other words, a process model defines a template, around which to organize and detail an actual process. [Mol06]

Wiki technology, the defining characteristic is the ease with pages can be created and updated in real-time. These web pages are used as knowledge bases and appear almost instantaneously online. The term ‘wiki’ derives from Hawaiian term ‘wiki wiki’, meaning ‘quick’ or ‘informal’. [Wik07, Mal05]

Acknowledgements

This thesis work was started in the turn of the years 2006-2007. I was lucky to get a topic which was interesting for me and which would probably interest both the university and the company I was working in. The University of Turku has been specializing in software processes and they also had a TEKES project planned, which was also discussed. Within SYSOPENDIGIA there was an interest concerning research on OpenMethod, and I want to thank Carl-Eric Backman and the company for providing me with the work.

Many thanks go to Juha Jääskinen for being the supervisor from the company and for providing me with all the support I needed. He has been active and spurring me to get the work done well. From the university, I want to thank my supervisor, Tuomas Mäkilä, who has also been active and promoting the academic thinking in the thesis. I also want to thank Peter Haumer for giving me suggestions and improvement ideas.

I want to thank my family for their support and encouragement. I also want to thank people who have been proofreading my text. Finally I am very thankful to all others who have been helping me with the thesis and who have been living with me during the thesis work. Special thanks go to my chipmunk Indy, who has disturbed me now and then, and thus hopefully has given me new ideas and points of view.

Contents

1. Introduction.....	1
2. What is software process modeling?	3
2.1 What is a software process and a model?.....	4
2.2 Traditional processes.....	8
2.3 Agile processes.....	10
2.4 Comparison of traditional and agile processes.....	12
2.5 Why model a software process? Benefits and drawbacks	13
3. Software process modeling and method engineering	16
3.1 Process construction.....	16
3.2 Selection of a process.....	18
3.3 Evaluation and improvement of a process	20
3.3.1 Knowledge Management	22
3.3.2 CMMI.....	22
3.3.3 SPICE.....	24
4. Software development methodologies.....	26
4.1 Rational Unified Process	26
4.2 OpenUP	30
4.3 OpenMethod™.....	32
4.4 Extreme Programming.....	33
4.5 Scrum.....	35
4.6 Comparison of RUP, OpenUP, OpenMethod™, XP and Scrum.....	37
5. Models and tools.....	40
5.1 SPEM 2.0.....	41
5.2 Eclipse Process Framework.....	45
5.3 Rational Method Composer.....	47
6. Modeling capabilities and usage of EPF and SPEM	48
6.1 Why model existing models with EPF and SPEM?.....	48
6.2 How was modeling done before EPF and SPEM?.....	50
6.3 How have EPF and SPEM been used?.....	52
6.4 Capabilities of EPFC 1.2.....	54
6.5 Sufficiency of SPEM 2.0.....	56
7. Modeling process based on an existing model with EPF / SPEM	60
7.1 Modeling with EPF and SPEM 2.0	60
7.2 EPF customization and tailoring	62
7.3 Process enactment	64
7.4 Benefits and problems experienced with modeling	65
8. Modeling OpenMethod™ with EPF and SPEM 2.0.....	68
8.1 Why model OpenMethod with EPF and SPEM?.....	68
8.2 Modeling time table.....	69
8.3 Modeling OpenMethod	70
8.4 Modeling results	76
8.5 User interviews and discussion	77
9. Conclusions and Summary	80
References.....	82
Appendix A: SPEM icons with explanations.....	91
Appendix B: Citations	94

1. Introduction

Software development methodologies are nowadays regularly used in the software development industry. The various different types of methodologies can be applied to different cases. A useful process modeling language standard exists, and the first tools based on this standard are now available.

What are the software processes and methodologies actually, and how can they be used to improve software development? How can the process itself be modeled, and why should it be done? What benefits and limitations have the current modeling language standards and tools? Along with these questions, I will seek answers to the main research question:

How will the Eclipse Process Framework and SPEM 2.0 improve process modeling and model usage?

In addition to the academic research question, this thesis is also a part of *OpenMethod™* research and development. This work includes providing profitable process content and best practices supporting iterative, incremental and agile development. This should be applicable to a wide set of development platforms and applications.

The case study includes studying the tool support for software process engineering including the following: method and process authoring, library management of plug-ins and configurations, efficient and practical CVS repository. The focus is mainly on customization and publishing a process.

As there are various meta-models and methodologies, the focus is on *SPEM 2.0* as it is the most recognized as a standard and though it is methodology independent, it seemed to be the best choice for *Rational Unified Process* –style methodologies such as *OpenMethod*. *Eclipse Process Framework Composer (EPF)* is the most advanced and productized non-commercial method and process authoring tool at the moment. As alternative study topics, for example the public-domain *OPEN Process Framework*, or the commercial *Microsoft Solutions Framework* with *Visual Studio Team System* would be different and interesting, but are not included due to the limitations of this thesis.

The limitations of this study are the following:

- **Focus on EPF and SPEM:** even if there are other meta-models and frameworks available, the scope is not widened, but rather concentrated around EPF and on SPEM compliant methodologies.
- **EPF under development:** the development in EPF community is rapid; a detailed discussion based on a specific version could be obsolete later on.
- **Limited amount of modeling and interviews:** the modeling is based on a specific part of OpenMethod and the interviews were conducted in a short time.

In this thesis, the second chapter contains the general ideas behind software process modeling and different life cycle models, including benefits and drawbacks. An example of traffic processes are introduced as an analogy to software development processes. This idea can be followed throughout the thesis. In the third chapter, we focus on method engineering and software process improvement. In the fourth chapter, we study software development methodologies and take a closer look at *Rational Unified Process (RUP)*, *OpenUP*, *OpenMethod*TM, *Extreme Programming (XP)* and *Scrum*. The fifth chapter includes the meta-model *SPEM 2.0* and the tools *Eclipse Process Framework (EPF)* and *Rational Method Composer (RMC)*, which are used for modeling software processes. In the sixth chapter, we discuss the modeling capabilities of EPF and SPEM, and in the seventh chapter, the focus is on modeling an existing model with EPF and SPEM. Modeling of OpenMethod is performed in chapter eight. The last chapter has the conclusions and summary. Several future work topics are also introduced.

2. What is software process modeling?

First, we study the definition of a software process, the role of the software process and the model of a software process. Then, traditional disciplined and agile approaches are introduced with a comparison of the approaches. Finally, we discuss the benefits and drawbacks of modeling. Methodologies, such as Rational Unified Process, OpenUP, OpenMethod, Extreme Programming and Scrum are introduced in chapter four in more detail, followed by a comparison of the methodologies.

The main division can be made between *traditional disciplined processes* and *agile processes* [Boe03]. This division is done mainly to ease comprehension and categorization as in practice both styles are often accommodated. This division does not only mean the development model (e.g. waterfall vs. evolutionary), but also the whole mindset – the approach to development, relationships inside the team and with the customer, technology used and organization structure and culture. Traditional processes have their origins in general engineering, but the agile approach exists only within software development [OT07].

Agile processes contain only what developers consider valuable. They emphasize flexibility and responsiveness to changes. Traditional processes are executed outside the developer's scope and they emphasize thorough planning and control. Usually, a mixture and balance of both styles is accommodated in a process. Standard processes are tailored according to the customer, environment and project needs. The development staff then uses this process with development tools for the project work. [SSE07]

While traditional processes continue to dominate the software development arena, several opinions and surveys demonstrate the growing popularity of agile processes. Organizations should carefully evolve toward the best balance of agile and traditional methods that fits their situation. [Ner05]

2.1 What is a software process and a model?

The software process ties people and technology together to develop software products in a specific environment, as depicted in Figure 2.1.1 [Elv06]. Environment can include business activity and atmosphere, one's own organization, other organizations, etc. [SD07]

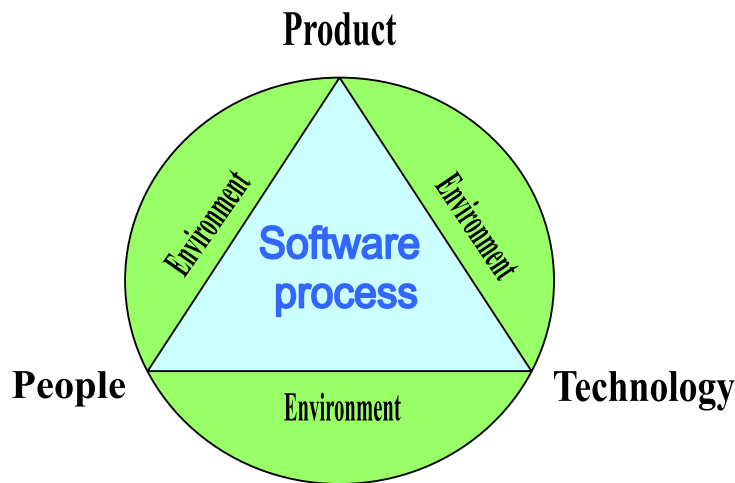


Figure 2.1.1 – The role of the software process [Elv06]

Now we will study the definitions of a process, software engineering and software development process. The definitions are based on *ISO* and *IEEE standards*.

A process has many definitions, for example

- A set of interrelated or interacting activities which transform inputs into outputs. [ISO 9000]
- A consecutive series of interrelated actions (steps) which together transform inputs into outputs for a given purpose. [ISO 8402]
- A partially ordered set of activities that can be executed to realize a given objective or to achieve some desired result. [ISO 15504]
- A sequence of steps performed for a given purpose; for example, the software development process. [IEEE 610.12]

If we combine these three we get

A set of related activities which together transform inputs into outputs to achieve a given objective.

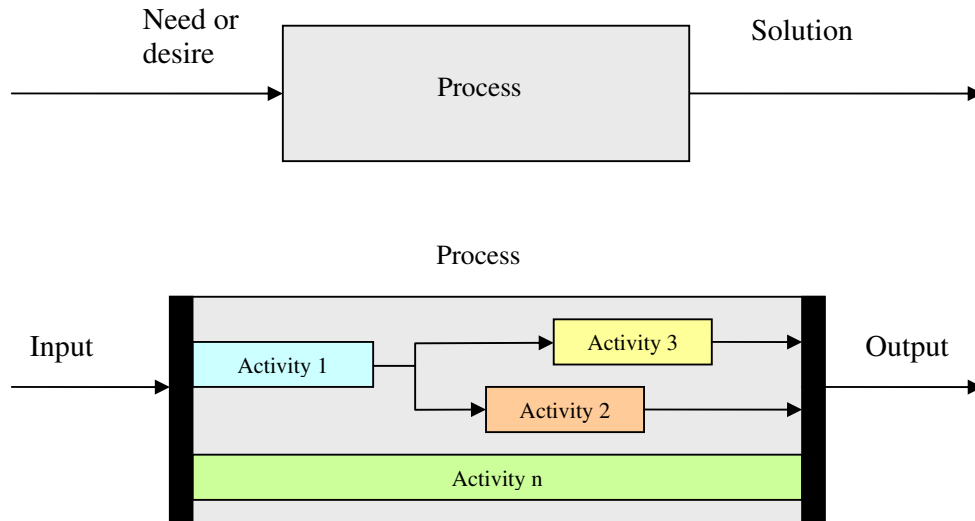


Figure 2.1.2 – Two conceptual models of processes: A specific process and a generalized one with serial and parallel activities [Joh04]

In Figure 2.1.2 the basic model of a process is shown. A process can be further divided into four elements: *who* does *what*, *how* and *when* [Hen06b]. Let us next introduce some more extensive definitions.

Software Engineering is defined by IEEE 610.12 as

“The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.”

However, it is good to keep in mind that software engineering is done “of the people, by the people and for the people” [Boe03]. Or as said by Bjarne Stroustrup (1991): “Design and programming are human activities; forget that and all is lost.”

Now a *software development process* is defined by IEEE 610.12 as

“The process by which user needs are translated into a software product. The process involves translating user needs into software requirements, transforming the software requirements into design, implementing the design in code, testing the code and sometimes, [*sic*] installing and checking out the software for operational use. Note: These activities may overlap or be performed iteratively.”

The sequencing of phases and milestones to express a lifecycle is one of the common characteristics found within process definitions. Processes define work sequences that use time and other resources. Work sequences produce outcomes. Processes also define roles behind work sequences and outcomes.

To clarify and to give an example of process and model thinking, we will now introduce an analogy between software development processes and traffic processes.

Traffic has rules of its own. These traffic instructions could be thought as traffic models, which provide the fundamental rules on how to drive a vehicle. However, if a driver is only given a vehicle with a traffic instructions book and put into traffic without any experience, it would be extremely hard to manage with others. For a successful adaptation to traffic, proper training, perhaps mentors or even exams are required, which is also the case in software development.

The rules may also be differently applied in different situations. In some traffic cultures, the actual traffic process does not follow the models or rules, but everyone knows (e.g. has the tacit knowledge) how to behave. The instructions might also differ in language, and a change from one driving culture to another will not usually be straightforward. The infrastructure can be thought as architecture, cities as systems connected by roads and railroads as interfaces. A method would be a way of transport (train, car, airplane) and a methodology a collection of these.

In order for traffic to be efficient, no one should neglect the main rules given. If the rules are not followed, is the fault with the rules? However, flexibility and situational adaptation is required in both traffic and software development.

A software process can be presented in natural language, by graphical descriptions without formality, and by process modeling languages [SSE07]. This is modeling and results as in software process model. A model is an abstract representation of a system that helps us to answer questions about the system [OOSE04]. Next, we will take a brief look at the model definitions of one software process modeling language, Software Process Meta-Model (SPEM) 2.0. In section 5.1, SPEM 2.0 is studied in more detail.

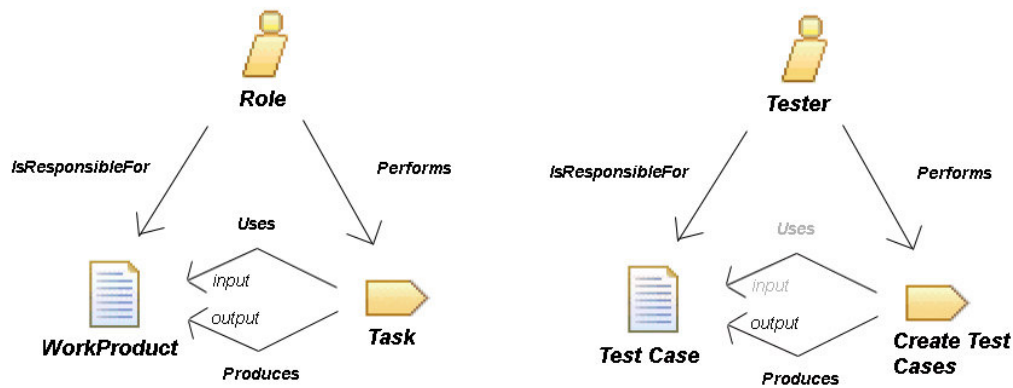


Figure 2.1.3 – The relationships between Role, WorkProduct and Task [SPEM2.0]

The basic building elements in SPEM are Role, WorkProduct and Task, which are often accompanied by Guidance. Roles perform Tasks and are responsible for WorkProducts. Tasks use WorkProducts as input and also produce WorkProducts as output. A static illustration of relationships between these three elements using SPEM icons is shown in Figure 2.1.3 with an example instantiation. These static elements can be applied to dynamic content, e.g. processes. [SPEM2.0]

Process modeling specifies a development process and is the base for any process engineering activities. However, any modeling approach is constrained by a certain way of thinking and of perceiving systems and software development [Kos99]. In addition, stakeholders' views, assumptions and beliefs constrain process approach. Different methods and techniques are applicable in different process contexts. The term *practice model* is used when describing the way software development is really done. The detection and comparison of the differences between practice and process models gives important information when validating the software development process. [Moo06]

A process model can be descriptive, prescriptive or proscriptive. *Descriptive modeling* tries to make the currently used processes explicit. *Prescriptive modeling* specifies the recommended way of executing the process. *Proscriptive modeling* describes non-allowed behavior and is usually used as an addition to prescriptive or descriptive modeling. [Kos99]

Interaction and engineering with models and system is called *Model Driven Engineering (MDE)*, and it can be divided into three types [Bez07]: *forward engineering*, *reverse engineering*, and *models at run-time*. In forward engineering, the system is created from a model; in reverse engineering, the model is created from a system; and in models at run-time, the model and system coexist and are kept synchronized.

In chapters seven and eight, we study the modeling process based on an existing model. It is important to understand the difference between this and the creation of a real-life work process model, e.g. reverse engineering.

In general, software development is a complex activity characterized by tasks and requirements with a high degree of variability. Further uncertainties arise with the diversity and unpredictability of people who engage in such tasks. The continuously changing environment and sophistication of the tools also present development problems. [Ner05]

2.2 Traditional processes

A software life cycle is a period which lasts from the beginning of development to the end of software use [Hai00]. A life cycle model is a way in which development is divided into phases and represents all the activities and work products necessary to develop a software system. [OOSE04]

The *waterfall model* is a linear life cycle model. The development is sequential, like a waterfall, with phases of requirements, analysis, design, implementation, testing, integration and maintenance. After each phase, the project is carefully examined by different parties before proceeding to the next phase. This is intended to eliminate the faults in early phases of the project. [Kil07]

In an *iterative* approach, the process consists of many iterations – each containing similar tasks that are repeated. The last iteration produces the final product. If the iterations produce a partial product and add new functionality to the system, it is *incremental* [OT07]. This approach thus comprises of some agile aspects.

The *spiral model* is a combination of a linear and iterative model. The development is divided into four phases which are executed to continuously expand the project. The phases are: determining objectives, evaluation and identifying risks, implementation and testing, and planning the next iteration. [Kil07]

The *V Model* gives equal weight to testing rather than simply treating it afterwards. It emerged as a reaction to some waterfall models, which show testing only as a single phase, though testing could take up to half of the project time [Gra02]. The V Model highlights the existence of several levels of testing by relating it to a specific development phase (Figure 2.2.1). The V Model depicts the level of abstraction in the following way:

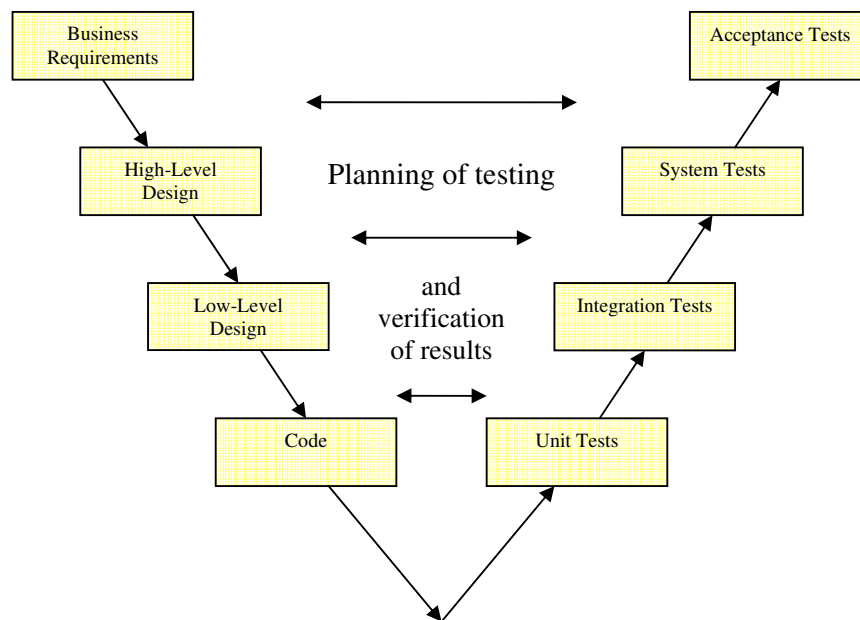


Figure 2.2.1 – The V Model [Gra02]

As we have now introduced some of the well-known traditional models, we can widen the scope of analysis to the whole approach and way of thinking. The traditional approaches promise predictability, stability and high assurance [Boe03]. Around the models, specific

focuses and styles are accommodated in the fields of application, management, technology, and personnel. They tend to be more mechanistic, formal and controlled, like the models. A detailed comparison between traditional and agile approaches is shown in Table 2.4.1.

2.3 Agile processes

The introduction of *Extreme Programming (XP)*, see section 4.4) has been widely acknowledged as the starting point for the various agile software development approaches. The need in the business community for lighter weight, faster and nimbler software development processes, especially in the Internet software industry, was one of the driving forces. [Abr02]

Although there is no agreement on what the concept of ‘Agile’ actually refers to, here are some definitions and meanings associated with the term agile:

- the quality of being agile; readiness for motion; nimbleness, activity, dexterity in motion [Abr02],
- quick and well-coordinated in movement; characterized by quickness, lightness, and ease of movement; nimble, able to improvise [Dict07, Lev05],
- adaptable; responsive to a customer or to changing circumstances [Lev05],
- resourceful; thoughtful or exhibiting some discipline [Lev05].

The ‘Agile Movement’ in software industry saw the light of day with the *Manifesto for Agile Software Development*, in which the developers or ‘agilists’ came to value the following: [AGILE07]

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

“That is, while there is value in the items on the right, we value the items on the left more.”

At start of the agile movement words like ‘process’ or even ‘methodology’ were considered almost dirty, but currently the agile methodologists are showing growing interest in advertising their agile processes and methodologies. [Ram06]

According to Abrahamsson [Abr02], a software development method is agile when it is

- **incremental**, small software releases with rapid cycles,
- **cooperative**, good communication with customers and developers working together,
- **straightforward**, the method is efficient, and easy to learn and modify,
- **adaptive**, has the ability to make last-moment changes with quick response.

Agile development generally uses the *evolutionary development model (EVO)*. In EVO, the development cycle is divided into smaller, incremental phases in which the customer is able to gain access to the product at the end of each cycle. The EVO model can also be understood as a sequence of repeating small waterfalls [Hai00]. The development team responds to customer feedback by changing the product, plans or process for the next phase.

Agile methodologies (such as Scrum and XP, introduced in sections 4.4 and 4.5) are ideal for projects with high variability in tasks, in the capabilities of people (because for example pair programming combines expertise and educates beginners effectively) and in the technology being used. They are also suitable for projects where the high value of the product is important to customers in tradeoff with accurate plans and timetables. Agile methodologies usually tend to suit small teams (4-10 members) better and the critical issue for success is especially the developers’ valuing of and trust in each other. [Ner05]

As previously mentioned, choosing an agile methodology itself is not enough for the whole development process to be considered agile. Organizations must renew their mindset – rethink their goals and reconfigure their human, managerial, and technology components in order to successfully adopt agile methodologies.

2.4 Comparison of traditional and agile processes

	Traditional	Agile
Fundamental Assumptions	Systems are specifiable, predictable, and can be built through precise and extensive planning	Quality, adaptive software can be developed by small teams through continuous design improvement and testing based on rapid feedback and change
Desired Organizational Structure	Mechanistic (bureaucratic with high level of formalization)	Organic (flexible and participate, encouraging cooperative social action)
Management Style	Command and control	Leadership and collaboration
Control	Process centric	People-centric
Knowledge Management	Explicit, produces large amount of documentation	Tacit, relies on teamwork and is in the heads of the team members; documentation as necessary
Development Model	Life cycle model (Waterfall, or some variation)	The evolutionary-delivery model (XP, Scrum...)
Project Cycle	Guided by tasks or activities	Guided by product features
Role Assignment	Individual, favors specialization	Self-organizing teams, encourages role interchangeability
Communication	Formal, with documents	Informal
Customer's Role	Important, focus during specification	Critical, like a team member
Technology	No restrictions	Favors object-oriented technology

Table 2.4.1 – A comparison of traditional and agile development [Ner05]

A general comparison with the traditional and agile approach is shown in Table 2.4.1. This table can also be used with Table 4.6.1 ‘A comparison of methodologies’ when evaluating the level of agility in methodologies. The division is, however, often superficial and the actual approaches usually apply both the traditional and agile elements.

2.5 Why model a software process? Benefits and drawbacks

Software development is about processes. Each step and pattern of the process is more or less visible depending on how explicitly it is defined and on the level of its formality, role and detail. The work process is defined by its environment, work culture, work practices, tools etc. Every development team follows a process. [SSE07, Bec03]

The whole process becomes visible by creating a model of the process. This defined process is the requirement for process understanding, analysis, execution guidance, learning and communication. It will also support the improvement of the process. However, typically companies have only a single, generic process for any project. Process variation has not been studied except in bigger companies [Jär06].

There are various benefits to software process modeling and we will examine some of them. Most references focus on the bright side of modeling, but we will also discuss the drawbacks. Benefits and drawbacks introduced here should also be noticed when focusing on the specific benefits and drawbacks of EPF and SPEM modeling in section 6.1.

In [Oja03] Ojala introduces the following benefits of software process modeling:

- align the whole process into manageable divisions
- standards, guidance and models support development in each phase
- models are tools and act as templates for project planning
- reduce the person dependency
- ease communication for people and other systems
- make working more efficient and faster
- increase quality, reliability, compatibility, usability and maintenance
- models and guidance help with training

Most of the arguments seem logical, but we can still ask how the models make communication easier. How will they make working faster? Obviously communications will be easier, if the language and terms used are coherent within the team and with other teams. However, will the models instead simply be in the way and slow down working?

This is possible in some cases, so the models cannot be seen as a cure for every problem without reasoning the case. Let us now study some benefits that Henderson-Sellers [Hen04a] depict which were not introduced before:

- ensure a consistent, reproducible approach to all projects providing a uniform approach to software development with guidance
- control and support the whole development life with good project planning and management
- consistency and traceability though the whole life cycle
- emphasize analysis and understanding through customer involvement
- reduce risk associated with shortcuts and mistakes
- produce complete and consistent documentation from one project to the next
- gain flexibility to process tailoring and managing and supporting different types of projects

Some benefits may slightly overlap with the previous ones, but the emphasis was here clearly on a larger scale. Haikala [Hai00] introduces the following benefits:

- easier for people to switch to the next project (or between projects) if the practices are similar to each other and the documentation is comprehensive
- when the process is anticipated, project planning and monitoring will be easier
- outside evaluation is easier for a project

Schönström [Sch05] sees the benefits from the knowledge management point of view (see section 3.3.1):

- model usage supports knowledge management creation and sharing
- models stimulate individual knowledge – new employees can use models (when they need information and guidance)
- models protect and support development. They contribute to a systematic and non-random environment
- models facilitate the sharing of individual knowledge, thus creating common ground for communication

Finally I propose some benefits which were not introduced by the previous authors:

- integrating work of from different teams and companies
- defining responsibilities between team members
- improvement possibilities and measurability as the process is defined and concrete
- the possibility to tailor, customize and export processes for certain uses
- models are part of a professional approach to software design

Since software process modeling is not only rewarding and productive, here are also some drawbacks and challenges which I found reasonable to include:

- the high cost of model creation and maintenance, is the extra work beneficial?
- the benefits of the model cannot be seen instantly
- models can make processes and daily tasks heavy if followed ‘by-the-book’
- people might resist new working ways and models
- methods and tools may narrow the development

The process enthusiasts usually only see the benefits of processes and modeling. However, the benefits obviously overcome the drawbacks even if the situation is more objectively observed. The choosing of an appropriate process becomes more important as the organizations and projects get bigger. In the following chapter, we will take a deeper look at creating a process, choosing the best process depending on the situation, and process improvement.

3. Software process modeling and method engineering

We have already introduced the basic definitions and the most common categorizations of processes. In this chapter, we study the construction of a process, method engineering, and process selection criteria. Lastly, we discuss process improvement and evaluation, knowledge management and have a brief introduction to CMMI and SPICE.

3.1 Process construction

Process modeling, and especially a standard modeling language, defines common concepts and terminology. A unified foundation for process frameworks makes it possible to compare, select and encapsulate fragments of process content from several sources. These fragments can be reused to efficiently create customized processes for desired projects with different dominant process assumptions. This is called *method engineering (ME)*, which is defined as follows: “The engineering discipline to design, construct, and adapt methods, techniques and tools for the development of information systems” [Ram06]. It has been motivated by the belief that no method fits all situations. [Jär06, Hen06]

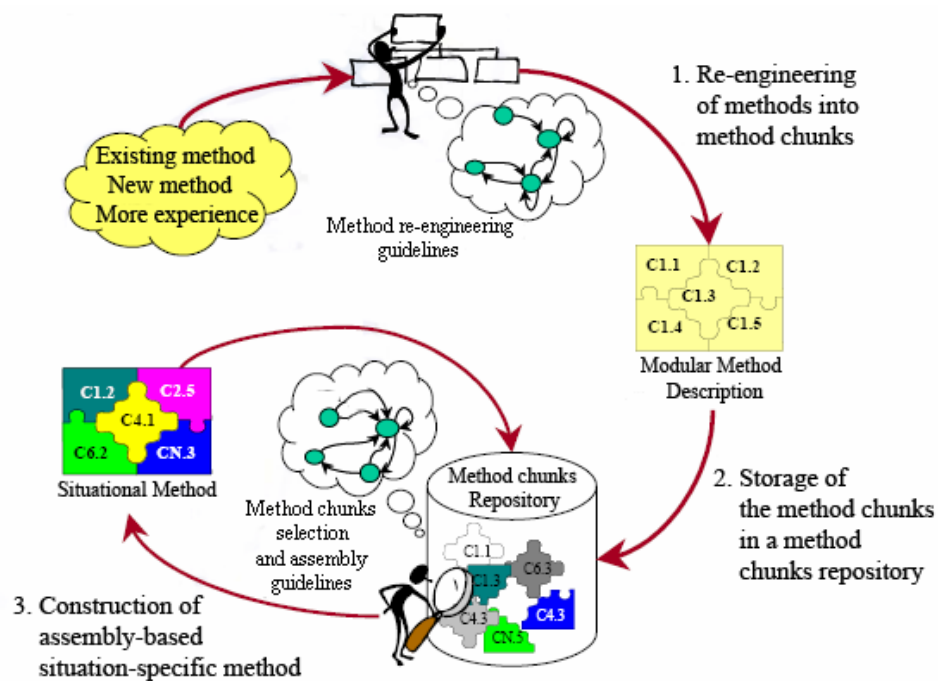


Figure 3.1.1 – The method engineering process [Elv06]

Typically a *chunk* will encapsulate both a *process part* (tasks, activities...) and a *product part* (templates, guidance...), whereas a *fragment* will be one or the other. These can be created by re-engineering methods and are stored in a method chunks repository. A method fragment is an autonomous and coherent part of a method. It supports the realization of some specific activity. A fragment can be composed of other fragments and have relationships with others [Hen06b]. A situational method is constructed ‘bottom-up’ to match the requirements of a unique project. This also exhibits flexibility as the fragments can be added dynamically. [Hen06a, Elv06] This *situational method engineering (SME)* is the most well-known subfield of method engineering (see Figure 3.1.1). Its goal is to achieve flexibility without reducing control of the development project [Jär07].

There are several method engineering approaches; the most prominent are the following:

- *Ad-hoc*: Constructing a new method from scratch
- *Paradigm-based*: Instantiating or abstracting an existing meta-model
- *Extension/Reduction-based*: Enhancing (or reducing) an existing method
- *Assembly-based*: Reusing parts of other methods [Ram06]
- *Configurative-based*: Configuring reference models in a specific domain [Bec07]

Problems with method engineering include the quality measure [Zhu07], the sophistication of the process [Hen04a], the efficiency and speed of tailoring compared to the non-tailored method [Coc00], and the correctness of the constructed method. Even if the fragments were extracted from a working method, would they also be suitable for another use? Using the traffic analogy, a car would be good to transport a few persons from a city to another city, but when transporting 50 persons, a train or even an airplane would be more reasonable depending on the available connections and distance. When planning a longer trip, different means of transportation (methods) should be used. Again, there are several variables to be considered and options to choose from, as in the software development.

The requirements for a good software development process according to Abrahamsson [Abr05] are the following: provides systematic support for high quality development (acts as a framework), produces visible results early, is easy to learn and adjustable, is aligned with the company’s strategic planning, and meets the needs of standard quality requirements (convincing users and customers).

3.2 Selection of a process

The choice of an appropriate method depends on the environment, goal and project type. Essential matters concerning the environment are the developers' experience, communication with the customer, and project size vs. occurring changes. Other essential matters are issues to plan ahead, usage of old solutions in the project, the level of detail in the process and the evaluation, and steering of the work. [OT07, OOSE04, Coc00]

Elvesæter proposes the chosen process to be dependent on the type of system [Elv06]:

- brand new system (very rare)
- re-engineering (old system exist)
- modification (fixing a major problem)
- adding a new module (functionality)

Henderson-Sellers introduces the following process selection criteria [Hen04a]:

- alignment with the organization's strategy
- project size and timeframe
- development type (web, business, distributed)
- safety requirements
- other: scalability, complexity, architecture, reliability, maintainability, support, cost, development life span

Waterfall models are easy to use when the anticipated development work does not include many changes or uncertainties. In a waterfall model, the process proceeds step by step until the work is finished. The waterfall process is well known and easy to understand, but it does not support planned iterations, the steps are strictly defined and the development of first prototype is slow. [OT07, OOSE04] If requirements and the current situation are well known, the design and analysis will be sufficient and the waterfall model is an arguable choice [Kil07]. The waterfall or iterative models also fit well with critical systems [Coc00, Lev05].

If a moderate number of changes are expected in the development work, the *iterative process* methods are a good choice. RUP, OpenUP and OpenMethod, introduced in the

chapter four, are iterative. Iterative methods can be customer or risk-based, and the duration of an iteration depend on the case. Shorter iterations result in faster feedback and risk management, but also in higher expenses. [OT07]

Agile methods are good when the number of changes and risk involved are big. They use feedback instead of planning as their primary control mechanism. Agile methods are more efficient than a traditional approach [Lar03], but do not provide the planning capability needed from the business perspective. XP and Scrum are agile and they are introduced in chapter four. It could be phrased that they “will provide the most bang for the buck, but will not say when the bang will be”. [OT07]

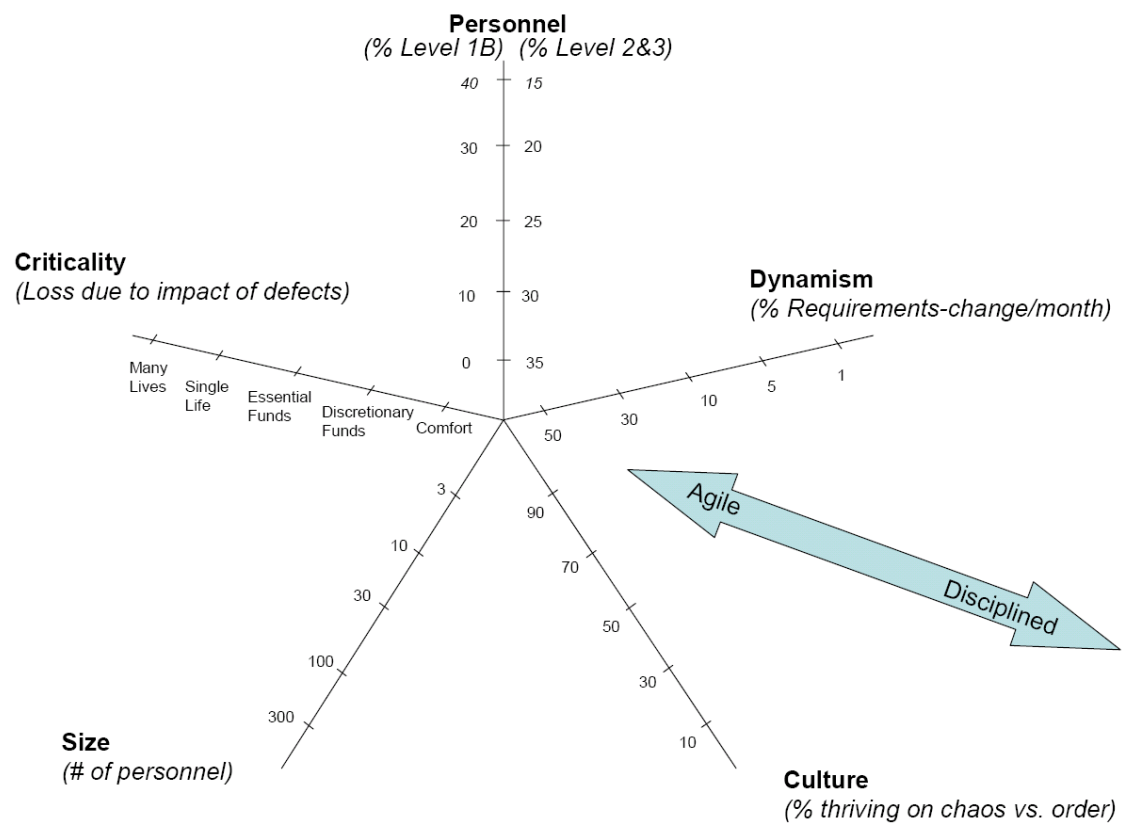


Figure 3.2.1 – Dimensions affecting the method selection [Boe03]

As a conclusion, the critical dimensions affecting method selection in terms of agility can be summarized as personnel (competency), dynamism, culture, size and criticality (see Figure 3.2.1) [Boe03]. The personnel level 1B is able to perform procedural method steps (e.g. coding a simple method, running tests), whereas personnel on levels 2 and 3 are able to tailor a method to a new situation or even revise (break the rules) when necessary. It is not, however, uncommon to change the selected method during the project [Jär07].

3.3 Evaluation and improvement of a process

The *Software Process Improvement (SPI)* term is used with methods such as *Capability Maturity Model Integration (CMMI)* and *Software Process Improvement and Capability dEtermination (SPICE)*. Järvi structures the important aspects of SPI as following [Jär06]: business, project and process coherence, process frameworks, process definition, SPI cycle and organization's capability.

The main SPI task is to constantly keep the software processes matching the company's business objectives. Process selection was discussed in section 3.2, and the different process methodologies will be introduced in chapter four. For process definitions, the SPEM provides a common standard, which is gaining support in the software industry and academic world. With the common notation, the common process definition conventions should also be developed. The SPI cycle is very firm. The IDEALSM model (Figure 3.3.1) is one well-known model and was originally based on CMMI. The SPI cycle could become more efficient with proper software process modeling. Depending on the organization's capability, the SPI's role can be more detached to each organizational area, and it can better take into account the needs of different parties. [Jär06]

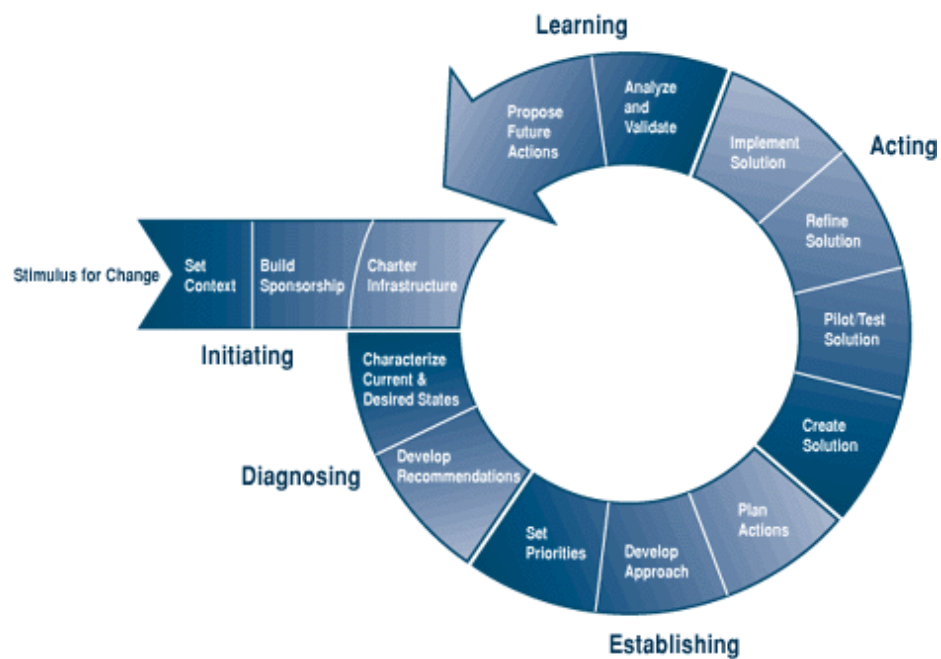


Figure 3.3.1 – The IDEAL model – a traditional SPI cycle with five phases [SEI07]

The IDEALSM model graphic, Copyright 1997 by Carnegie Mellon University is used in this publication with special permission from the Software Engineering Institute.

Börjesson [Bör04] measures the SPI success by implementation success – how initiatives lead to actual changes in engineering practice. SPI requires planning, dedicated people, management time and capital investment. The most efficient initiatives target practices in a single unit or a project which requires few compromises. Engineers and practitioners are highly committed and allocate time for improvement as they understand the need for the new approach, and appreciate the SPI initiative. SPI reduces development cost, improves productivity and customer satisfaction [Nia06].

A successful SPI approach emphasizes iterative development, which helps in correcting failures and modifying processes based on practical experience. The more iterations, the more probable is the success of implementation. In SPI, chaos is, however, expected, since changes cause debate, anxiety and resistance. [Bör04] The productivity vs. time curve is shown in Figure 3.3.2.

There are the following challenges and drawbacks in SPI [Sch05]: the increase of bureaucracy with the cost of reduced innovation, software development is basically dependent on problem solving (and not only on a good process), and that while SPI is technically oriented, it is also a human activity requiring also social skills.

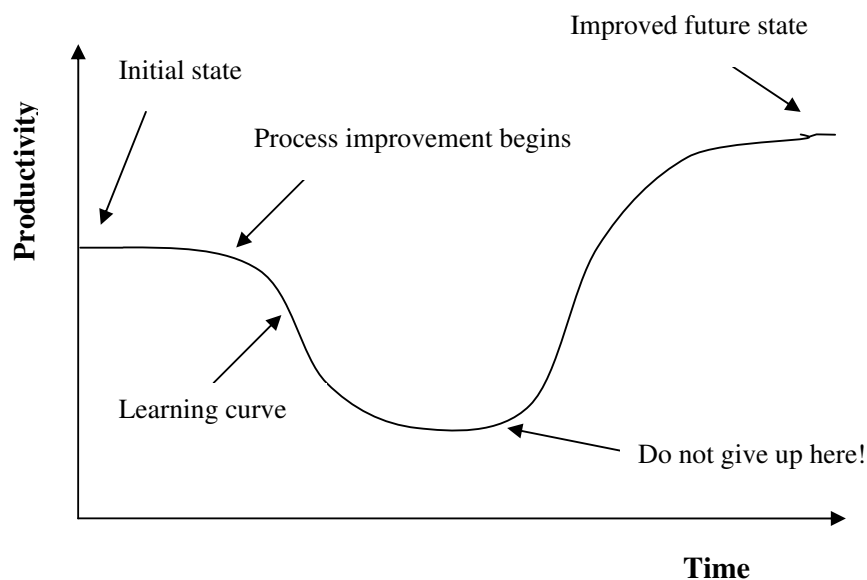


Figure 3.3.2 – Productivity vs. time in process improvement [Wie05]

3.3.1 Knowledge Management

Schönström [Sch05] argues that organizations should focus more on understanding and improving the knowledge processes of software development together with traditional SPI. The main knowledge processes are considered to be *knowledge creation* and *knowledge transfer*. These are here briefly introduced to allow a more extensive view of SPI.

Knowledge creation can be divided into *socialization*, *externalization*, *combination* and *internalization*. In socialization, new tacit knowledge is shared in everyday social interaction. Externalization is the process where tacit knowledge is made explicit, as concepts, documents etc, which can be stored in a repository. Explicit knowledge in a software project includes the process it uses, e.g. RUP, and tacit knowledge is knowledge of how processes work in a real project. During combination, explicit knowledge is collected from various sources, processed, and a more complex and systematic knowledge is generated. In internalization, the explicit knowledge is processed and converted into tacit knowledge by individuals. This is when knowledge is applied and used in practical situations.

Knowledge transfer requires a shared language and a common system of meaning. Otherwise, individuals of one group may view another group's knowledge as useless. Trust has proved to be an important factor for successful knowledge sharing as knowledge can be highly political and because knowledge is power. [Sch05]

3.3.2 CMMI

Capability Maturity Model Integration (CMMI) is a process improvement approach for organizations. The *Capability Maturity Model (CMM)* was originally developed as a response to the software crisis in the 1980s [Lju04]. As CMM models evolved into various areas, CMMI was developed to combine these models into a single integration [SSE07]. CMMI helps an organization to

- appraise its organizational maturity,
- establish improvement priorities and implementations of these,
- focus attention on key areas,
- select contractors. [Man07, Sch05]

CMMI is not a process description, but it rather describes the characteristics of a good process, and gives recommendations on assessment and improvement of current processes. It also provides guidance on how to increase maturity and performance. The changes in process usually aim at improving the performance of the processes. [Man07, CMMI07]

The CMMI is one of the most frequently used improvement frameworks. As thousands of organizations have used CMMI during their SPI project, it can be considered a standard SPI model [Sch05]. Process modeling is suitable for any capability level. The ‘staged representation’ part corresponds with the CMM model and the ‘continuous representation’ element corresponds with the SPICE model [Ter06].

The CMMI suggests five maturity levels [Man07, Sch05, SSE07]:

- *Initial*: The process is unpredictable, poorly controlled and reactive. Process success depends on the competence of a few people within the organization. Default starting level and most of the organizations are at this level.
- *Repetitive*: The process is characterized for projects. They are performed and managed according to their documented plans.
- *Defined*: The process is characterized for organization and proactive. Common practices and processes are defined in detail throughout the organization, and are tailored for the specific needs of a project.
- *Managed*: The process is measured and controlled. Quality and process performance are managed statistically and they are predictable.
- *Optimizing*: The focus is on the process improvement. Processes are continuously developed based on history knowledge. Reasons for performance changes are identified and measurable targets are set for improvement.

The challenges and drawbacks of CMMI are: it requires significant training to interpret the model, it assumes that the reader is a software/system expert, it is most suitable for bureaucratic organizations, it is large and heavy [SSE07], and it has a limited perspective - not taking organizational context into greater consideration [Sch05].

3.3.3 SPICE

Appraisal methods such as *SPICE (Software Process Improvement and Capability dEtermination, ISO/IEC 15504)*, give guidelines for determining the maturity of an organization's processes compared to a reference model. [SSE07]

The SPICE project was started in 1992 due to the need of organizations to reduce risks associated with software projects and improve the quality of the software. It had the following design goals:

- intended for process improvement and capability determination
- the standard could be used in a wide variety of environments
- use objective and, where possible, quantitative criteria
- produce profile outputs instead of number results with the support of comparisons with outputs of other similar assessments
- promote the technology transfer of software process assessment into the software industry [Ele98, Joh04]

The initial task of the SPICE project was to develop a suite of documents for an international standard for software process assessment [Joh04]. The process assessment can be used to process improvement and capability determination. The relationships in process assessment are depicted in Figure 3.3.3.1. Note that the tacit knowledge of the process is not observed in this model.

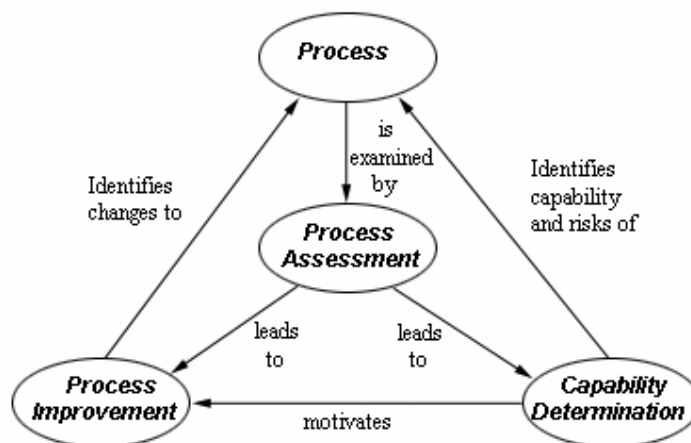


Figure 3.3.3.1 – Relationships in Process Assessment [SPICE]

The main activity is the process assessment, which is conducted with an Assessment Model, a Reference Model, an Indicator Set, an Assessment Method and one or more competent assessors. [Ele98, SPICE]

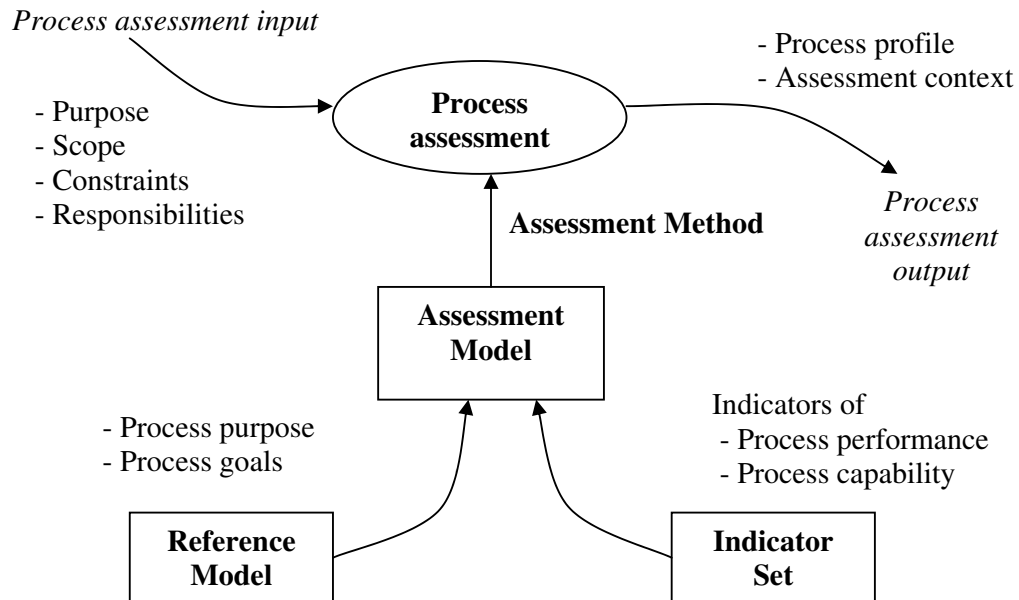


Figure 3.3.3.2 – The framework for process assessment [Ele98]

SPICE defines the Assessment Model and the Reference Model. The Reference Model defines the necessary rules that the Assessment Model must follow. The Reference Model is two-dimensional, consisting of the process dimension and the process capability dimension [Ele98, SPICE]. The capability levels correspond mostly with the CMMI levels: The main difference is that the CMMI level ‘initial’ consists of SPICE levels 0 and 1, where level 0 does not fulfill process requirements, and level 1 fulfills the basic process requirements [Ter06].

The actual software process assessment is conducted using an Assessment Model. The Assessment Method is not defined explicitly; however, the requirements for an Assessment Method are defined. This means that there can be several Assessment Methods available which meet these requirements. The informative criteria for a competent assessor are defined in SPICE. SPICE can be used to assess CMMI models and CMMI Product Suite and Reference Documents are compatible with SPICE [SPICE].

4. Software development methodologies

A *software development methodology* (also known as method, framework and model) is a structured collection of best practices, guidelines and tools for software development. It aims to achieve a wanted goal in a given environment. [OT07] Though the term methodology is often referred as method, the methodology is the body of methods and meant to support all software development phases [Elv06]. A timetable with a collection of routes and means of transportations could be considered to be a methodology for transporting if it also contains advice on how to use the methods (e.g. how to get to the train station) and what to do if all means of transportation are not available (e.g. service break on a specific train route, please use the specific bus connection instead). The term ‘framework’ is used when only a basis for doing is given instead of exact instructions. [Kni07]

First we will study in depth the following software development methodologies: Rational Unified Process, OpenUP, OpenMethod, Extreme Programming and Scrum. RUP is introduced most extensively to gain a more thorough view on the concept of disciplines and phases. Since the terms defining the methodology are used interchangeably and the use and meaning varies by the point of view, we will use the term introduced with each methodology’s reference. These methodologies were chosen on two different criteria; OpenMethod and RUP, which are similar to each other, with OpenUP as an open source example, and XP and Scrum as the most agile and utilized examples. Secondly, OpenMethod and RUP, with methodologies included with Eclipse Process Framework.

4.1 Rational Unified Process

The *Rational Unified Process (RUP)* is an iterative software development process framework. It has been developed by *Rational Software*, which has been owned by *IBM* since year 2002 [Amb06]. RUP is marketed to provide “best practices and guidance for successful software development.” [RUP701]

RUP is one of the most popular and complete process models used by developers in recent years. It is also the best-known and most extensively documented commercial variant of

Unified Process [Jaf05, Wes05]. *OpenUP* is an open source framework of Unified Process and explained in section 4.2.

The popularity of RUP depends on various factors. In the late 90s companies were ready for process standardization. Many companies had own processes but began to search for commercial ones. The RUP process development team was launched in 1996 and it believed that the average user had to find it easier to do their jobs with the RUP than without it. To ensure user value, the RUP was tightly integrated with user tools. The RUP's technical keys for success were the following: it was Web-enabled, easy-to-use and non-intrusive. The favorable business climate during the evolution of RUP was also the key element for RUP to gain popularity. [Kro03]

Three elements that define RUP are

- An underlying set of philosophies and principles for successful software development.
- A framework of reusable method content and process building blocks.
- The underlying method and process definition language.

Software development risks are as for any other development: lack of resources, insufficient funding, tight deadlines, slow organizations. The risks implicitly for software development are new and unknown technologies, undiscovered requirements and complicated architecture.

The RUP has the following strategies, also known as *RUP best practices*, to handle the risks:

- Develop iteratively.
- Manage requirements.
- Use component architecture.
- Model visually.
- Verify quality continuously.
- Manage changes.

RUP is included with Rational Method Composer, which allows customization of the process. It contains RUP for several different content areas, domains and technologies. Rational Method Composer is explained more detailed in section 5.3. Since RUP version 7.0 the terminology and concepts are as proposed for the SPEM 2.0.

RUP divides the software development lifecycle into two dimensions – *phases* and *disciplines*. The phases of RUP are *inception*, *elaboration*, *construction* and *transition*. As shown in Figure 4.1.1, phases divide the process over time and disciplines logically group activities by nature. Disciplines can last over all the phases and have different time consumptions; for example in early iterations time is spent on requirements and later on more on implementation. [RUP701]

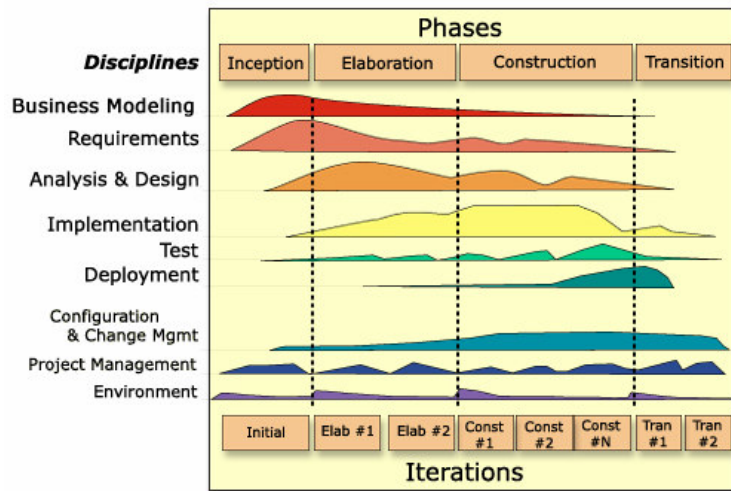


Figure 4.1.1 – The Rational Unified Process – Phases and Disciplines [San07]

The nine disciplines make the vertical dimension. Each discipline has its own activities, tasks, work products and guidance [Gom05]. Let us now briefly explain the RUP dimensions starting with the disciplines:

Business Modeling

This discipline is used to understand the structure of the target organization, both statically and dynamically. This helps understanding the scope of the project.

Requirements

The major objectives are to understand the customer's needs, scope the system to be built and provide detailed requirements and use-case model of the system.

Analysis & Design

This discipline explains how to convert requirements into models and work products which are suitable for implementation.

Implementation

This discipline contains development of source code, unit testing and integration. These are performed iteratively.

Test

The main activities are evaluating and assessing the product quality. Testing is done throughout the process lifecycle.

Deployment

The objective of this discipline is to deliver the system to the customer. This includes training, packaging, distributing, testing and installing the system.

Configuration and Change Management

This discipline explains how to manage, control and synchronize the work products. As RUP is iterative, a lot of builds are created which results in many versions of artifacts.

Project Management

The purpose of this discipline is to manage project by planning, risk management, monitoring progress and metrics.

Environment

The Environment discipline supports the development organization by providing tools and processes which suit the need of the current project.

Now we introduce the *RUP phases*. RUP has four phases: *Inception*, *Elaboration*, *Construction* and *Transition*. Those are decomposed over time. Every phase is

characterized by one or more iterations and each is concluded by a major milestone. If phase objectives are met, the project will move to the next phase.

Inception

The goal of the inception phase is to develop a common understanding among all stakeholders on the objectives for the project.

Elaboration

The second phase of RUP is the elaboration, which purpose is to achieve understanding what the system should do and provide the baseline of the system architecture.

Construction

In the construction phase the system is built, and the architecture is validated. This is the most iterative and most time consuming phase of RUP.

Transition

Transition is the last phase and includes the final beta testing, and minor fixes based on the user feedback. The product is now delivered to the customer.

RUP's strengths include well-defined work products, it is easily combined with techniques from other methods, and it is easily customized. RUP is also widely adopted; hence, it has good learning and consulting resources [Lar03]. By 2003, RUP has been used by over half a million users in more than three thousand companies [Cas07].

RUP seems to be a very big and complex methodology. It has what is needed for software development, but the enormous amount of information can easily make it hard to use. In the following section we will study the open source version of Unified Process, OpenUP.

4.2 OpenUP

OpenUP is an open source framework of Unified Process and part of Eclipse Process Framework. OpenUP/Basic, which is the most lightweight version of OpenUP, is referred

in this thesis as OpenUP. Have a look at section 5.2 for more information about EPF. This section is based mostly on references [OpenUP] and [Bad05].

OpenUP was originally known as *Basic Unified Process (BUP)* developed by IBM. It was donated to Eclipse Foundation in 2005 and renamed to OpenUP in 2006. OpenUP is a collection of best practices from different sources – not only from IBM [Dej06].

OpenUP is *minimal, complete* and *extensible*. E.g. it has only the required content, it can be used as an entire process to build the whole system, and it can be used as a base on which more content can be added as needed.

OpenUP core principles are the following:

- Collaborate to align interests and share understanding.
- Balance competing priorities to maximize stakeholder value.
- Focus on the architecture early to minimize risks and organize development.
- Evolve to continuously obtain feedback and improve.

Like RUP, OpenUP is organized in two dimensions: the *phases* are inception, elaboration, construction and transition, and the six *disciplines* are requirements, architecture, development, test, project management and change management. The roles are divided into the following seven roles: analyst, architect, developer, tester, project manager, any role and stakeholder.

OpenUP strengths versus others include open source development and community with constant improvement and up-to-date best practices. It is easily adopted and understood, and it is well defined [IT07].

OpenUP exists so far only as an EPF version (and EPF Wiki [EPFW07]) and it is the mostly developed compared to the EPF versions of Scrum and XP. It has content translated at least partly for Portuguese, Russian and Spanish. The latest version 1.0 was published on August 1st, 2007 with the release of EPF Composer 1.2. The method content and process content are completely done and comprise of instances of all major SPEM 2.0 concepts.

As a conclusion, OpenUP is a much simpler process than RUP, but it preserves the essential characteristics of RUP like iterative development, use cases and scenarios driving development, risk management and architecture-centric approach. OpenUP is designed for small teams working together in the same location. It supports flexibility and agility, since it has a small number of roles, tasks and artifacts and its process allows selection of desired method elements.

4.3 OpenMethod™

OpenMethod™ by *SYSOPENDIGIA*, later OpenMethod, is an Application Development Method which is intended for IT developer organizations and for organizations buying IT projects. OpenMethod version 9.0 is used for closer observation in this thesis. OpenMethod provides guidelines for the business-oriented development of ICT systems, architectures and processes. OpenMethod employs the best market practices and contains concrete process descriptions, practical instructions, check lists, templates and examples. [SD07]

OpenMethod is used for enhancement of systems, architectures and business-processes. It unifies the operation methods in different phases of system development. For quality assurance OpenMethod has instructed survey and testing practices. At the moment OpenMethod does not contain project management. This is covered in a product called *OpenProject*. A clear overlap can be noticed with the RUP and OpenMethod practices.

OpenMethod best practices are:

- Business-driven development
- Manage and accept requirements
- Model visually
- Reuse components
- Develop iteratively
- Control project quality, risks and changes
- Take care of software security

OpenMethod is developed according to customer feedback and project experience. It has hundreds of users in Finland, including companies outside SYSOPENDIGIA. A new version of OpenMethod is published twice a year.

OpenMethod strengths include focus, Finnish language, specific point of view (e.g. Rational tool support for ReqPro, Rose, SoDA, RSM, examples, templates), and a possibility to affect the development locally. OpenMethod can also be categorized as a methodology instead of a framework as it also gives the actual recipes for how to do the work. The modeling of OpenMethod is depicted in chapter eight.

4.4 Extreme Programming

XP (eXtreme Programming) was developed by Kent Beck in 1996. The advent of XP sparked the agile movement and currently it is the most famous of the agile methods. [Ram06, Sal03, Mar03]

XP is a set of simple and concrete principles and practices that has been found effective in software development processes. It evolved from the problems caused by long development cycles with traditional methods. Extreme Programming's success is based on the focus of responding to customer's changing needs rapidly and many project teams will be able to adopt it as-is [Mar03, Sal03]. XP, however, lacks of a comprehensive project management view [Abr03].

XP's key principles and values are

- **interaction:** developers and the customer must be closely connected during the whole project life cycle and communication should be fluent,
- **keep it simple:** developers should concentrate on implementing only the necessary functionalities and keeping the code as simple as possible,
- **feedback:** developers get rapid feedback from unit tests and customer, which helps learning and developing the system,
- **courage:** when other values are met, the courage is needed to be able to make big changes and abandon defective code when necessary. [Sal03]

The life cycle of XP consists of six phases: *Exploration*, *Planning*, *Iterations to Release*, *Productionizing*, *Maintenance* and *Death* (Figure 4.4.1). [Ram06, Abr02]

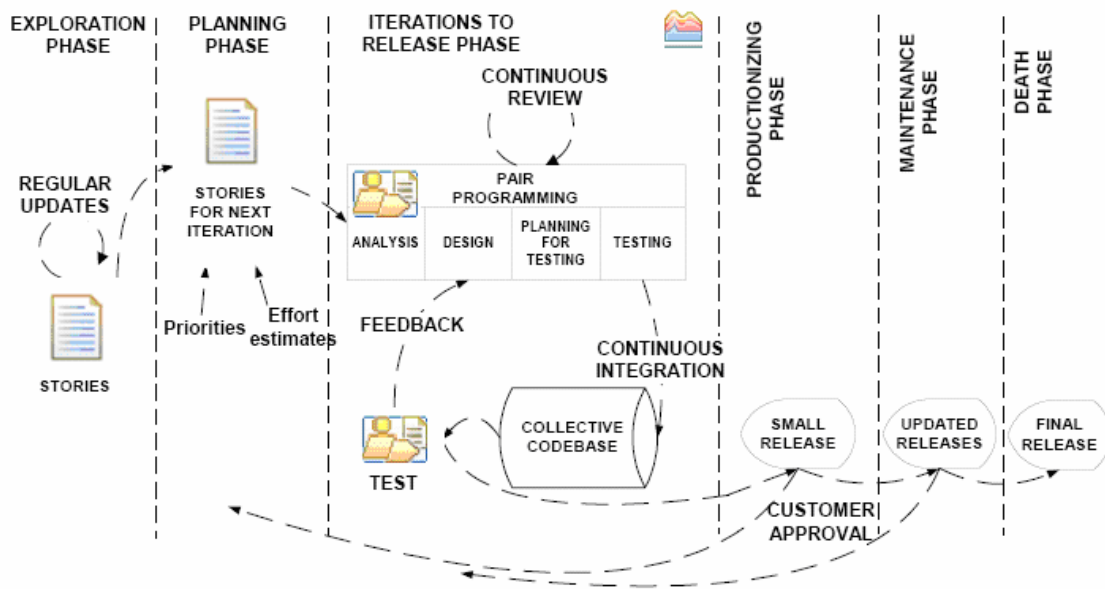


Figure 4.4.1 – A general overview of a typical XP process [Ram06]

1. **Exploration:** The customer writes the initial list of high-level requirements, while the team familiarizes itself with tools and practices and determines the overall system design through prototyping.
2. **Planning:** Focus is on prioritizing the requirements, estimating the effort and time needed for each requirement and determining the schedule for the first release.
3. **Iterations to Release:** This phase includes several iterations before the first release. The customer decides the requirements selected for each iteration.
4. **Productionizing:** Focus is on extra testing, verification and validation of the first release and deployment into user production environment.
5. **Maintenance:** Implementation of remaining requirements into the running system. The project is not over yet, but rather in a phase of system evolution.
6. **Death:** The customer does not have any further requirements to be implemented. The documentation is written, review is conducted and the focus is on project closing. [Mar03, Abr02]

XP's strengths include practical, high-impact development techniques, emphasized communication between all stakeholders and daily measurement. [Lar03]

The EPF version of XP is still under construction. The latest version is 0.1, which contains the key concepts, best practices, tasks, work products and roles. The method content is modeled rather straightforward and there is not yet any actual process content.

4.5 Scrum

The term *Scrum* originally comes from a strategy in rugby. It means “a struggle for the ball by the rival forwards hunched tightly round it”. In British, Scrum also has a meaning of “a disordered or confused situation involving a number of people”. [Dict07]

In software development Scrum was introduced in 1986. It was used to refer an adaptive, quick and self-organizing product development process originating from Japan [Abr02, Ram06]. Scrum is a management and control process for building software that meets business needs [Kos03]. It does not define any specific techniques for the implementation phase, but rather concentrates on how the team members should work flexibly in a constantly changing environment.

Scrum's key practices and principles are [Lar03]:

- Self-directed and self-organizing team
- Once work to an iteration is chosen, no external work is added
- Daily short meetings with specific questions
- Usually 30-day iterations followed by a demo to the stakeholders
- Client-driven adaptive planning in each iteration

Scrum consists of three phases: *pre-game*, *development* and *post-game* (Figure 4.5.1).

1. **Pre-game:** focus is on setting the basis for the iterative-incremental development.

This phase has two subphases:

- a. **Planning:** concentrates on producing the list of prioritized requirements, analyzing the risks, estimating the resources and determining an overall schedule.
- b. **Architecture / High-level Design:** determine the overall architecture of the system

2. **Development:** Focus is on iterative and incremental development. Each iteration (*Sprint*) is typically one month in duration. The daily questions include: "What have you done since yesterday, what are you planning to do by tomorrow, and do you have any problems preventing you from accomplishing your goal?" Larman [Lar03] also included questions "Any tasks to add to the Sprint Backlog?" and "Have you learned or decided anything new, of relevance to some of the team members?"
3. **Post-game:** Focus is on integrating and releasing system into user environment.

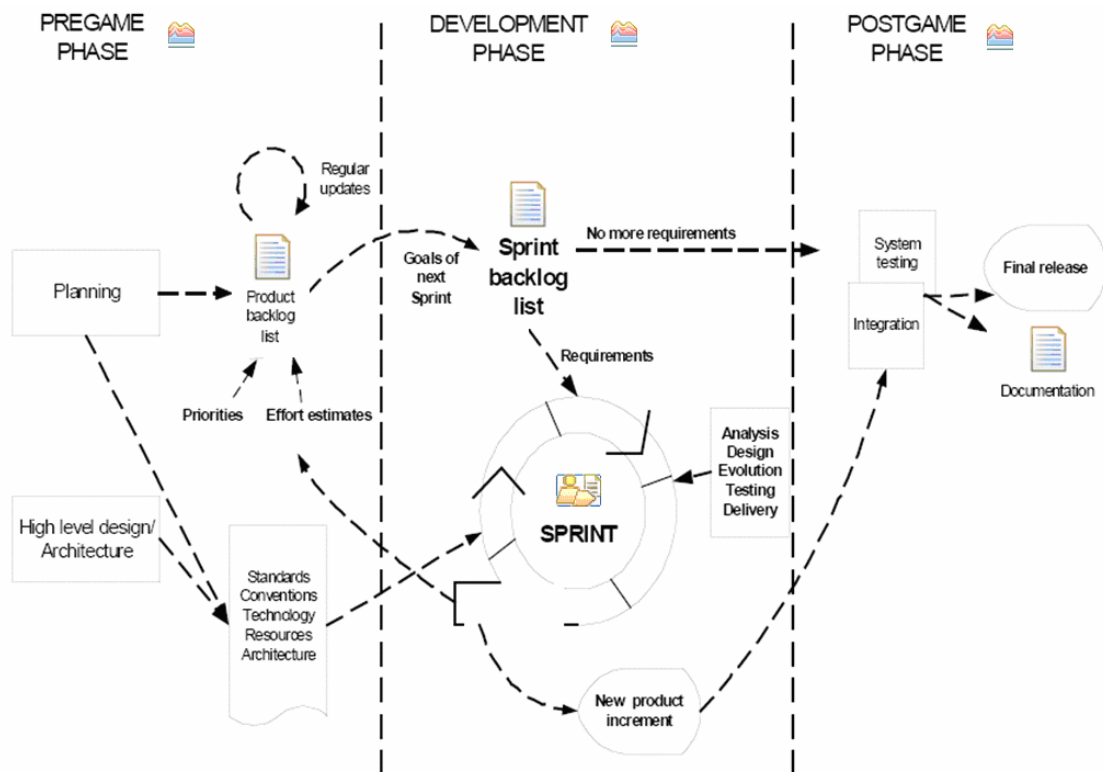


Figure 4.5.1 – The Scrum process [Ram06]

Scrum's strengths include simple practices, individual and team-based problem solving, team communication, learning, and value-building with openness and visibility [Lar03]. Scrum has been used in hundreds of organizations [Sch07]. Kniberg [Kni07] presents a good practical view on Scrum and how to combine it with XP.

The EPF version of Scrum was originally released in French but has now also an English version. It contains a short introduction, roles, work products, and Scrum lifecycle.

4.6 Comparison of RUP, OpenUP, OpenMethod™, XP and Scrum

The comparison of methodologies with others is difficult and the result is easily based on the subjective experiences of the practitioner and the intuitions of the authors [Ram06].

According to [Ram06] the comparisons can be approached in these different ways:

1. Introduce an idealized methodology and evaluate others against it.
2. Distill a set of important features from several methodologies and compare each method against it.
3. Define a meta-language as a communication tool and a frame of reference against which you describe the methodologies.
4. Try to relate the features of each method to specific problems.

We will use the second approach in our study. The intention is not to value one methodology over another, but rather identify the differences and similarities between these different software development methodologies. See section 2.4 for general comparison of agile and traditional processes. The results of the comparison are shown in Figures 4.6.1 and 4.6.2.

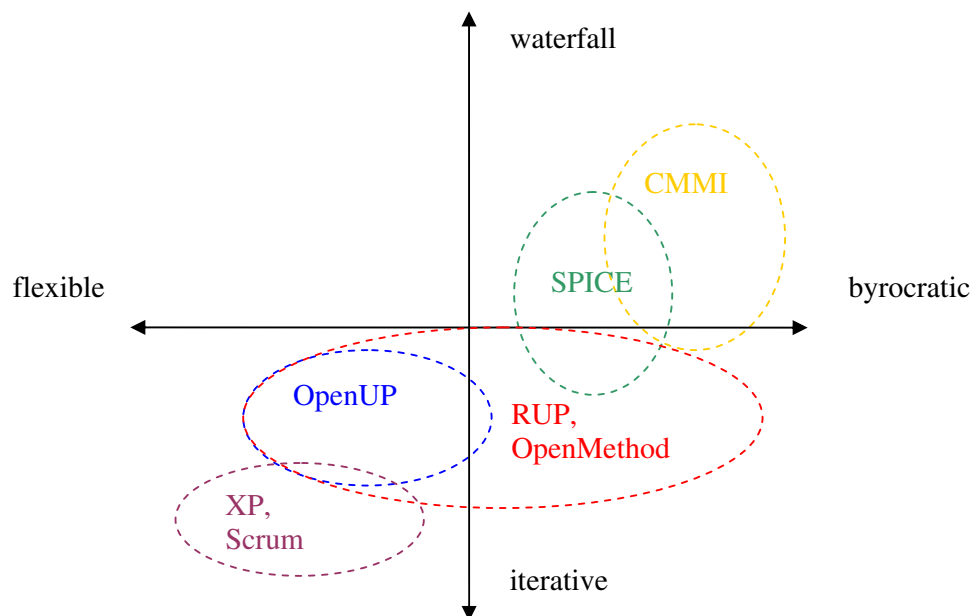


Figure 4.6.1 – A comparison of methodologies in flexibility and iterativity [Hai07]

	RUP	OpenUP	OpenMethod	XP	Scrum
Key points	Complete SW development model with tool support	Agile and open source version of the Unified Process	Customer driven development model with hands-on tools	Customer driven development, small teams, daily builds	Small, for self-organizing teams, with 30-day release cycles
Categorization	Framework	Framework	Methodology	Methodology (discipline)	Framework
Usage	Medium to large projects	Medium projects	Medium to large projects	Medium projects	Medium projects
Special features	Business modeling, tool family support	Minimal, complete and extensible	Well-tried hands-on tools and tool support available	Ongoing refactoring, for uncertain requirements	Product development view of Scrum, focus on project management
Commercial (License)	Yes	No	Yes	No	No
Team size	Medium to large	Small, co-located teams	Not specific (Small to large)	Small and medium size teams, co-located	Small teams (or medium divided), co-located
Language	English	English, partly in Portuguese, Russian and Spanish	Finnish	English + other translations	English + other translations

	RUP	OpenUP	OpenMethod	XP	Scrum
Strengths	Well defined work products, easily combined with techniques from other methods, good learning and consulting resources	Open source development with constant improvement and up-to-date best practices, easily adopted and understood	Focus, Finnish language, hands-on tools, specific point of view with tool support available, continuous user driven development	Practical, high-impact development techniques, emphasized communication between all stakeholders and daily measurement	Simple practices, individual and team problem solving, team communication, learning, and value-building with openness and visibility
Shortcomings	No tailoring descriptions (for changing needs), requires a lot of training, complex, too strongly oriented towards Rational tools	Few practical real-life experiences so far, no tool support	Currently project management in OpenProject, Finnish language, complete life cycle not equally covered	Process is vague, rather a set of principles and practices than a methodology, lack of formalism and management practices, scaling problems with bigger projects	Lack of detail in integration and acceptance tests, lack of scalability, based on assumption that human communication is enough
Development, releases	Continuous	Continuous, open source	Continuous, twice a year	Stabilizing, studies	Stabilizing, studies

Table 4.6.1 – A comparison of methodologies [Abr02, SD07, Ram06, Lar03, Ter06]

5. Models and tools

This chapter focuses on the models and tools aiding software process modeling. The meta-model chosen is SPEM as it is gaining acceptance as a process meta-model standard. Tools can be generally divided into *CASE* (*Computer Aided Software Engineering*) and *CAME* (*Computer Aided Method Engineering*) tools.

CASE tools are software which help or promote a phase or phases of software development [Hai00]. They are used for standardization and normalization by developers and are instantiated by a CAME tool [Kos99, Mol06]. A CASE tool could, for example, be a UML tool or a testing tool. The benefits of CASE tools are eminent: they increase productivity, ease method use, and generate at least partly automated documentation [Hai00]. The CASE tools have the following drawbacks: they are often hard coded for certain methods and may change current working practices and thus cause resistance. They have limited possibilities to assess and evaluate different methods, and have a limited ability to support customized method in a specific situation [Kos99]. CASE tools are also often expensive and require extensive training [Hai00].

CAME tools are software which aid in method engineering. They are used by method engineers to compose new methods, assist selection of method fragments from repository and they instantiate a method specific CASE tool. CAME tools are based on meta-models (e.g. SPEM 2.0). Method fragments can be created in several ways: models of the fragments are defined by instantiating the method meta-model, fragments are assembled in order to satisfy some specific situation, or fragments are obtained by modification of other fragments to better satisfy the situation (see section 3.1). [Mol06]

In sections 5.2 and 5.3 we study CAME tools Eclipse Process Framework Composer and Rational Method Composer.

5.1 SPEM 2.0

The SPEM 2.0 is a meta-model developed by *Object Management Group (OMG)*. In this Thesis, SPEM is referred to *Software & Systems Process Engineering Metamodel Specification v. 2.0*, Proposed Available Specification ptc / 2007-08-07. This section is mainly based on [SPEM2.0] –reference.

Since the 80s, there have been various process meta-models and finally *Object Management Group* decided to standardize a process meta-model. The specification of SPEM 1.0 was released in 2002. In 2005, SPEM 1.1 [SPEM1.1] was released with minor updates. SPEM 1.1 still had several shortcomings; for example, semantics was ambiguous and hard to understand and there was a lack of enactment support. SPEM 2.0 was released in 2007 fixing the flaws in previous versions, being compliant with UML 2 and providing guidance on migrating existing process models from SPEM 1.1 to SPEM 2.0.

SPEM is a model of a model, a meta-model, which is used to describe a concrete software development process or a family of related system and software development processes.

SPEM describes structures needed to formally express and maintain method content and processes, i.e. it defines a language and representation schema for method contents and processes. SPEM is not, however, intended to be a generic modeling language; rather, it defines the ability to choose the behavior modeling approach that fits the implementer's needs. SPEM uses the *UML 2.0* infrastructure and diagram interchange standards.

A meta-modeling language (meta-meta-model) is used to describe the SPEM meta-model itself. The *MOF 2.0 standard* [MOF] provides such a language. The layers can be depicted as levels M0, M1, M2 and M3. The layers can theoretically continue to M4, M5, etc. but these are not used in practice [Jeu07]. M3 provides MOF which is instantiated by SPEM 2.0 on the M2 layer. UML 2 meta-model instantiates MOF in the same way. On layer M1 there are concrete instances, such as 'Use Case', 'Analyst'. Layer M0 consists of the performing process.

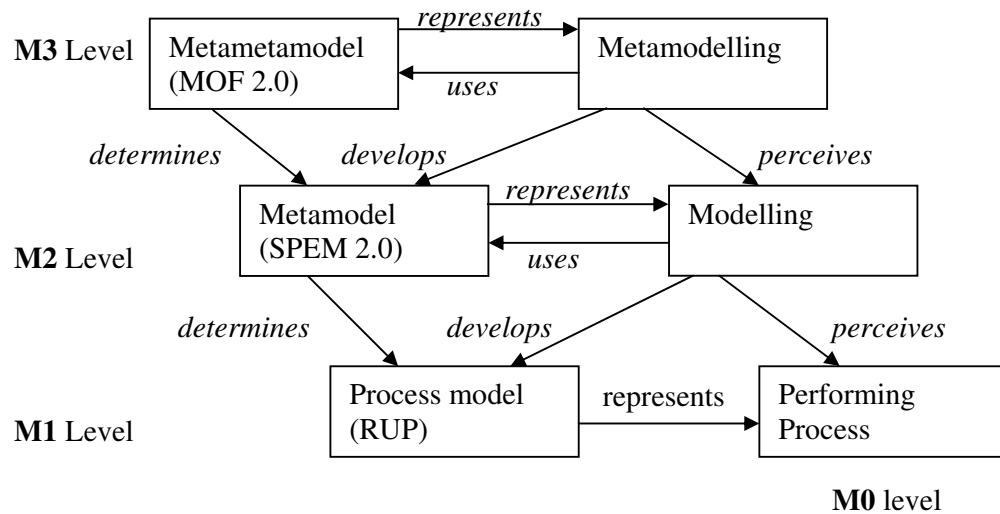


Figure 5.1.1 – Process modeling layers [Jär05, Kos99, SPEM 2.0]

SPEM 2.0 has the goal of being a standard for software process [Hau06]:

- *Definition:* Providing unambiguous semantics
- *Presentation:* Providing clear notational guidance
- *Representation:* Providing a concrete modular, configurable, and reusable meta-schema
- *Interchange:* Define standard format for process content interchange
- *Planning:* SPEM processes must fit to be used as planning template
- *Enactment:* Fit for direct enactment and execution metrics.

SPEM tries to accommodate a large range of development methods and processes of different styles, cultural backgrounds and communities. SPEM consists of seven main meta-model packages (Figure 5.1.2): Core, Process Structure (defines flexible ad-hoc process models which are ideal for the representation of agile processes), Process Behavior (links process models to externally defined models such as UML 2.0), Managed Content (adds textual description and documentation capabilities), Method Content (defines reusable method content elements which provide base of documented knowledge), Process With Methods (refines process models to base and trace process on Method Content) and Method Plug-in (supports scaling to method libraries with plug-ins that extend each other with variability mechanisms).

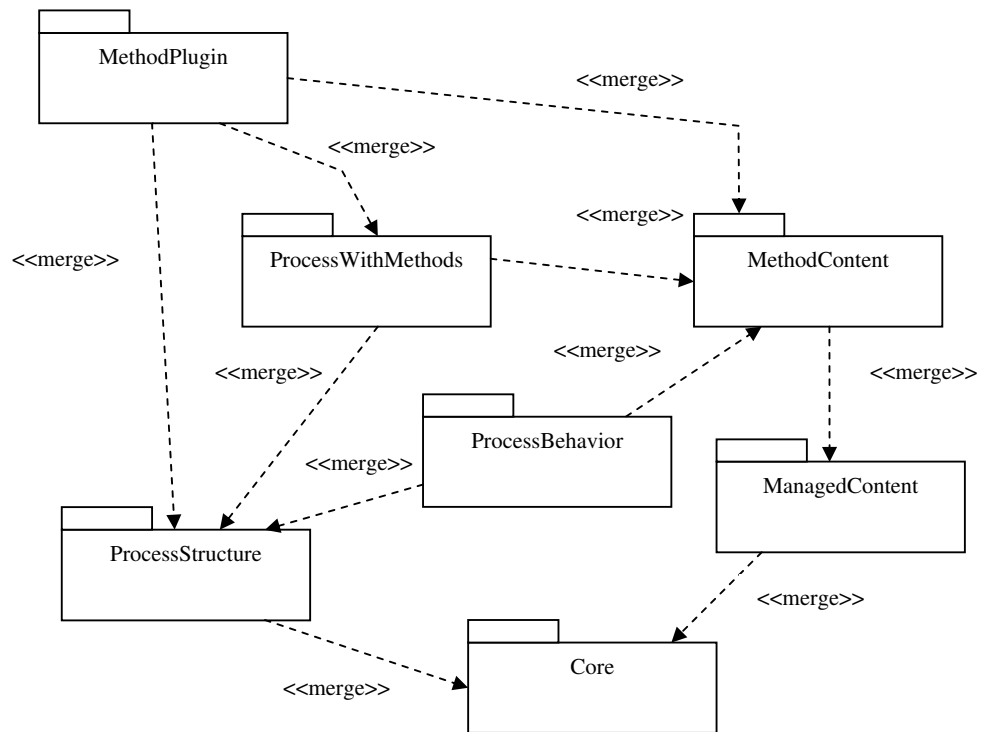


Figure 5.1.2 – Structure of the SPEM 2.0 Meta-Model [SPEM2.0]

SPEM regards the main elements of a software development process as consisting of roles, the work products they are responsible for, and the tasks that they perform on the work-products. The relationships between these are depicted in Figure 2.1.2. The elements are often accompanied with Guidance. The behavior of these elements is not, however, as bound as in SPEM 1.1.

Method framework is divided into method content and process content. Method content describes the reusable building blocks and general development techniques and practices in lifecycle-independent form. The process content then applies method content for assembly of different executable processes. These are described with breakdown structures.

Table 5.1.3 depicts some of the most essential SPEM 2.0 elements.

5. Models and tools













Name	Description	Icon
Activity	An Activity represents something that one or more roles perform. It is a grouping of nested process elements (e.g. Breakdown Element) such as other Activity instances, Task Uses, Role Uses, Milestones, etc. Activities are related to timelines.	
Breakdown Element	A Breakdown Element is any element that is part of the process structure.	-
Delivery Process	A Delivery Process is a special Process describing a complete and integrated approach for performing a specific project type. It describes a complete project lifecycle end-to-end that has been detailed by sequencing Method Content in breakdown structures and can be used as a template for projects.	
Discipline	A discipline is a collection of related tasks that define a major 'area of concern'. It works as an aid to understand the project from traditional waterfall perspective as it is common to perform tasks concurrently across several disciplines. (Icon used with EPF)	
Guidance	Guidance is a Describable Element that provides explanations and additional information related to other Describable Elements. Specific types of guidance having a specific structure and type of content are classified with Kinds, for example Guidelines, Templates, Checklists, etc.	
Milestone	The development processes consist of sequences and milestones. A milestone describes a significant event in the development – it is the point where an iteration or phase formally ends. This provides a check-point for whether the process is ready to move forward.	
Phase	A phase is a significant period within a development process. During phase, a well-defined set of objectives is met and the phase ends with a major management checkpoint, milestone.	
Process	Processes use content elements to relate them to partially ordered sequences that are customized to specific projects. A process focuses on the lifecycle and the sequencing of work in breakdown structures.	
Process Pattern (Capability pattern)	A process pattern is a reusable building block for creating new development processes - Delivery Processes or larger Process Patterns. It describes a cluster of Activities that provides a consistent approach to common problems. This cluster has process knowledge for example of a discipline.	
Role Definition	A Role Definition is a Method Content Element that defines a set of related skills, competencies, behavior and responsibilities of an individual or a set of individuals. Roles are neither individuals nor necessarily equivalent to job titles. Roles are used by Task Definitions to define who performs them as well as define a set of Work Product Definitions they are responsible for.	
Task Definition	A Task Definition is a Method Content Element and a Work Definition that defines an assignable unit of work being performed by Roles Definition instances. The granularity of a Task Definition is generally a few hours to a few days. A Task is associated to input (mandatory and optional) and output Work Products and it usually affects one or only a small number of Work Products. A Task provides complete step-by-step explanations of doing all the work required to achieve this goal.	
Tool Definition	A Tool Definition is a special Method Content Element that can be used to specify a tool's participation and capabilities in a Task Definition. It is also used as a container for tool mentors.	
Work Product Definition	Work Product Definition is a Method Content Element that is used, modified, or produced by Task Definitions. Roles use Work Products to perform Tasks and produce Work Products in the course of performing Tasks.	

Table 5.1.3 – Some essential SPEM 2.0 icons with short explanations

5.2 Eclipse Process Framework

The *Eclipse Process Framework (EPF)* project is an *Eclipse Technology* open source project. Eclipse is an open source community building an open development platform for developing tools [Ecl07]. EPF aims to provide an extensible framework and exemplary tools based on SPEM concepts for defining and managing the software development process. This section is mainly based on [EPF07].

EPF aims at producing a customizable software process engineering framework, which supports different process methods, project types and development styles. It also contains exemplary process content and tools. The EPF project has two goals:

- “To provide an extensible framework and exemplary tools for software process engineering. This contains method and process authoring, library management, configuring and publishing a process.”
- “To provide exemplary and extensible process content for a range of software development and management processes supporting iterative, agile, and incremental development, and applicable to a broad set of development platforms and applications.”

Eclipse Process Framework Composer (EPF Composer) is a tool platform for those who are responsible for implementing and maintaining processes for development organizations or individual projects. The EPF Composer version depicted in this work was first 1.02, which was released on March 5, 2007, and later version 1.2, which was released on August 1, 2007. The term EPF is also used when referring to the EPF Composer in this thesis.

The benefits of EPF in enterprises are various [Kro07a]. EPF helps with leveraging existing best practices rather than reinventing the wheel. It provides a level of consistency and shared language across the organization for process modeling and related topics. As practitioners learn how to improve their approach to software development, new practices can be easily captured and effectively deployed. As the process content evolves and is rolled out to the organization, it enables continuous process improvements.

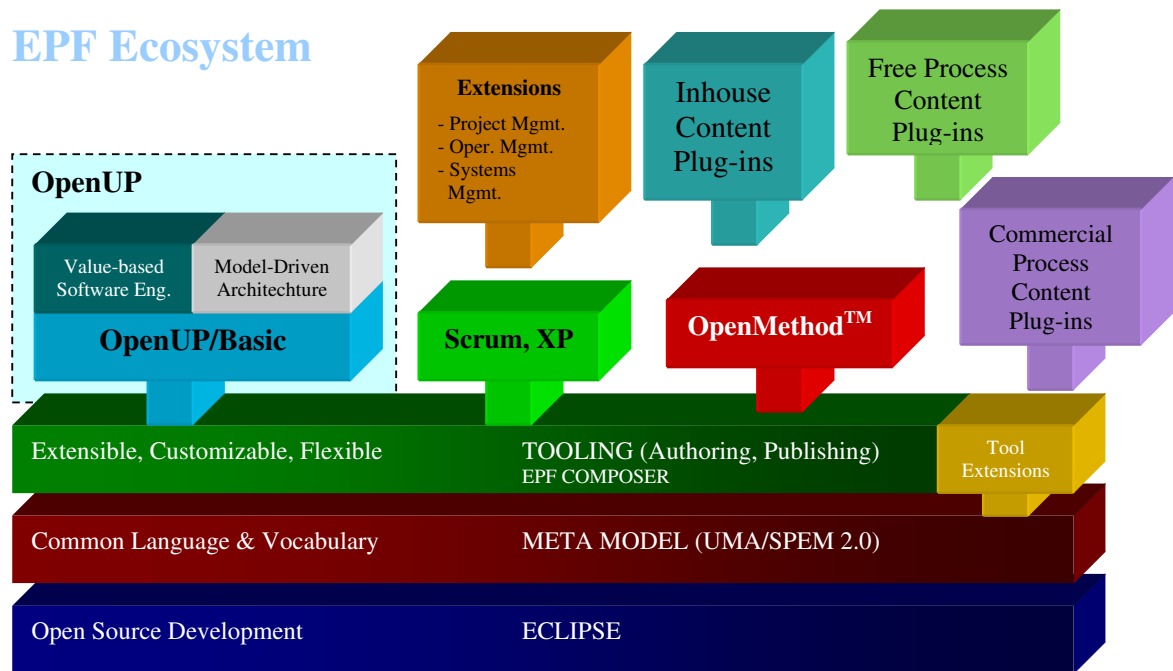


Figure 5.2.1 – the EPF Ecosystem with OpenMethod [EPF07]

From a first time user's viewpoint, EPF Composer is quite easy to learn. It has overviews, tutorials and first steps, and by following these, users can quickly become familiar with the tool. It is recommended that the user explores the tool's online help, which contains several interactive tutorials that provide step by step instruction for various scenarios.

The main authoring perspective is divided into library, configuration and main (editing) frames. The publishing feature produces content as a web site. This gives centralized access to information about the practices and processes. The content can be navigated through using different perspectives, such as by work product, by role, or by process. EPF supports the export of content to XML, method plug-ins, and MS Project.

Eclipse Process Framework consists of EPF Composer and plug-ins of the following methods: *OpenUP*, *XP* and *Scrum* (see Figure 5.2.1). OpenUP is a lighter open source framework version of RUP. XP (Extreme Programming) and Scrum are agile software development methodologies. These are covered in more detail in chapter four.

5.3 Rational Method Composer

IBM Rational Method Composer (RMC) is a commercial product by IBM Rational Software built on top of Eclipse. It is designed for the same purpose as EPF - authoring of method content and for publishing configurations of method content as processes. EPF Composer is a part of RMC code and was donated to the Eclipse Foundation by IBM as open source. The idea is that EPF Composer will be a core component of RMC. RMC will add value through special features and support that might not be possible in an open source product. The main difference between EPF Composer and the RMC tool is the lack of integrations with other IBM Rational tools such as *Rational Portfolio Manager*, *Rational Software Architect* and *Rational Process Workbench*. The Rational Unified Process (RUP) is included in the RMC software. [Hau05]

RMC includes several delivery processes of RUP [RMC06]: RUP for Small Projects, RUP for Large Projects (Classic RUP), different content areas such as RUP for Business Modeling and RUP for SOA Governance. Domains and technologies such as RUP for J2EE, RUP for .NET and RUP for User Experience are also covered.

The RMC capabilities and modeling with RMC can be compared with the EPF Composer capabilities and modeling presented in the following chapters.

6. Modeling capabilities and usage of EPF and SPEM

In this chapter we study the background and basis for the actual modeling. First we focus on why EPF and SPEM should be used, how the modeling has been done before EPF and SPEM, and what alternatives there are. Then we take an overview on how EPF and SPEM have been used so far and end with a discussion on the capabilities of EPF and sufficiency of SPEM.

The whole modeling process can be divided into a) *notations and language*, b) *the actual modeling activities*, and c) *context*, e.g. what to model. In the end of this chapter we focus on the notations and language. Chapter seven concentrates on the modeling process and in chapter eight we study the modeling of OpenMethod. The chapters six and seven connect the tools and methodologies presented in earlier chapters and the case study presented in chapter eight.

6.1 Why model existing models with EPF and SPEM?

Eclipse Process Framework provides solutions for common problems relating to method and process managing. In section 2.5 we studied the reasons behind modeling a software process. These general benefits and drawbacks also apply when modeling with EPF and SPEM.

According to Peter Haumer, a solution architect at IBM, these are some of the common problems with suggestions [Hau07a, SPEM 2.0]:

- *Development teams need an easy way to share their knowledge* with a central database for practices and processes. Typically processes are not documented or they exist in various presentation formats. They should be easily accessed at the workplace during work phase and not be separate from the actual work process. EPF supports CVS and Subversion for version control.
- *Problems of integrating different development processes which have own formats.* Usually books, publications and companies' internal method contents and processes differ in format. It is difficult to integrate processes from different sources if they

lack widely adopted standards and defined concepts. The SPEM language provides grounds for presentation of method content and processes.

- *Teams need up-to-date information for training.* Teams need to be trained to effectively perform development processes. Training materials and knowledge base must constantly reflect the actual practices used in work processes and defined projects. The use of EPF and the published web page as a knowledge base for reusable method content in training and developing helps to overcome this problem.
- *Teams need guidance for sizing the processes in a right way.* Processes are rarely ready-to-use, but rather need to be tailored before the project and during the lifecycle. Process tools must scale both upwards and downwards on demand and have the ability to add new or tailored processes. EPF supports different configurations and publishing options allowing easy tailoring and scaling.
- *Make sure the practices are applicable with the standardized practices.* Processes need to be in a standardized form within an organization. Teams and process developers therefore need to have the ability to manage process definitions and perform auditable tailoring of these processes to individual projects.
- *Effective usage of processes in projects.* If a process cannot be used in real projects, it is not useful and is difficult to improve. The gap between process engineering and enactment must be bridged. This can be done by using similar and standardized representation and terminology. The project lead need the ability to import the processes into the project environment itself and these have to have a link back to planning elements such as roles and tasks. SPEM 2.0 provides process definition structures with enactment support.

In order for a process to follow the real life processes and be usable in software development it must evolve. One relatively efficient way to help the process to evolve is to model it. Modeling will bring up problems within the process and suggestions for improvement as it is difficult to model a development process without fully understanding and studying it.

SPEM is a standardized way of expressing any software development process. 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. Vendors like IBM, Osellus and Intel are implementing SPEM and more organizations are starting to use it to model their processes. [Sue04, EPFRel07]

The SPEM can provide necessary concepts for modeling, documenting, presenting, managing, interchanging, and enacting the development method and processes. As SPEM 2.0 is UML 2 compliant, it provides the ability to work with UML 2 tools. EPF can be used by developers, process authors, process coaches, academia, service providers, tool providers, and management. [SPEM2.0]

As a conclusion, SPEM could be considered the most significant development enabler to address the process automation needs in the market. SPEM is an open standard for modeling any software development process. Since SPEM is an open standard, customers can use a development methodology from one vendor, model the processes that support that methodology in SPEM using the process modeling system from another vendor, and enact and monitor the modeled processes from a third vendor. A process automation system would help the organizations to model, enact and monitor their processes. In this way it has a direct positive impact on the organization's quality, cost and time-to-market objectives. Organizations are able to easily create knowledge of executing a process centric project, and to use this knowledge to implement continuous quality, cost and process improvements.

6.2 How was modeling done before EPF and SPEM?

Organization have used various styles, including own specialized models or notations and for example customizations of UML diagrams. Within IBM there already were three different meta-models for RUP, IBM Global Services Method and SUMMIT Ascendant. *Unified Method Architecture (UMA)* is based on the meta-models of these methodologies and the SPEM 1.1 definition in order to unify all approaches within IBM as well as to support to most important standards in industry [OpenUPa]. UMA was used in the development of SPEM 2.0 and EPF aims to support the final SPEM 2.0 in near future. [Hau07a]

Process modeling languages can be roughly divided into *software process meta-models* and more general *business process modeling meta-models*.

Unified Process Model (UPM, also known as *Unified Software Process Model, USPM* [Kru01]) was used with older RUP versions. It is a software process modeling meta-model, as is SPEM, and contains guidelines for UML use and is expressed in UML [Oja03]. USPM is compatible with SPEM 1.0, and has only small variations in terminology. USPM is used for process authoring with the *Rational Process Workbench* tool. This tool was used to create earlier versions of RUP. [Kru01]

In [Mäk06] Mäkilä introduces a tool for modeling software process with SPEM 1.1 called Spemmet. It was developed on a web platform and was the author's attempt to get practical understanding of the SPEM 1.1 based process modeling. This tool was successfully used to model some parts, but the SPEM 1.1 standard was considered to be too ambiguous. Furthermore, the development of EPF Composer made it obsolete.

As alternatives to SPEM and EPF, there are for example *OPEN (Object-oriented Process, Environment, and Notation) Consortium's OPEN Process Framework (OPF)* [Ram06, Hen04b] and *ISO/IEC 24744 meta-model* [Gon07]. Osellus has developed the IRIS product family using SPEM for process authoring and tailoring (IRIS Process Author), and for enactment and improvement (IRIS Process Live) [Ose07].

Domain Specific Language (DSL) tools or *meta-CASE* tools can also be used for software process modeling [Kel07]. These tools support the generation of a specific environment, new modeling languages and method support for these languages. Examples include Microsoft DSL tools or MetaEdit+ from MetaCase.

The more general business process modeling meta-models include *Yet Another Workflow Language (YAWL)*, *Business Process Modeling Notation (BPMN)*, *Business Process Execution Language (BPEL)*, *Process Interchange Format (PIF*, used in MIT, Stanford, etc.), *Process Specification Language (PSL)*, *Core Plan Representation (CPR*, used by DARPA), *Workflow Management Coalition Process Definition (WfMC)*, and *Architecture of Integrated Information Systems (ARIS)* [Bre00, YAWL07, Bro03, Mäk06].

YAWL is based on workflow patterns and is considered as a powerful workflow language [YAWL07]. BPEL is an XML language for composition of web services [Bro03]. The WfMC is a non-profit, international organization of workflow vendors, customers and users. It aims at producing standards such as the Process Definition. [Bre02]

We have now covered some history of the SPEM 2.0 evolution and the tools used before or instead of the EPF Composer. There are also various process meta-models beyond the software development scope and some of them were mentioned. In the following section the focus is on the specific uses of EPF and SPEM.

6.3 How have EPF and SPEM been used?

EPF has 26 committers from different organizations, over one thousand downloads per week and over 100 press references according to the 2007 statistics. [EPFRel07]

Three methodologies have been modeled with EPF, which can be downloaded from the Eclipse Process Framework homepage. These are *OpenUP* (see section 4.2), *XP* (see section 4.4) and *Scrum* (see section 4.5). The current major EPF version 1.2 was released on August 1, 2007 with added contributions such as the OpenUP/DSDM (*Dynamic Systems Development Method*). Agile Modeling and translation of the content are still in progress. According to the *EPF newsgroup*, there are several cases where current method content (OpenUP for instance) is being translated into different languages. EPF native language support is also growing constantly. EPF has currently native language support for Chinese, Danish, French, German, Italian, Japanese, Korean, Portuguese, Russian and Spanish. [EPF07, EPFRel07]

In [Hau06, SPEM 2.0] the case studies include:

1. Fujitsu DMR MacroScope, which shows how SPEM 1.1 processes can be migrated into the SPEM 2.0 format.
2. Microsoft Solutions Framework (MSF) Agile case study is a proof of concept that SPEM 2.0's concepts are sufficient to model all of MSF Agile's process concepts and structures using EPF Composer.
3. MDA Process (OpenUP/MDD) is an extension to OpenUP/Basic, which models OMG's Model-Driven Architecture approach.
4. IBM Tivoli Unified Process (ITUP) models processes using SPEM 2.0 concepts and guidance to help understanding and making ITIL recommendations directly actionable.
5. PMBOK, Sierra System used all major SPEM 2.0 concepts to re-model the *Project Management Body of Knowledge (PMBOK)*.
6. SOA Governance Lifecycle and Management Method, shows that SPEM 2.0 has also been used to represent organizational governance processes.
7. OnDemand Process Asset Library (OPAL), represents a CMMI process repository developed with SPEM 2.0. Building blocks could be selected and assembled into a process based on the needed maturity level.
8. E&TS Application Specific Integrated Circuits Method shows that SPEM 2.0 has also been used to model hardware developing processes. Different methods were created for project consistency and documentation and training were used by both project managers and technical teams.

EPF committers have also an interest in knowing about the OpenUP deploying. Questions on the homepage include: "There have been thousands of downloads of the process but we haven't received much feedback on the results people are experiencing. Understanding the successes and failures the OpenUP community is experiencing will help us to deliver more value in the future." [EPF07]

EPF can be conveniently used with *Wiki Technology* (Figure 6.3.1) [EPFW07]. This has been used for example within the EPF community to develop method content. Wiki can be used to gather useful feedback by writing comments around content or editing content, e.g. the experiences are captured through harvesting. Communities can be built around key

content areas, and process content improved without learning SPEM or EPF Composer [Kro07b].

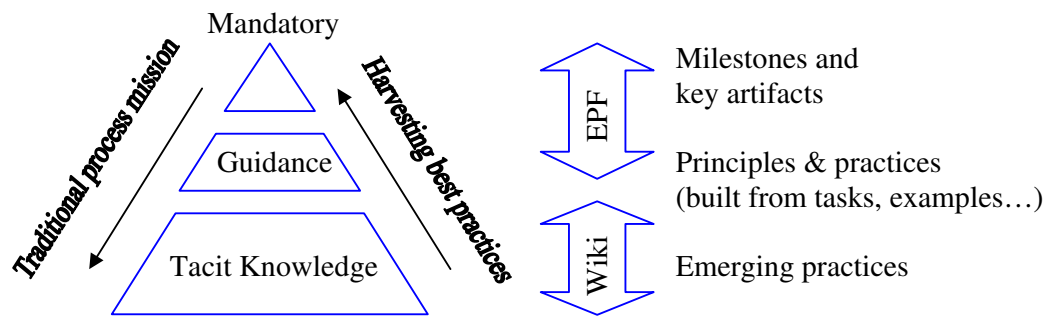


Figure 6.3.1 – EPF with Wiki technology [Kro07b]

Eclipse Plug-in Central introduces how organizations are leveraging EPF [EclIPC]:

- *IBM Rational Method Composer* is used to create current versions of RUP, see section 5.3 for details.
- *Agile Enterprise Architecture* defines how ‘Enterprise Architecture’ is done in an agile way.
- *Agile / Iconix Road, Iconix Process* has been modeled using EPF.
- *Wilos (Wilos Is a cLever Orchestration Software)* helps team members to manage projects using exported XML file from EPF. It includes a personal assistant which provides guidance for following the selected process, displays project items, manages tasks etc. [Wil07]

Now we have covered modeling before EPF and SPEM, the usage of EPF and SPEM, and some alternative approaches. In the following sections the focus is on the capabilities of EPF Composer 1.2 and SPEM 2.0.

6.4 Capabilities of EPFC 1.2

The EPF capabilities can be divided into Method Authoring, Process Authoring, Library Management and Content Extensibility and Configuring and Publishing [Elv06].

Method Authoring

The organization's best practices can be captured as a set of method content as defined in the SPEM 2.0 metamodel: roles, work products, tasks and guidance (templates, white papers, examples, check lists, etc.). These can be reused through extensibility, replacement and contribution, and categorized as desired. A rich-text editor allows the documentation of elements, links to other elements, files, web-pages or pictures. Graphical diagrams show the relevant relationships.

Process Authoring

Reusable process content can be organized into processes along a lifecycle dimension by defining Work Breakdown Structures. EPF Composer allows the construction of reusable process chunks called capability patterns. A capability pattern may define how to design, implement and test a scenario or a user story, and this pattern can be reused in processes. The capability patterns are used to create a delivery process, which describes a complete and integrated approach for performing a specific project type.

Library Management and Content Extensibility

An XML-based library enables flexible configuration management and content interchange for distributed client-server implementations (CVS, Subversion). Method and process content can be packaged into plug-ins allowing simple distribution, management and extensibility. Process content can also be exported to tools such as MS Project.

Configuring and Publishing

A process configuration can be created by selecting a set of plug-ins and packages. Publishing allows selecting the configuration and processes with variations in the layout of the produced html-pages.

EPF Composer version 1.2 has the following key features added [EPFRel07] which were noticed during the modeling case: the support for new SPEM 2.0 innovations (variability and fine-granular configurations, which were applied to the modeling when the version

changed from 1.02 to 1.2), process, breakdown structure and Rich Text Editor improvements (which were used to improve the appearance of diagrams and method content), and the support for additional platforms and technologies.

6.5 Sufficiency of SPEM 2.0

The process modelling language used reflects perspectives and approaches taken on systems development [Kos99]. It can also constrain the aspects that can be captured during process modelling and supported during the actual process. The most frequently addressed aspect of a process modeling language is its paradigm (vs. abstraction, modularity or genericity).

Process modeling methods are based on widely varying philosophical, theoretical, and conceptual foundations. We need approaches for defining process modeling languages of different foundations, e.g. MOF. The process meta-metamodel specifies the premises that can be used to specify process metamodels.

A process modelling language contains a *process ontology* (conceptual framework), a *notation*, and *language semantics* [Kos99]. An ontology for processes determines in which terms and ways we abstract, discuss and determine software processes. A notation is the system of signs or symbols which represent a conceptual framework. Semantics deals with the meaning of signs and symbols.

The SPEM 2.0 conceptual framework provides the necessary concepts for standardized representation and managed libraries of reusable method content, support of modeling, documenting, managing, configuring and enacting development methods and processes. SPEM 2.0 reuses some key classes from UML 2 infrastructure and defines the notation of specific process diagrams. SPEM 2.0 documentation also contains full semantics for the meta-model. [SPEM2.0]

The references on SPEM 2.0 are so far mostly EPF/SPEM community based and might focus more on the bright side of the meta-model. The critics presented in references are mostly for SPEM versions 1.x as the SPEM 2.0 definition is still recent. Next we will study the new capabilities of SPEM 2.0 while discussing the shortcomings depicted with SPEM 1.x and how SPEM 2.0 will remedy these issues.

As SPEM 1.1 can be migrated to SPEM 2.0, we can conclude that SPEM 2.0 has the sufficiency of at least SPEM 1.1. In addition, SPEM 2.0 takes advantage of the UML 2 standard and its new functionality. It defines extensions for enactment and has a clear separation of method content and process content. In addition to this, it supports many different lifecycle models, has flexible process variability and extensibility plug-in mechanism, and reusable process patterns [SPEM2.0].

In [Hen04a] the study includes the flexibility and compliance of different frameworks and meta-models. SPEM 1.0 is depicted as following:

- flexible technique-to-task mappings,
- process fragment selection exists,
- capability levels are not considered.

As previously noted, the two first observations apply with SPEM 2.0, but SPEM 2.0 also supports the concept of capability levels through variability and extensions.

The SPEM 1.1 has been examined in [Jär05] as following:

- shortcomings of process components with ambiguous definition,
- modeling agile methodologies require a different approach,
- a challenging organization of process components.

The process component definition has become less ambiguous; self-containedness constraints and 'Unification' mechanisms have been removed and replaced with Ports and Work Product correspondence concepts. 'Refers To Dependency' has also been deprecated. The loosely connected process fragments and flexibility of SPEM 2.0 suit agile development better than SPEM 1.1. The organization of a process component could, however, create challenges also with SPEM 2.0. For example, when making a process of RUP disciplines it is clearly seen that one discipline interacts with the others and is connected to many phases. This generates a complicated relationship between components. These problems might, however, arise from a conceptual level above SPEM.

Ramsin introduces SPEM 1.0 with the following strengths [Ram06]:

- Flexibility and configurability due to the generality of the meta-model (though there are limitations because of dependency on RUP as meta-model basis).
- A well-defined general framework.
- Well-formed rules when instantiating processes.

And with the following weaknesses:

- A process component library is not included.
- Specific procedures for instantiating a software development process using the meta-model is not offered.
- Lacks detailed specification documents: the OMG document is a very general description.
- Lacks sub typing for important process components, which results in the meta-model being of very limited practical use: Poor coverage of lifecycle activities and lacks explicit support for umbrella activities. Modeling and artifact production issues are not explicitly addressed.
- Mainly for modeling processes similar to RUP; limits applicability and generality
- The developer is responsible for constructing the methodology. The rules are not enough to prevent poor instantiations.

Let us now analyze the weaknesses compared to SPEM 2.0. The package is now divided into Method Package and Process package, allowing a specific process component library. The SPEM 2.0 meta-model can be directly instantiated for an implementation; however, a specific procedure is not introduced. The SPEM 2.0 is much more extensive and contains more details and examples than the SPEM 1.1 definition. Process components are renewed as shown before with [Jär05] analysis. SPEM 2.0 supports many different lifecycle models, and has also been used in modeling agile methodologies such as XP and Scrum. Finally [Ram06] points out the developer's responsibilities in constructing a methodology, which should be obvious with all meta-models.

As a conclusion, the shortcomings of SPEM 1.1 discussed in various references have been mostly corrected in SPEM 2.0. There have been several contributors participating in this development [Ben05, SPEM2.0]. The sufficiency of SPEM 2.0 can also be evaluated by its usage and case studies, which were covered in section 6.3. The modeling with EPF and SPEM is covered in the following chapter and the study of EPF and SPEM 2.0 being suited for the modeling of OpenMethod is discussed in chapter eight.

7. Modeling process based on an existing model with EPF / SPEM

In the previous chapter we examined the capabilities and usage of EPF and SPEM. In this chapter we discuss the modeling with EPF and SPEM on a general level. The specific case study is in chapter eight.

First we study the actual modeling activities and then move on to concentrate on EPF customization and tailoring, process enactment, and finally review the problems and benefits notified on the general level.

7.1 Modeling with EPF and SPEM 2.0

Modeling can be thought as reverse engineering as the model is created from a system [Bez07]. The model, however, is usually created from an existing model instead of the actual process.

Modeling creates the process models and is thus a fundamental activity in adopting process modeling. The basic objective of a process model is to answer the following questions [Bre00]:

- What is to be done and when?
- Who does it?
- What is produced?

Depending on the use of process models in an organization, the required accuracy, level of detail, the amount of work effort, and the frequency of modeling work varies [Mäk07]. SPI activities (section 3.3) could be applied in parallel with the modeling activities. In this section, however, we will mainly focus on modeling. The reason behind modeling is often the need to improve the process.

The model has to be easily understood. The idea is that any good model exhibits a single, coherent vision. Conceptual integrity is the degree to which a model can be understood by a single human mind, despite its complexity [Lan05].

According to [Mäk07], the modeling process as a ‘light-weight approach’ is the following (see Figure 7.1.1, articles [sic]):

1. **Start of modeling project:** define goals, purpose, scope, the participants and the schedule of the modeling project. There should be one lead modeler responsible for the modeling work.
2. **Construction of initial model:** the initial model is constructed based on the existing process documentation. The initial model can be created by identifying roles, work products, tasks, phases, iterations and milestones in the documentation. The process elements and their immediate relationships are then modeled with EPF Composer.
3. **Verification of initial model:** process authors of the existing documentation verify that the initial model is in line with the existing documentation.
4. **Planning of interviews:** interviews clarify unclear parts of the model, find contradictions between process documentation and the actual process, and examine how different stakeholders use the process.
5. **Execution of interviews:** the interviews can be done in sequence or independently from each other. The former approach increase the modeling speed and the latter the objectivity of the model.
6. **Refinement of model:** model is refined based on the comments. EPF Composer with a version control system provides powerful tools for flexible modeling.
7. **Verification of final model:** the verification meeting with participants assures that the modelers have understood the interviewees and that modeling is done correctly.
8. **Closing of project:** the lead modeler stores the model properly, distributes it to all necessary parties and closes the project.

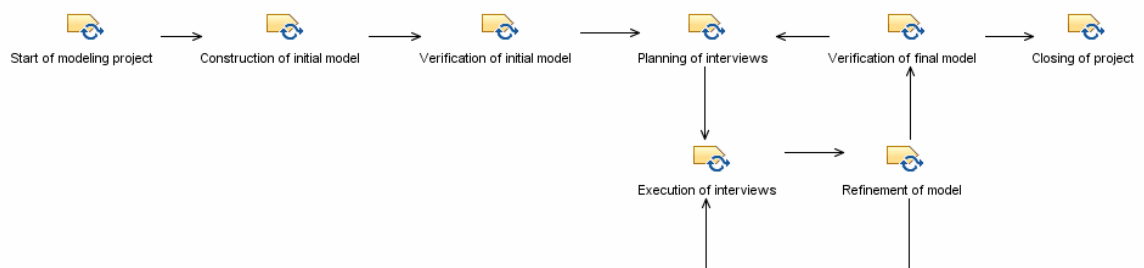


Figure 7.1.1 – Modeling approach modeled with EPF Composer 1.2 [Mäk07]

By-the-book modeling can be more complex than the light-weight approach, but the main concept is similar. We will assess these depicted modeling activities with our actual modeling experience in section 8.3. Maintaining the model should also be observed, and other possible measurements, such as verification, should be utilized to ensure the correctness of the model.

Models are created for knowledge creation and knowledge transfer (see section 3.3.1). The development process model created is not the ultimate goal, and not even the only product of the modeling process. Transformations in knowledge, agreements and commitments in people's minds are at least as important. [Lan05]

7.2 EPF customization and tailoring

Process flexibility and customization is one of the key elements for successfully adopting a particular process in real life. To assess flexibility and capability for customization of existing processes, some objective evaluation criteria must be developed. [Hen04a]

Organizational variability, which is usually the reason behind process customization, can provide the following thoughts:

- Different organizations have different skills and techniques. Organizational culture often determines which techniques are used.
- Different projects often demand different levels of formality and accuracy. All projects do not require the same tasks to be done.
- Organizations perform at different capability or maturity levels. Those on higher levels can perform jobs that the ones on lower levels cannot.

Some criteria can be derived from these facts as follows [Hen04a]:

- Does the process allow the desired technique selected for each task?
- Does the process allow the selection of tasks and work products to be used in each project depending on the characteristics of the project?
- Does the process allow customization of tasks and work products depending on the organization's capability level?

EPF supports variability elements, which can be used to choose the appropriate elements for, for example, each capability level. The configuration and publishing options allow the selections of desired elements further on.

A customization of the process for different uses both inside the organization and outside the organization (e.g. the process is tailored for a customer) also requires different points of view. EPF itself and published contents can be customized in several ways, from style sheets and languages to content and process tailoring, depending on customer or project needs.

EPF uses a specific working folder for method content. If a process needs to be tailored for another use, one possibility is to do that in a different working folder. This, however, might complicate the maintenance process as the common building blocks would be separated. In a situation where, for example, one additional role is added, the same role has to be added to both folders. One solution is to use CVS as the main database which contains all the necessary method content (tasks, roles, guidance etc.) and process information. When a new role is added, it is also added to CVS, and the other users or work folder instances will be able to update to the new changes.

A more elegant solution for customizing is to use different configurations depending on the situation. There can be many configurations in the same working folder. The included plug-ins and packages can be fully selected in the configuration. It is also possible to tailor the views for the publishing version completely. For example, if a new configuration needs to be created without the 'analysis and design' discipline, processes relating to inception phase and work products, the new configuration is simply created with a new view which only has the needed content and categories. It is also possible to exclude the unwanted packages in the package selection.

A user can make various changes in the publishing of a configuration. Published processes can be freely selected and many other minor variances can be made. These include publishing a glossary, an index and choosing a title and a banner image. Search capability can be added when creating a website as a Java EE web application packaged in a WAR file. The publishing operation also supports changing several details in layout and diagrams.

Tailoring of the published results, the html pages, is practically unlimited. The result of direct html tailoring will, however, disappear in each publishing. One way to alter already published pages is to create a modification packet (zip), which will be installed on the published packet. The modification packet could include modified style sheets, language packets or Java scripts to enhance the published version. As the layout of the published pages is an important factor in organizations, more efficient customization of published pages is needed. A possibility for adding enhancements in the layout of the published pages could be utilized.

Native language support (NLS) consists of ten languages at the moment. For example, executing “epf -nl fr” starts the French version of EPF, but the contents naturally remain original (English). Language packets (NL1, NL2, NL2a) can be used for additional language support. As the texts are in properties files, it is easy to translate the EPF user interface or published non-content texts to any desired language. The EPF web page has detailed instructions for translating the content that comes with EPF (e.g. OpenUP).

EPF Composer is an open-source project which allows complete tailoring of EPFC itself if additional special features are needed.

7.3 Process enactment

Process enactment means evoking process performance according to a process model either by a human or a machine [Kos99]. The ability to modify the process model to reflect knowledge gained from real projects is necessary for process improvement. This requires mechanisms for enactment. Since SPEM 2.0 covers notations, concepts and semantics, it serves not only as a specification for a process model but also for enactment of the process.

The two most common ways of enacting the SPEM 2.0 meta-model are [SPEM2.0]:

- Mapping the processes into Project Plans and enacting these with project planning and resource management tools such as *IBM Rational Portfolio Manager* or *Microsoft Project*.

- Mapping the process to a business flow and then executing this using a workflow engine such as a *Business Process Execution Language (BPEL)*-based workflow engine.

The processes defined by SPEM 2.0 breakdown structures contain information attributes (hasMultipleOccurrence, isRepeatable etc.) which aid the Project Planner in making correct instantiation decisions. Plans can also be mapped using a work product breakdown structure which is suitable for agile environments. In these, the plans for work activities are not followed step-by-step; instead, they define which resource produces which work products in which state for which date.

EPF Composer supports export to XML and MS Project for enactment or other use. Microsoft Project is a common planning tool which is not based on an explicit meta-model. MS Project meta-model is based on the project concept, which is described as a set of tasks. These are ordered with links and assigned to different resources. The creation of a project plan from a process model can be seen as a model transformation between SPEM and MS Project meta-models. [Bez04]

The idea of models as exchanged artifacts may require model transformations. These transformations can be carried out on projects and documents. Model transformations provide the integration of different tools and expand the possibilities of model usage. [Bez04]

7.4 Benefits and problems experienced with modeling

Modeling with EPF Composer presents benefits and problems which are now discussed from a more practical point of view. The OpenMethod specific observations are introduced in section 8.3.

The first major EPF version 1.0 was released on October 2nd, 2006 and the latest, 1.2, on August 1st, 2007. There have also been three smaller releases between these and a 1.2.0.1 update on September 14th. During modeling, the version changed from 1.0.2 to 1.2. New versions are compatible with the older ones, but the method content created with the new

EPFC cannot be read by an older version. The new versions convert the content to a new format. With 1.0.2 version there were problems with the conversion if the content names included Scandinavian letters, but version 1.2 did not seem to suffer from those.

The changes in EPF versions are easily adapted, and therefore no special training is required for old EPFC users to use the new EPF Composer. Nevertheless, training or support material is still needed for those users using the old (pre-EPF) model and for process authors. The EPF-modeled version will probably look and feel different, but with a proper mindset and attitude, other methods and materials with SPEM notations can be easily understood.

Resource files for a custom language must be re-checked and installed separately for each update. This should also be done when installing EPFC on a new computer. The easiest way is to have a separate packet for the resource files for each EPFC version. This packet can be installed after the EPFC installation to provide the desired language support.

The development community is strong and active at the moment, but if the EPF project ends, what will happen to development? The latest version can be used and the content improved, but possible bugs in EPFC itself will remain unfixed and tool development stops. EPF is however, open source, which will enable other inspired contributors to continue the work. The organizations using EPF may also continue to develop it to some extent.

The rich text editor (even with the new EPFC 1.2 features) is rather simple, and making the content visually convincing will probably require the use of direct modifications to the HTML code or links to PDF or other presentation files. The publishing layout is also simple and a tool should be developed to provide better customization and enhancements to the layout. The style sheets should be chosen within the EPFC publishing tool instead of modifying the published HTML or CSS pages afterwards. The design for different browsers and resolutions should also be taken into account. Glossary and index pages are also visually very simple.

Modeling of roles can be straightforward when modeling an existing model. Nevertheless, the concepts must be clear when creating a new model or introducing the role. The role is

not an individual. One way to face this problem is to create a table mapping roles to job positions, or to use the job position directly, especially in non-technical activities. This also helps avoiding that they are too abstract. The work products cannot be separated into output and input – the same instance exists in both cases. For example, ‘code’ is input but also output to ‘testing’. There is no way to distinguish between these two, even if ‘code’ is changed during ‘testing’. The same observation has been made by Gonzalez-Perez as he is introducing the ISO/IEC 24744 meta-model [Gon07].

Let us now focus on some more specific problems and improvement issues regarding EPFC 1.2. The following shortcomings were reported to Bugzilla and are waiting to be fixed:

- No notation exists for specifying the direct relationships between tools and Work Products, Tasks or Roles. Guidance must be added through Tool Mentors.
- Visual layout problems in diagrams in content with long names (e.g. the Finnish term ‘järjestelmäsuunnittelija’); EPF Composer could support hyphenation
- Diagrams are saved in .JPEG format instead of lossless .GIF or .PNG
- For search capability, the page should be published as JAVA EE, but EPF doesn’t include any guidance for this and the usage of the page
- Process pattern synchronization has to be done manually instead of automatically

There were also some other minor bugs which were reported and these will probably be corrected soon. Some of the problems can be easily overridden; the image size is limited to 600x600 – this can be altered by directly editing the HTML code, disciplines are ordered automatically – renaming with a starting letter overrides this. The documentation is also continuously growing and being enhanced. The conclusion regarding EPF Composer is that it still has some minor drawbacks, but it is continually being improved. Practically, the tool is stabilized.

In this section, we have discussed some general modeling aspects, customization, and specific problems observed when modeling with the EPFC tool. In the following chapter, we will concentrate on the modeling of OpenMethod.

8. Modeling OpenMethod™ with EPF and SPEM 2.0

In the previous chapter, we introduced modeling on a general level. In this chapter, we discuss the reasons for modeling acquired from the user questionnaire, the timetable, the modeling of OpenMethod with EPF and SPEM, and the OpenMethod user interviews. More extensive and detailed information is found in the SD's internal material.

8.1 Why model OpenMethod with EPF and SPEM?

In section 2.5, we introduced the reasons for modeling. The more specific arguments for modeling with EPF and SPEM were introduced in section 6.1. These previously found benefits and drawbacks can also be applied to modeling OpenMethod.

The specific reasons for OpenMethod modeling have been collected from interviews and questionnaires. The interviews were conducted after the initial model and are introduced with the benefits and challenges in section 8.4. In this section, the focus is on the questionnaire made before the initial modeling of OpenMethod. The questionnaire was a part of an OpenMethod user survey.

Answers to the questionnaire were received from over 160 people. Their roles included a wide variety of professionals from project managers to developers and testers. One third of the informants used OpenMethod as their primary methodology. Others mainly either used customers' methodologies or did not use any specific methodology at all. More than half of the informants had used OpenMethod, primarily design and analysis, though other areas were also applied. Over half of the informants used a methodology daily, weekly or at least monthly. The following results were chosen based on importance and frequency.

The strengths of OpenMethod were considered to be as follows:

- Includes good, useful materials, guidance, templates, examples and check lists
- Coherent approach to development, provides credibility
- Works with bigger projects
- Practical and easy to approach compared to RUP, which provides only a framework
- Good support for tools, especially IBM/Rational
- No need for everyone to reinvent the wheel

Drawbacks and needs for development were as follows:

- More focus needed on development phase, add best practices for CVS and ISM use
- Ability to use agile methodologies such as Scrum or XP with OpenMethod should be added
- OpenMethod should be easily found and sufficient training should be available
- Merge of processes should be possible during organization integrations
- Process use in projects should be more standardized, while still allowing customization
- The methodology should be easier to understand and use, and have a good focus

The process authors emphasized the importance of making the OpenMethod structure more coherent (each author may have their own styles), and enhancements to usability and maintenance. The SPEM standard will provide the required common concepts and terminology. The linkage between elements and the management of up-to-date material supporting CVS is also important. Adding the ability to customize the process for specific needs and publish different versions should also be considered.

8.2 Modeling time table

The actual modeling took less than 80 hours. The meetings, background work, studying EPF and knowledge work around the subject should also be considered. The whole thesis project was started in January, 2007. The estimate was to finish before autumn and the rough estimated time for the whole project was 500 hours. First the subject, main concepts and focuses were chosen. The meetings, gathering of background materials, and their study took most of the time. The actual modeling started in March, and at that time the focus was on studying EPF and its usage. The modeling was finished at the beginning of June. The actual time usage is shown in Figure 8.2.1.

The meetings, studying and other work should also be taken into account when estimating the time needed to model OpenMethod completely. The creation of the modeling instructions and other guidance materials was divided equally between the modeling and thesis writing. The detailed estimations of the whole modeling work are depicted in the internal materials. The modeling included parts of ‘analysis and design’ and ‘testing’. The

OpenMethod structure was also modeled, including all roles and the introduction. This provides a more extensive view of the possibilities of EPF and SPEM.

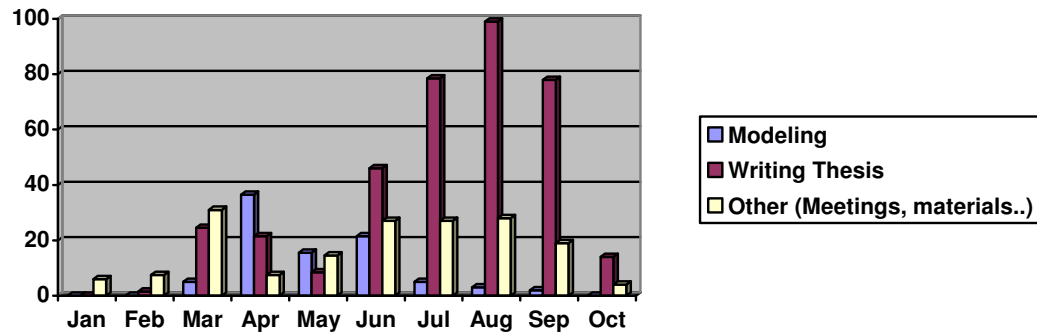


Figure 8.2.1 – Working hours spent on thesis (2007)

The original estimations were well achieved. The time used for final improvements, checks and proof-readings increased the whole time used. 610 hours were spent on the project and the thesis was finished at the beginning of October, 2007.

8.3 Modeling OpenMethod

The modeling was started at the end of March. Before the actual modeling, several meetings were held. In these meetings, the scope, reason and goals were defined. The scope included modeling of the ‘design and analysis’ and ‘testing’ processes and disciplines. To have a more in-depth comprehension of the model, it was also decided to model the OpenMethod framework (skeleton). This included all the roles and a fully functional tree browser (e.g. a navigational panel). The overall thesis schedule and participants were applied to the modeling part, too.

The new model was constructed based on the current version of OpenMethod. The results were frequently discussed with the process author. This part differs from the modeling theory [Mäk07] covered in section 7.1. EPF Composer (see sections 5.2 and 6.4) was used in the modeling process. Figures 8.3.1 (current OpenMethod) and 8.3.2 (modeled with EPFC) show the differences in the main published view.

8. Modeling OpenMethod™ with EPF and SPEM 2.0

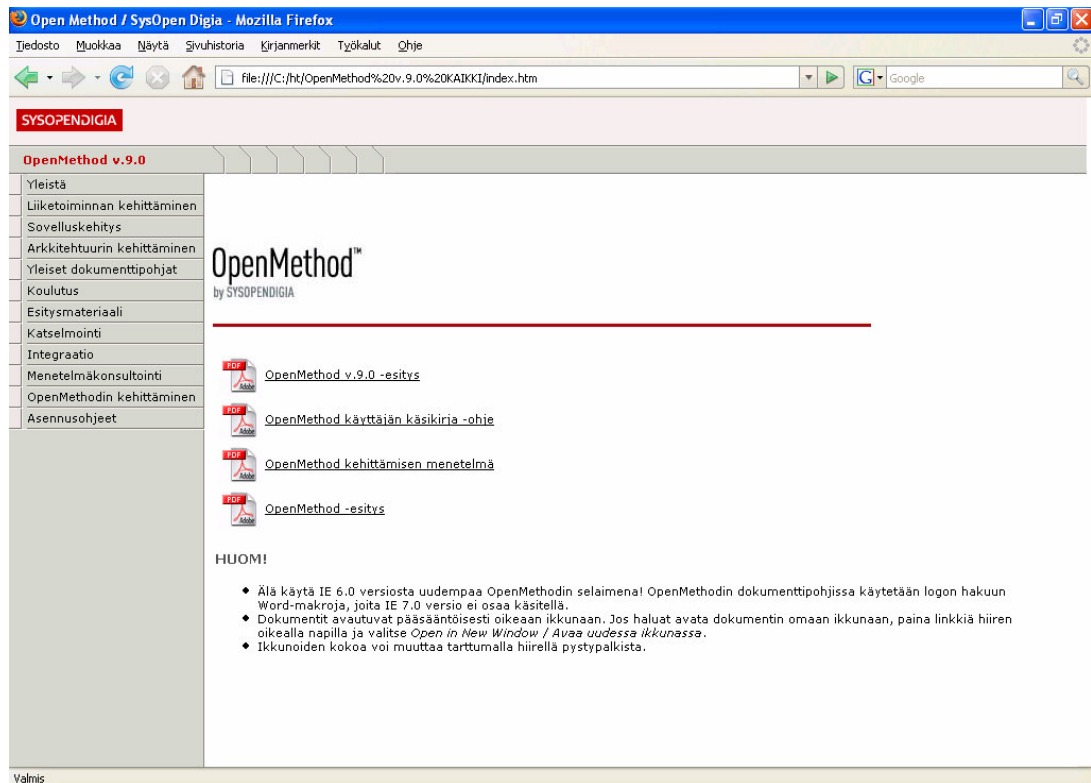


Figure 8.3.1 – The original OpenMethod opening page (in Finnish)

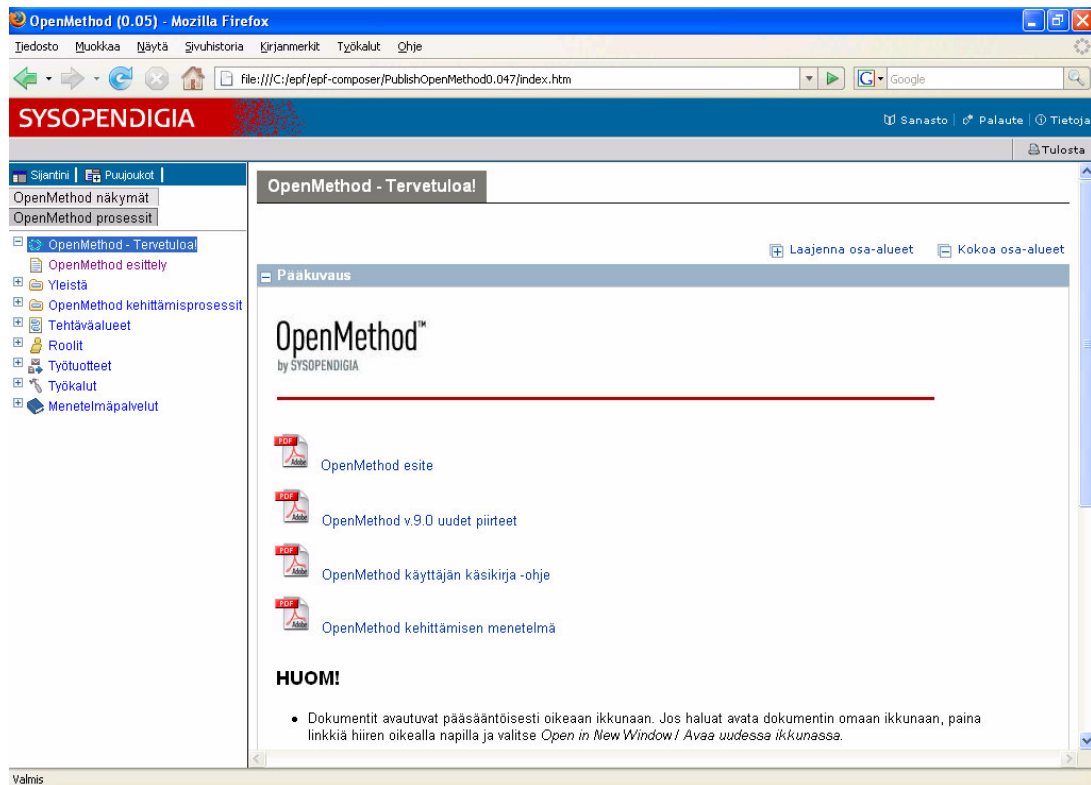


Figure 8.3.2 – OpenMethod opening page modeled with EPF Composer (Finnish)

Both the original and the EPFC version are generated on a web page. EPFC version has tabs in the tree browser, whereas in the original they are over the main frame. EPFC version also has a ‘where am I?’ link to ease navigation, as well as easy printing, glossary, and feedback functions. The method content in pages is structured into frames depending on the content type (descriptions, relationships, etc) and can be expanded and collapsed.

The modeling began with the modeling of the roles, work products, tasks and guidance. These were categorized to the appropriate packages in *Method Content / Content Packages*. Figure 8.3.3 shows a compacted library view (the dots replace other method content). To avoid long lists of tasks or work products, some content packages were divided into smaller ones. Modeling the roles was rather straightforward. The general information with presentation names and brief descriptions was added. The ‘detailed’ and ‘staffing information’ was not included in this initial model because ‘brief descriptions’ covered enough information for now. Some roles also required modifications, and it was therefore decided to add the more detailed information later. The work products and guidance were modeled in the same way. In guidance, however, some links to PDF files were added to provide additional information, examples, or templates.

The tasks were modeled with the descriptions and the steps. The roles were linked to the tasks as performers, the work products as inputs and outputs, and the guidance as guidance. OpenMethod has more performer types and all these were added as additional performers.

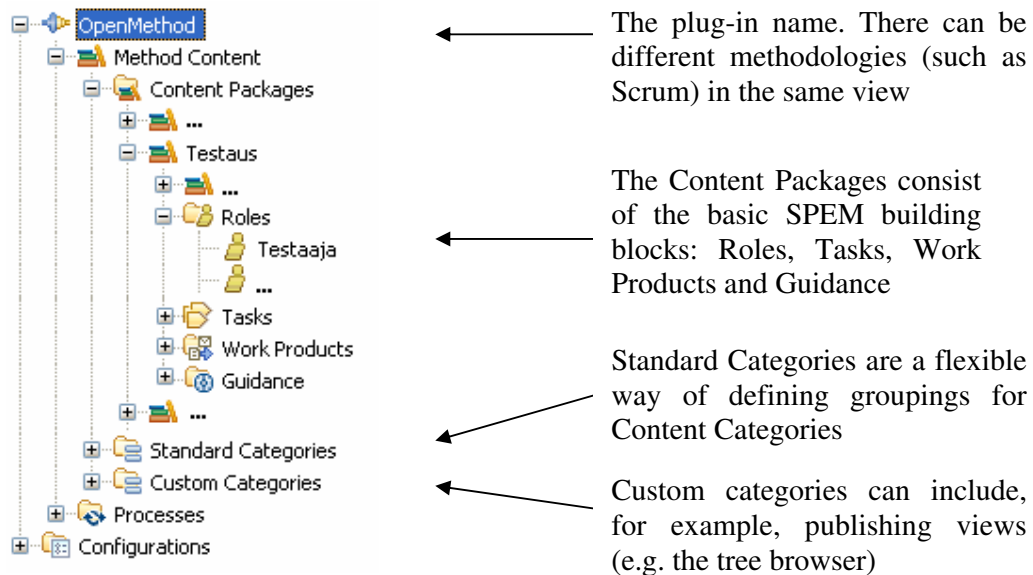


Figure 8.3.3 – A part of the EPFC library view for the OpenMethod content

As discussed in section 3.3, modeling is fundamentally an SPI activity. The goal was not only to gain experience of EPF Composer, SPEM 2.0, and the modeling process, but also the process improvement. Descriptions of the method content were updated and standardized. There were also some obsolete roles so the role structure needed to be simplified. With an up-to-date and clear content linkage, the mistakes with content couplings were also easier to find.

After the content packages were completed, the next task was the categorization of content into *Standard Categories* (see Figure 8.3.4). Disciplines for ‘analysis and design’ and ‘testing’ were created and the appropriate tasks and guidance were added. ‘Analysis and design’ was actually defined as a discipline grouping and divided into smaller disciplines to avoid long lists of tasks and to ease the categorization. Grouping was rather straightforward as the current OpenMethod was well categorized and EPFC used the same logic in its categorization.

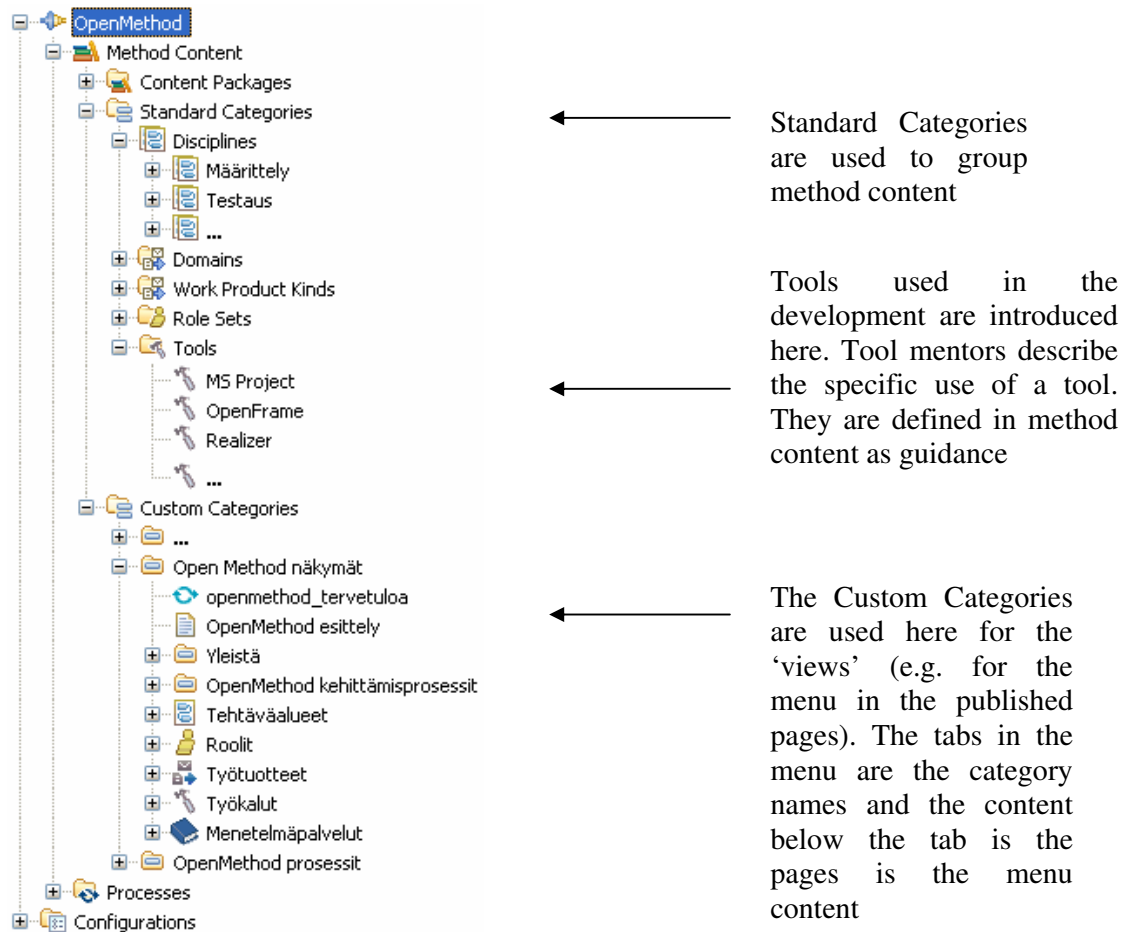


Figure 8.3.4 – EPFC library view of the Standard and Custom Categories

Next ‘analysis and design’ and ‘testing’ domains were added with the appropriate work products. A role set with all the roles was added to ‘Role Sets’ and a separate ‘Role Set Grouping’ was created with roles belonging to each discipline. Tools were completed with the most used tools. These tools were linked to the tool mentor guidance. After the concept of organizing content had been well understood, this work was done rather easily.

Customer Categories and the idea of the *views* were a bit difficult to understand at first. Custom categories can be used to create any method content. Other content can also be assigned to them. One fundamental use of custom categories is to create views (e.g. the tree browser or the menu) of the published content (see Figure 8.3.4). Icons can be freely selected for the items to clarify the content type. Two custom categories were created for the tree browser. One was for main contents and the other one concerned processes. Custom categories for the views can be selected in configurations. They are discussed in more detail later in this chapter.

After the method content was modeled, main focus was shifted on the process content (see Figure 8.3.5). Processes are divided into *Capability Patterns* and *Delivery Processes*. A Capability Pattern (Process Pattern) is a reusable building block for creating new development processes. A Delivery Process is a special Process describing a complete and integrated approach for performing a specific project type. As the creation of a Delivery Process would require method content from various disciplines, two Capability Patterns were modeled instead of creating a Delivery Process. A Delivery Process could be later on created from the Capability Patterns and the process would consist of several disciplines.

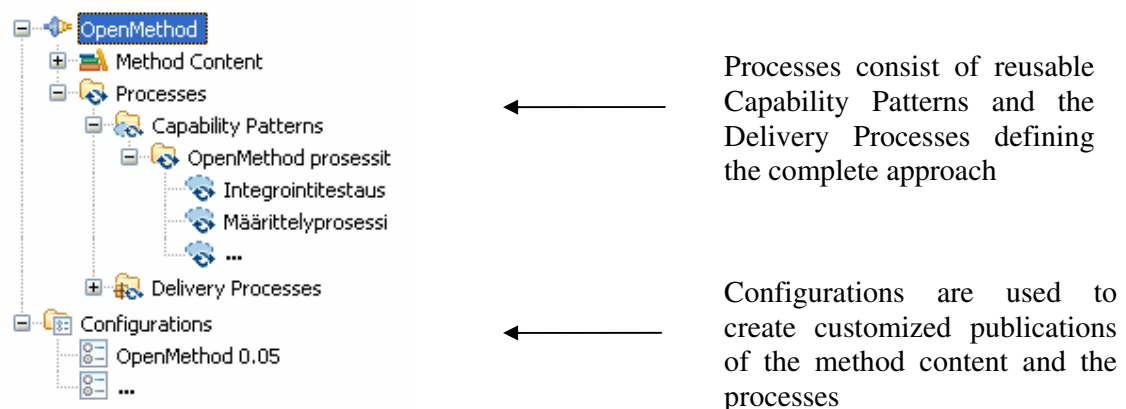


Figure 8.3.5 – EPFC library view of the Processes and Configurations

Capability patterns for ‘analysis and design’ and ‘testing’ were created. As each of the tasks was only from one discipline, adding the method content was simple. The processes were divided into phases ending with a milestone. The diagrams were modified a bit to evaluate the capabilities of the diagram editor. The processes would require more studies in order to directly use them in real-life projects (e.g. enacting to MS Project, etc.) Now the processes act as a general overview and express the possibilities of EPF Composer.

Method configurations (see Figure 8.3.5) are the basis for publishing method content and processes. Management of the configurations allows the user to select the working sets of content and processes for a specific project. As the content and processes are organized into plug-ins which are further organized into method packages, the configuration is simply a selection of the required plug-ins and packages. The same library can contain, for example, material for the whole process lifecycle, but publishing could be done for the ‘analysis and design’ phase only. The configurations can also be used to create customer specific publications which replace or extend the base material.

The OpenMethod configuration was created with all the previously created OpenMethod content. Views were added to this configuration from Custom Categories. An extra configuration for ‘Customer X’ was also created with some of the tasks replacing and extending the basic OpenMethod content. This was done for testing the content variability.

When all the method and process content had been modeled, it was time to evaluate publishing. Note that the content was added iteratively and publishing was initially done quite often to study the usage of EPFC. The contents can be previewed in the *Authoring perspective*, and browsed in the *Browsing perspective*. Publishing the configuration generates the HTML pages which present all the method content and processes of the particular configuration. In most cases, the default publishing options were used. Various options were tried for testing different layouts and the OpenMethod logo was used as the banner image.

The published opening page of OpenMethod can be seen in Figure 8.3.2. The first problem with the published pages was the language difference; the content was modeled in Finnish, but EPFC element names were in English. The required language changes were made to several properties files. This way the whole publishing version could be in Finnish. EPFC

includes the native language support (packets NL1, NL2, NL2a) and there is a framework for the Finnish language in packet NL2a. This should be used when using Finnish language with EPFC to a greater extent. The language properties should, however, be checked during the installation of a new EPFC version and when installing EPFC on a new computer.

8.4 Modeling results

Overall the real modeling activity followed closely the theory of modeling covered in section 7.1. The main difference was the concentration on the initial model with the process author and iterating at the beginning. The interviews were done at the end and the suggestions and observations at that point were mostly on a general level.

A part of the model was stored on a Subversion repository, but the actual team work for method authoring was not performed any further. The repository itself seemed to work fine and there is a lot of experience of authoring using a version control system in the EPF community.

The capabilities of EPFC (section 6.4) and the sufficiency of SPEM 2.0 meta-model (section 6.5) were suitable for this modeling project. The general benefits and problems experienced (section 7.4) can be applied also to this specific OpenMethod modeling. The problems tended to be mainly minor layout issues. Method content should be modeled more to have a better understanding on the modeled processes and of the use and capabilities of diagrams. The enactment of the process and usability with tools like MS Project would also require more material.

The differences with the original OpenMethod and the published EPFC version are clear. The content is linked in the EPFC version and is therefore more consistent. The layout of publishing is fixed, but can be easily modified to some extent with the changes to style sheets. Some of the colors were changed mainly for testing the layout possibilities and to make the published pages more different from the default EPFC publishes. The EPFC publish includes by default also a page printing possibility; glossary, index and feedback features are optional. A significant amount of the original method content is guidance, which is in PDF or Word format. These remained the same in the EPFC published version.

The main benefits which I noticed with EPF Composer and the new model were customization, good content management, a coherent layout, and the possibilities enabled through SPEM 2.0 and the future tools supporting it. The challenges were familiarizing with the new tool, Finnish language support, and the long-term adaptation of the new model.

Now we have covered the modeling part as a proof of concept of the EPFC and SPEM modeling. The goals of the OpenMethod research depicted in the introduction were achieved: creation of method and process content, customization, and the usage of a CVS repository. When the tool was properly studied and its usage was well-known, modeling was done rapidly. There were also some improvement issues which were discussed with the process author. These are explained in the internal material in more detail.

8.5 User interviews and discussion

After the OpenMethod was modeled with EPFC to the necessary level, user interviews were conducted to obtain comments and suggestions for the new model. The goal of these interviews was to present the new model to some OpenMethod users, to receive further suggestions and user experiences of the new model usage, and to make the final adjustments to the model.

The target group was wide. Even if the number of interviewees was only five, they covered regular OpenMethod users, developers (e.g. method authors) and trainers. Each interviewee received a packet containing the EPFC published version of OpenMethod, usage instructions, links to the EPF material, and several warm-up questions. The questions were about OpenMethod experience, positive and negative comments about the published version, and the EPF Composer tool. There were also questions about the usage of MS Project.

Each of the actual interviews lasted for approximately one hour. At the beginning, there was a brief introduction to the topic, EPF Composer, and the published version. Not all the interviewees had had sufficient time to familiarize themselves with the materials packet. In these cases, some extra time was spent on this in the introduction. The discussion then

continued and focused on the most important concerns depending on the interviewees' interest and expertise. The questions included the following:

- How much do you use OpenMethod and which parts of it?
- What features are better in the new model?
- What are the challenges in the new model?
- How can OpenMethod be improved?
- What are the benefits and drawbacks of using the EPF Composer?
- Do you have any other ideas or suggestions?

The answers were discussed with the interviewees and the main observations were collected. The interviews produced good material and new ideas since each of the interviewees had a different point of view, although additional or longer interviews could naturally have yielded more information. The interviews were scheduled ad-hoc, which led to some cancellations.

Let us now introduce the benefits and challenges observed. The observations shown are the topics that came up most frequently and were considered most relevant. As the questionnaire could be considered to be a pre-study to the modeling, the interviews reflect the changes in the models.

According to the interviewees, the main benefits were the following:

- The layout of the new model corresponds to the old; adaptation for old users to use the new version should be rather straightforward
- The layout is more consistent and clearer
- Rational Software Modeler is also built on Eclipse, which eases the EPFC use
- Support for MS Project will add value to the model
- Good support for centralized knowledge management and version control
- Potential for development, but requires commitment
- Publishing allows customization for different situations (projects and customers)
- Links to other method content make it easier to find material
- EPFC handles the changes to content and automatically keeps the links up-to-date
- Ability to extend OpenMethod with other methodologies

The main challenges were the following:

- The content must be well categorized and easily found; the ‘hyperlink jungle’ of RUP should be avoided
- Old users are used to the current views; a new mindset with links must be adopted
- The multiple features of EPFC can make it complicated
- Possible problems which are not observed in a smaller model may arise later
- An exact and detailed process model is hard to follow in software development
- Making the users utilize EPFC and the published method content in their actual work
- Making users actively update the content and keep it up-to-date
- Confirming that the published version contains all the desired content

The questionnaire, observations made during the study and the interviews all yielded similar results. The new model brings possibilities which overcome the challenges. The main challenge is not about how good the model is in theory or how versatile the process authoring tool is, but rather how people will use them in practice. If the users do not know that the methodology exists or do not believe in it, they will naturally be reluctant to use it. Similar conclusions are also made by other authors [Wes05, Ayd07, Mir07], and Bajec [Baj07] summarizes the reasons for low real practical use of methodologies to *inflexibility* of methodologies and their *social inappropriateness*. It therefore seems to be a common challenge to have good implementation of a new process. Though the tool itself is not difficult to use, its implementation requires a good overall educational strategy. The support for the methodology must be provided through its whole lifetime and user feedback and experience should be used to continuously improve it. This also acts as a signal to the users that their contribution is valued and results in action.

The method engineer’s view is not the only important view. The end-user’s view should be also considered. A regular user might not be interested in SPEM, but rather wants an easy user interface and a familiar layout. Method and process content should be easily found and used. These arguments must also be considered by the method engineer. The organizations may also require the published versions to be visually stylish. This should be more carefully considered in the EPF Composer tool.

9. Conclusions and Summary

This thesis began with a general introduction to processes, models and the differences between the traditional and agile approaches. The idea behind software process improvement was then discussed and an overview was provided of some of the best known methodologies. All these responded to the general questions of the introduction. After this background knowledge, the focus was shifted to the Eclipse Process Framework Composer tool, the meta-model SPEM 2.0, and their usage and capabilities.

The main research question is how Eclipse Process Framework and SPEM 2.0 will improve process modeling and model usage. This was discussed in the previous chapter, while in chapter seven the discussion was on a more general level. To summarize the results, we have the following improvements: good method content management with version control support, the standardized meta-model allows combination with other methodologies and tools, easy customization for specific needs, and enactment support provides the use of project tools such as MS Project and thus connects processes to real-life project use. The modeling process itself will also improve the original process when conducted in an appropriate manner. In this way, focusing on collection and real use of method content and process data will be much more beneficial for an organization than redefining the processes. As to the traffic analogy, the successful adaptation of the new model finally results in more efficient, cheaper and environmentally friendly traffic.

There are several challenges with the EPF and SPEM. Modeling takes time and requires commitment to use all the potential of the tool and the meta-model. The users should find it easier to work with the new model and tools than without them. The support for Finnish language requires a small amount of extra work. There might also be unexpected problems which cannot be evaluated with a limited model and usage. The productivity vs. time curve (Figure 3.3.2) will now be reaching a rather low point before beginning its upward climb.

The Eclipse platform is widely used and the EPF project will most likely continue to be successful as it is also tightly connected to IBM and the Method Composer with the Rational Unified Process. SPEM 2.0 has also been noticed, as discussed in section 6.3. Both EPF and SPEM 2.0 have already been used by several vendors. Tools such as Wilos have a direct impact on the practical usage of EPFC process models in projects.

The research was successful and its goals were accomplished well. There are several areas of research that can be followed up on after this thesis. Future work could include the studies of handling the complete OpenMethod with the EPFC tool, and the users' real long-term practical interest in and usage of the method content and the process models.

Other areas that need to be further explored are the combinations of different method and process content (e.g. OpenMethod and Scrum), the more versatile customizations in the layout of the publications, and the use of maturity models (CMMI and SPICE) with the modeling. The possibility of project wizards, which could select the appropriate roles (or persons), methods, etc. for specific projects, is also an interesting topic for a study.

It is important to follow the evolvement of the EPF community and the usage of EPF and SPEM 2.0 in other projects. The discussion between traditional and agile methodologies will continue and the results depicted with real projects are important.

The development around EPF and SPEM 2.0 will be active. This is pioneer work with the advantage of being at the top, but also having to challenge the risks of the unknown.

References

Books

- [Bec03] Becker, J., Kugeler, M., Rosemann M. (2003), "*Process Management: A Guide for the Design of Business Processes*", Springer-Verlag Berlin and Heidelberg GmbH & Co.
- [Ele98] El Eman, K., Drouin, J., Melo, W. (1998), "*SPICE: The Theory and Practice of Software Process Improvement and Capability Determination*", IEEE Computer Society, Los Alamitos, California.
- [Hai00] Haikala, I., Märijärvi, J. (2000), "*Ohjelmistotuotanto*", 7. painos, Talentum Media Oy.
- [Kos99] Koskinen, M. (1999), "*A Metamodelling Approach to Process Concept Customisation and Enactability in MetaCASE*", Jyväskylä University Printing House, Jyväskylä.
- [Kni07] Kniberg, H. (2007), "*Scrum and XP from the Trenches*", InfoQ Enterprise Software Development Series.
- [Lan05] Lankhorst, M. (2005), "*Enterprise Architecture at Work: Modelling, Communication, and Analysis*", Springer.
- [Lar03] Larman, C. (2003), "*Agile and Iterative Development: A Manager's Guide*", Addison-Wesley Professional.
- [Mar03] Martin, R. (2003), "*Agile Software Development: Principles, patterns, and practices*", Prentice Hall, Pearson Education.
- [Mil06] Milovanov, L. (2006), "*Agile Software Development in an Academic Environment*", PhD thesis, TUCS Dissertations, No 81, December 2006.
- [OOSE04] Bruegge B., Dutoit A. (2004), "*Object-Oriented Software Engineering Using UML, Pattern and Java*", International Edition, Pearson Education.
- [Sch05] Schönström, M. (2005), "*A Knowledge Process Perspective on the Improvement of Software Processes*", Doctoral Dissertation, Lund University.
- [Wie05] Wiegers, K. (2005), "*Software Process Improvement Handbook: A Practical Guide*", Process Impact.

Articles and other publications

- [Abr02] Abrahamsson, P., Salo, O., Ronkainen, J., Warsta, J. (2002), "*Agile software development methods. Review and analysis*", VTT Publications, Espoo.

- [Abr03] Abrahamsson, P., Warsta, J., Siponen, M., Ronkainen, J. (2003), "New Directions on Agile Methods: A Comparative Analysis", Proceedings of the International Conference on Software Engineering, May 3-5, 2003, Portland, Oregon, USA.
- [Ayd07] Aydin, M. (2007), "Examining Key Notions for Method Adaptation", IFIP WG8.1 Working Conference on Situational Method Engineering: Fundamentals and Experiences (ME07), Geneva, Switzerland.
- [Bad05] Baduino, R. (2005), "Basic Unified Process: A process for Small and Agile projects", IBM.
- [Baj07] Bajec, M., Vavpotic, D., Furlan, S., Krisper, M. (2007), "Software Process Improvement Based on the Method Engineering Principles", IFIP WG8.1 Working Conference on Situational Method Engineering: Fundamentals and Experiences (ME07), Geneva, Switzerland.
- [Bec07] Becker, J., Knackstedt, R., Pfeiffer, D., Janiesch, C. (2007), "Configurative Method Engineering – On the Applicability of Reference Modeling Mechanisms in Method Engineering", Proceedings of the 13th Americas Conference on Information Systems (AMCIS 2007). Keystone, CO, USA. 2007. S. 1-12.
- [Ben05] Bendraou, R., Gervais, M., Xavier, B. (2005) "UML4SPM: A UML2.0-Based Metamodel for Software Process Modeling", MoDELS 2005, LNCS 3713, pp. 17-38.
- [Bez04] Breton E., Bézivin, J. (2004), "Applying The Basic Principles of Model Engineering to The Field of Process Engineering", CEPIS, UPGRADE, The European Journal for the Informatics Professional V(5):27—33
- [Bez07] Bézivin, J., Barbero M., Jouault F. (2007), "On the Applicability Scope of Model Driven Engineering", 4th International Workshop on Model-based Methodologies for Pervasive and Embedded Software (MOMPES 2007), University of Nantes, France.
- [Boe03] Boehm, B., Turner, R. (2003), "Observations on Balancing Discipline and Agility", Agile Development Conference, Utah.
- [Bre00] Breton E., Bézivin, J. (2000), "An Overview of Industrial Process Meta-Models", 13th International Conference Software & System Engineering and their Applications, Paris, France.
- [Bre02] Breton E., Bézivin, J. (2002), "Weaving Definition and Execution Aspects of Process Meta-Models", Annual Hawaii International Conference on System Sciences (HICSS'02)-Volume 9
- [Bör04] Börjesson A., Mathiassen L. (2004), "Successful Process Implementation", IEEE Software, IEEE Computer Society.

-
- [Cas07] Castro-Herrera, C. (2007), "*Towards a unified Process for Automated Traceability*", Master's Thesis, Center for Requirements Engineering, DePaul University, Chicago, USA.
- [Coc00] Cockburn, A. (2000), "*Selecting a Project's Methodology*", IEEE software July/August 2000, 64-71.
- [Gom05] Gomes, A., Stüeken, J., Saeed, S. (2005), "*A Study of Rational Unified Process*", Blekinge Institute of Technology, Sweden.
- [Gon07] Gonzalez-Perez, C. (2007), "*Supporting Situational Method Engineering with ISO/IEC 24744*", IFIP WG8.1 Working Conference on Situational Method Engineering: Fundamentals and Experiences (ME07), Geneva, Switzerland.
- [Dej06] Dejong, J. (2006), "*Taking the Rational Out of RUP – Two unified processes, OpenUP and EssUP, streamline practices*", SD Times, November 1, 2006.
- [Hen04a] Henderson-Sellers, B., Serour M., McBride T., Gonzalez-Perez C., Dagher L. (2004), "*Process Construction and Customization*", Journal of Universal Computer Science, vol. 10, no. 4 (2004), 326-358.
- [Hen04b] Henderson-Sellers, B., Gonzalez-Perez, C. (2004), "*A comparison of four process metamodels and the creation of a new generic standard*", University of Technology, Sydney, Australia.
- [Hen06a] Henderson-Sellers, B. (2006), "*SPI – A Role for Method Engineering*", University of Technology, Sydney.
- [Hen06b] Henderson-Sellers, B., Cossentino, M., Seidita, V. (2006), "*A metamodelling-based approach for method fragment comparison*", Eleventh International Workshop on Exploring Modeling Methods in Systems Analysis and Design.
- [Jaf05] Jaferian, P., Elahi, G., Shirazi, M., Sadeghian, B. (2005), "*Extending Business Modeling and Requirements Disciplines of RUP for Developing Secure Systems*", EUROMICRO-SEAA'05.
- [Jeu07] Jeusfeld, M. (2007), "*Partial Evaluation in Meta Modeling*", IFIP WG8.1 Working Conference on Situational Method Engineering: Fundamentals and Experiences (ME07), Geneva, Switzerland.
- [Joh04] Johnson, M. (2004), "*A Case Study in Balanced Software Process Development*", Turku Centre for Computer Science, TUCS Technical Report, No 599, March 2004.
- [Jär07] Järvi, A., Hakonen, H., Mäkilä T. (2007), "*Developer Driven Approach to Situational Method Engineering*", IFIP WG8.1 Working Conference on Situational Method Engineering: Fundamentals and Experiences (ME07), Geneva, Switzerland.

- [Jär06] Järvi A., Mäkilä T., Hakonen H. (2006), "*Changing Role of SPI – Opportunities and Challenges of Process Modeling*", Proceedings of the 13th European Conference on Software Process improvement (EuroSPI 2006), Joensuu, Finland.
- [Jär05] Järvi, A., Mäkilä, T. (2005), "*Observations on Modeling Software Processes with SPEM Process Components*", Proceedings of The 9th Symposium on Programming Languages and Software Tools.
- [Kel07] Kelly, S. (2007), "*Domain-Specific Modeling: The Killer App for Method Engineering?*", IFIP WG8.1 Working Conference on Situational Method Engineering: Fundamentals and Experiences (ME07), Geneva, Switzerland.
- [Kos03] Koskela, J. (2003), "*Software configuration management in agile methods*", Espoo 2003, VTT Publications 514, 54 p.
- [Kru01] Kruchet, P. (2001), "*A Process Engineering Metamodel*", Rational Software.
- [Lev05] Levine, L. (2005), "*Reflections on Software Agility and Agile Methods: Challenges, Dilemmas, and the Way Ahead*", SEI, Carnegie Mellon University, Pittsburgh.
- [Mir07] Mirbel, I. (2007), "*Connecting method engineering knowledge: a community based approach*", IFIP WG8.1 Working Conference on Situational Method Engineering: Fundamentals and Experiences (ME07), Geneva, Switzerland.
- [Moo06] Moor, A., Delugach, H. (2006), "*Software Process Validation: Comparing Process and Practice Models*", Proceedings of the Workshop on Exploring Modeling Methods for Systems Analysis and Design (EMMSAD'06).
- [Mäk07] Mäkilä, T., Järvi, A., Milovanov, L. (2007), "*Light-weight Approach for Software Process Modeling – A Case Study*", In New Exploratory Technologies 2007, Oct 2007.
- [Mäk06] Mäkilä, T., Järvi, A. (2006), "*Spemmet – A Tool for Modeling Software Processes with SPEM*", Proceedings of the 9th International Conference on Information Systems Implementation and Modelling, ISIM '06.
- [Ner05] Nerur, S., Mahapatra, R., Mangalaraj, G. (2006), "*Challenges of Migrating to Agile Methodologies*", Communications of the ACM, May 2005/Vol. 48, No 5.
- [Nia06] Niazi, M. (2006), "*Software Process Improvement: A Road to Success*", PROFES 2006, LNCS 3034, pp. 395-401.
- [Ram06] Ramsin, R. (2006), "*The Engineering of an Object-Oriented Software Development Methodology*", Submitted for the degree of Doctor of Philosophy, York, UK.

- [Rei07] Reinhartz-Berger, I., Aharoni, A. (2007), “*Representation of Method Fragments: A Domain Engineering Approach*”, Proceedings of the Workshop on Exploring Modeling Methods for Systems Analysis And Design 2007.
- [Sch07] Schwaber, K. (2007), “*What is Scrum?*”, Scrum Alliance.
- [SPEM2.0] Object Management Group (2007), “*Software & Systems Process Engineering Metamodel Specification, version 2.0*”, Proposed Available Specification ptc/2007-08-07.
<http://www.omg.org/docs/ptc/07-08-07.pdf>
- [SPEM2.0a] Object Management Group (2007), “*Software Process Engineering Metamodel Specification, version 2.0*”, Final Adopted Specification ptc/07-03-03.
<http://www.omg.org/docs/ptc/07-03-03.pdf>
- [SPEM1.1] Object Management Group (2005), “*Software Process Engineering Metamodel Specification, version 1.1*”, format/05-01-06.
<http://www.omg.org/docs/formal/05-01-06.pdf>
- [Wes05] Westerheim, H. (2005), “*The Introduction and Use of a Tailored Unified Process – A Case Study*”, EUROMICRO-SEAA’05.
- [Zhu07] Zhu, L., Staples, M. (2007), “*Situational Method Quality*”, IFIP WG8.1 Working Conference on Situational Method Engineering: Fundamentals and Experiences (ME07), Geneva, Switzerland.

Web pages

- [AGILE07] Manifesto for Agile Software Development
<http://agilemanifesto.org/>
Accessed on October 2nd, 2007.
- [Amb06] Ambler, S. (2006), “*History of the Unified Process*”
<http://www.enterpriseunifiedprocess.com/essays/history.html>
Accessed on October 2nd, 2007.
- [Bar07] Barnett, L. (2007), “*Making Sense of the (Too) Many Agile Processes*”, Agile Journal, 6.5.2007.
<http://www.agilejournal.com/articles/from-the-editor/making-sense-of-the-%28too%29-many-agile-processes/>
Accessed on October 2nd, 2007.
- [Bro03] Brown, P., Szeffler, M. (2003), “*BPEL for Programmers and Architects*”,
<http://www.bptrends.com/publicationfiles/BPEL4ProgArchies.pdf>
Accessed on October 2nd, 2007.




- [Dict07] Dictionary.com
<http://www.dictionary.com>
Accessed on October 2nd, 2007.
- [ECL07] Eclipse – an open development platform homepage
www.eclipse.org
Accessed on October 2nd, 2007.
- [EPF07] Eclipse Process Framework Project homepage
www.eclipse.org/epf/
Accessed on October 2nd, 2007.
- [EPFRel07] Eclipse Process Framework Project, “*EPF 1.2 Release Review*”.
http://www.eclipse.org/projects/slides/EPF_1-2_Release_Review.pdf
Accessed on October 2nd, 2007.
- [EclPC] Eclipse Plug-in Central
http://www.eclipseplugincentral.com/Web_Links-index-req-viewcatlink-cid-878.html
Accessed on October 2nd, 2007.
- [EPFW07] EPF Wiki
<http://www.epfwiki.net/>
Accessed on October 2nd, 2007.
- [Gra02] Graham, D. (2002), “*The Forgotten Phase*”, Development Tools, Dr. Dobb’s Portal.
<http://www.ddj.com/development-tools/184414873>
Accessed on October 2nd, 2007.
- [Hau05] Haumer, P. (2005), “*IBM Rational Method Composer: Part1: Key Concepts*”, developerWorks, IBM.
<http://www-128.ibm.com/developerworks/rational/library/dec05/haumer>
Accessed on October 2nd, 2007.
- [Hau07a] Haumer, P. (2007), “*Eclipse Process Framework Composer, Part 1: Key Concepts*”, Eclipse Process Framework homepage.
<http://www.eclipse.org/epf/general/EPFComposerOverviewPart1.pdf>
Accessed on October 2nd, 2007.
- [Hau07b] Haumer, P. (2007), “*Eclipse Process Framework Composer, Part 2: Authoring method content and processes*”, Eclipse Process Framework homepage.
<http://www.eclipse.org/epf/general/EPFComposerOverviewPart2.pdf>
Accessed on October 2nd, 2007.
- [Kro07a] Kroll, P. (2007), “*Who will benefit from the Eclipse Process Framework*”.
<http://www.eclipse.org/proposals/beacon/Who%20will%20benefit%20from%20Eclipse%20Process%20Framework.pdf>
Accessed on October 2nd, 2007.

- [Kro07b] Kroll, P. (2006), “*Eclipse Process Framework – Open Source Initiative*”.
<http://www.eclipse.org/epf/community/Eclipse%20Process%20Framework-%20An%20Open%20Source%20Process%20Initiative%20by%20Per%20Kroll.ppt>
Accessed on October 2nd, 2007.
- [Mal05] Malloy, E. (2005), “*Interactive Internet – Welcome Wiki Technology*”,
Blizzard Internet Marketing inc. Newsletter.
<http://newsletter.blizzardinternet.com/featured-article-3/2005/08/15/>
Accessed on October 2nd, 2007.
- [Ose07] Osellus Incorporated, (2007)
<http://www.osellus.com/>
Accessed on October 2nd, 2007.
- [RUP701] Rational Unified Process 7.0.1 -materials on IBM web-page
<http://www-306.ibm.com/software/awdtools/rup/>
Accessed on October 2nd, 2007.
- [Kro03] Kroll, P. (2003), “*The RUP: An industry-wide platform for best practices*”,
The Rational Edge,
<http://www.ibm.com/developerworks/rational/library/873.html>
Accessed on October 2nd, 2007.
- [San07] Serqi Santos, S. (2007), “*Comparing the Rational Unified Process (RUP) and Microsoft Solutions Framework (MSF)*”.
<http://www.ibm.com/developerworks/rational/library/apr07/santos/index.html>
Accessed on October 2nd, 2007.
- [SD07] Commercial advertisements and Webpage information.
<http://www.sysopendigia.com>
Accessed on October 2nd, 2007.
- [SEI07] Software Engineering Institute, Carnegie Mellon University
<http://www.sei.cmu.edu/ideal/>
Accessed on October 2nd, 2007.
- [SPICE] Software Process Improvement and Capability dEtermination
<http://www.sqi.gu.edu.au/spice/>
Accessed on October 2nd, 2007.
- [Wik07] Wikipedia, “*Wiki*”,
<http://en.wikipedia.org/wiki/Wiki>
Accessed on October 2nd, 2007.
- [Wil07] Wilos Is a cLever Process Orchestration Software (Wilos)
<http://www.wilos-project.org/drupal/>
Accessed on October 2nd, 2007.







- [YAWL07] Wawl, Yet Another Workflow Language
<http://www.yawl-system.com>
Accessed on October 2nd, 2007.
- [You06a] Eggtronic Team Process Video (2006)
<http://www.youtube.com/watch?v=wML90z2hmZ0>
Accessed on October 2nd, 2007.
- Other**
- [Abr05] Abrahamsson, P. (2005), “*Agile Software Development: Introduction, Current Status & Future*”, Introduction to Software Engineering 2005 course material, Jyväskylä.
- [CMMI07] *Capability Maturity Model Integration Version 1.2 Overview*, Software Engineering Institute, Carnegie Mellon University.
- [Elv06] Elvesæter, B. (2006), “*Method engineering for software development and integration*”, Lecture #7, SINTEF ICT.
- [Fir06] Firesmith, D. (2006), “*Method Engineering using OPFRO*”, EuroSEPG 2006.
- [Hai07] Haikala, I. (2007), “*Ohjelmistotuotannon menetelmät, kevät 2007*”, course material,
<http://www.cs.tut.fi/~otm/>
Accessed on October 2nd, 2007.
- [Hau06] Haumer, P. (2006), “*Second Revised SPEM 2.0 Submission*”, OMG Meeting, St. Louis.
- [Hau07c] E-mail discussion with Haumer, P., September, 2007.
- [IT07] ‘IT-viikko’ magazine, (2007), “*Ketteryys kiinnostaa*”, April 12th, 2007.
- [Kil07] Kilpivuori, T. (2007), “*Impact of technology and technical methods on informations systems, and their parties and to their processes*”, Master’s Thesis, University of Turku.
- [Lju04] Ljungqvist, P. (2004), “*Tailored CMMI Software Process Assessment*”, Master's Thesis, Åbo Akademi University.
- [Man07] Mantell, K. (2007), “*CMMI and IBM Rational Unified Process: A practical route to greater development maturity*”, “CMMI Made Practical”, London, 19-20th March, 2007

- [Mol06] Molesini, A. (2006), "*Agent Oriented Software Engineering*", Alma Mater Studiorum, Università di Bologna.
- [Oja03] Ojala, A. (2003), "Hallittu järjestelmäkehitys, laadukas lopputulos?", OUGF Spring Seminar.
- [OpenUP] OpenUP 1.0, publish created on 1.8.2007.
- [OpenUPa] OpenUP 1.0, publish created on 1.8.2007, uma_vs_rup concept.
- [OT07] Järvi, A., Alhoniemi, E. (2007), "*TKO_5353 Ohjelmistotuotanto*", course material, University of Turku.
- [PRO00] Probasco, L. (2000), "*The Ten Essentials of RUP: The Essence of an Effective Development Process*", Rational Software White paper.
- [RMC06] IBM Rational Method Composer, Rational Software White paper.
- [Rub05] RubyTurtle (2005), "*Microsoft Methodologies: An overview*", A RubyTurtle Whitepaper.
- [Sal03] Salonen, T. (2003), "*Application of Extreme Programming in Software Development*", Final Theses, Turku Polytechnic.
- [SSE07] Järvi, A., Mäkilä, T. (2007), "*Seminar on the Software Development Processes 2007*", University of Turku.
<http://users.utu.fi/tusuma/SPI/>
Accessed on October 2nd, 2007.
- [Sue04] Suen, V. (2004), "*SPEM Overview*", Osellus Webinar series.
- [Ter06] Tervonen, I. (2006), "Ohjelmistotekniikka – Luento 2", University of Oulu.

Appendix A: SPEM icons with explanations

Name	Description	Icon
Activity	<p>An Activity represents something that one or more roles perform. It is a grouping of nested process elements (e.g. Breakdown Element) such as other Activity instances, Task Uses, Role Uses, Milestones, etc. Phases and Iterations are special Activities which have predefined attribute values.</p> <p>Activities are one of the fundamental concepts for defining processes. Activities are related to timelines and instances of specific Breakdown Element instances can define different relationships and textual documentation properties for occurrences in different activities.</p> <p>For example, a SPEM 2.0 user creates two instances of a Role Uses that represent a Role Definition called “System Designer” for two different activities. In both of the activities the same Role Definition “System Designer” could be modeled with different relationships such as different responsibilities for Work Products to represent the fact that the System Designer has to focus on different responsibilities in different activities (e.g. he might be responsible for different work product in an early phase of a project than in a later phase; with phases modeled as Activities).</p>	
Breakdown Element	<p>A Breakdown Element is any element that is part of the process structure.</p>	-
Delivery Process	<p>A Delivery Process is a special Process describing a complete and integrated approach for performing a specific project type. It describes a complete project lifecycle end-to-end that has been detailed by sequencing Method Content in breakdown structures.</p> <p>A Delivery Process is defined on the basis of experience with past projects or engagements, and the best practice use of a development or delivery approach. It can be used as a template or reference for planning and running projects with similar characteristics as defined for the process.</p> <p>For example, a process engineer can define alternative Delivery Processes for software development projects that differ in the scale of the timetable, engagement and staffing necessary, the type of the software application to be developed, the development methods and technologies to be used, etc.</p>	
Discipline	<p>A discipline is a collection of related tasks that define a major ‘area of concern’.</p> <p>Categorization of work based upon similarity of concerns and cooperation of work effort is an effective way to organize content, which makes comprehension easier. It works as an aid to understand the project from traditional waterfall perspective as it is common to perform tasks concurrently across several disciplines. For example, test tasks can be performed in close coordination with implementation and even during analysis and design tasks.</p> <p>Every discipline also defines ways of working. Capability patterns are used to show how the tasks categorized by the discipline work together in the most generic way. These are often used for educating and teaching practitioners and it helps users to understand the whole process by breaking it into smaller areas of concern.</p>	 <p>(Icon used with EPF)</p>

Appendix A: SPEM icons with explanations

Name	Description	Icon
Guidance	<p>Guidance is a Describable Element that provides explanations and additional information related to other Describable Elements.</p> <p>The particular Guidance should be classified with Kinds that indicates a specific type of guidance having a specific structure and type of content. Examples for Kinds for Guidance are Guidelines, Templates, Checklists, Tool Mentors, Estimates, Supporting Materials, Reports, Concepts, etc.</p>	
Milestone	<p>Development processes consist of sequences and milestones. Milestone describes a significant event in development – it is the point where an iteration or phase formally ends. This provides a check-point for whether the process is ready to move forward.</p>	
Phase	<p>Phase is a significant period within a development process. During phase, a well-defined set of objectives is met and the phase ends with a major management checkpoint, milestone.</p>	
Process	<p>Processes use content elements to relate them into partially ordered sequences that are customized to specific projects. A process focuses on the lifecycle and the sequencing of work in breakdown structures.</p>	
Process Pattern (Capability pattern)	<p>A process pattern is a reusable building block for creating new development processes - Delivery Processes or larger Process Patterns. It describes a cluster of Activities that provides a consistent approach to common problems. This cluster has process knowledge for example of a discipline.</p> <p>Process pattern supports copy and modify operations, which allows the process engineer to customize the pattern's content according to specific needs. Patterns can also be used through the Activity Use mechanism. In this way the activities can be factored out into patterns and used over and over in a process. When the pattern is updated, all changes will automatically be reflected in processes that applied that pattern.</p> <p>Process patterns are suitable for Agile Development where the project does not exactly follow a process, but rather works flexible based on process fragments of best practices. Process patterns suit also for describing best practices on performing work for a Discipline or for a specific development technique or phase.</p>	
Role Definition	<p>A Role Definition is a Method Content Element that defines a set of related skills, competencies, behavior and responsibilities of an individual or a set of individuals. Roles are not individuals nor are necessarily equivalent to job titles. Roles are used by Task Definitions to define who performs them as well as define a set of Work Product Definitions they are responsible for.</p> <p>The mapping from individual to Role, performed by the project manager when planning and staffing for a project, allows different individuals to act as several different roles, and for a role to be played by several individuals. Examples of roles include Developer, Stakeholder and Tester.</p>	

Appendix A: SPEM icons with explanations




Name	Description	Icon
Task Definition	<p>A Task Definition is a Method Content Element and a Work Definition that defines an assignable unit of work being performed by Roles Definition instances. The granularity of a Task Definition is generally a few hours to a few days. A Task is associated to input (mandatory and optional) and output Work Products and it usually affects one or only a small number of Work Products.</p> <p>A Task provides complete step-by-step explanations of doing all the work required to achieve this goal. This description is complete, independent of when in a process lifecycle the work would actually be done. Therefore, it does not describe when you do what work at what point of time, but describes all the work that gets done throughout the development lifecycle that contributes to the achievement of the Tasks' goal.</p> <p>When a Task Definition is used in a process, a reference object defined as Task Descriptor provides information, what will actually be performed at that specific point of time. All of the Task Definition's default associations and parameters can be overridden in this actual process definition.</p> <p>Tasks Definition (as well as Activities) can further be used for planning and tracking progress; therefore, if they are defined too fine-grained, they will be neglected, and if they are too large, progress would have to be expressed in terms of a Task Definition's parts (e.g. Steps, which is not recommended).</p>	
Tool Definition	<p>A Tool Definition is a special Method Content Element that can be used to specify a tool's participation in a Task Definition. It is also used as a container for tool mentors.</p> <p>A Tool Definition describes the capabilities of a CASE tool, general purpose tool, or any other automation unit that supports the associated instances of Role Definitions in performing the work defined by a Task Definition.</p>	
Work Product Definition	<p>Work Product Definition is Method Content Element that is used, modified, or produced by Task Definitions.</p> <p>They may serve as a basis for defining reusable assets. Roles use Work Products to perform Tasks and produce Work Products in the course of performing Tasks. Work Products are the responsibility of Role Definitions, making responsibility easy to identify and understand, and promoting the idea that every piece of information produced in the method requires the appropriate set of skills. Even though one Role Definition would be responsible of a specific type of Work Product, other roles can still use the Work Product for their work, and perhaps even update them if the Role Definition instance has been given permission to do so. ^{1,21}</p>	

Table A.1 – some essential SPEM 2.0 icons with explanations (Annex A: SPEM icons)

Appendix B: Citations

ALL CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL CONTAINED IN THIS PUBLICATION IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.