UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

A META-PROCESS FOR DEFINING ADAPTABLE SOFTWARE PROCESSES

TESIS PARA OPTAR AL GRADO DE
DOCTOR EN CIENCIAS MENCIÓN COMPUTACIÓN

JULIO ARIEL HURTADO ALEGRÍA

PROFESORA GUIA:
MARÍA CECILIA BASTARRICA PIÑEYRO

MIEMBROS DE LA COMISIÓN:
ALEXANDRE BERGEL
SERGIO OCHOA DELORENZI
ROMMAIN ROBBES
JOLITA RALYTÉ

Este trabajo ha sido parcialmente financiado por NIC Chile

SANTIAGO DE CHILE
AGOSTO, 2012

Resumen

Lograr proyectos de software productivos y con calidad dentro de una industria dinámica y competitiva, requiere definir modelos de proceso correctos y adecuados al contexto. Así, el mejor proceso debe estar correctamente definido y debe ser adecuado a las particularidades del proyecto en el que será usado. Típicamente, un ingeniero de procesos define un proceso específico para cada proyecto en forma ad-hoc, lo cual resulta costoso, irrepetible y propenso al error. Por otro lado, la especificación de procesos demanda un esfuerzo enorme y una vez éstos son especificados, son pocos los enfoques y aún menos las herramientas, que asistan al ingeniero de procesos a analizar la calidad de sus modelos de proceso. En los últimos cinco años hemos asesorado empresas de software en la especificación de sus procesos de software. Como parte de este trabajo una serie de problemas fueron identificados, éstos indican la presencia potencial de incorrectas concepciones y especificaciones, así como inadecuadas adaptaciones en el modelo del proceso. Para prevenir errores en la adaptación de procesos, esta tesis propone Casper, un metaproceso para definir modelos de proceso adaptables al contexto. Casper usa un enfoque basado en modelos para adaptar el proceso de desarrollo generando procesos específicos a proyectos a partir del proceso organizacional y el contexto específico del proyecto. El enfoque es sistemático, repetible y no depende de un usuario experto en ingeniería de procesos. Para asistir al ingeniero de procesos en el análisis de problemas concepuales y de especificación, en esta tesis se desarrolló Avispa. Avispa es una herramienta que gráficamente presenta diferente patrones de error de un modelo de proceso de software resaltando los errores potenciales a través de indicadores comprensibles e intuitivos. Los enfoques de Casper y Avispa han sido validados aplicándolos en la definición y análisis de algunos modelos de proceso de la industria de software Chilena y algunos procesos públicos disponibles desde la comunidad de Eclipse Process Framework. Estos enfoques muestran ampliamente la utilidad práctica del enfoque dirigido por modelos para lograr modelos de proceso de alta calidad.

Abstract

Reaching quality and productivity in software projects within a competitive and dynamic software industry requires defining context suitable and correct software process models. Thus, the best process depends on the particularities of each project and on its correctness. Typically a process engineer defines a specific process for each project in an ad-hoc fashion, which is expensive, unrepeatable and error prone. On the other hand, software process specification demands an enormous effort, but once specified there are few approaches and even fewer tools that aid the process engineer to analyze the quality of the process. For the last five years we have assisted software companies in specifying their software processes. As part of this work a series of problems that indicate the potential presence of misconceptions, misspecifications or unsuitability has been identified. To prevent suitability errors this thesis proposes to define context suitable process models using CASPER, a meta-process for defining adaptable software process models. CASPER uses a model-based approach to software process tailoring that generates project specific processes based on the organizational process and the project context. The approach is systematic, repeatable and it does not depend on the expert people using it. To assist process engineers to analyze the misconceptions, misspecifications of their processes, AVISPA has been developed. AVISPA is a tool that graphically renders different error patterns of a software process model highlighting potential errors as intuitive and comprehensible indicators. CASPER and AVISPA approaches have been validated applying them for defining and analyzing some software process models of the Chilean software industry and some public processes obtained from the Eclipse Process Framework community. These approaches show largely the practical usefulness of a model driven approach to achieve process models with high quality.

I would like to dedicate this thesis to my children, Arturo and Katherinne, my parents Carmen Ruth and Julio Hernán and my wife Lida Piedad. Arturo, for your unlimited understanding. Katherine, for her tender love. Carmen Ruth and Julio Hernán for their insightful advice. Lida Piedad, for her unconditional love. They were largely my great accomplices in this achievement.

Acknowledgements

And I would like to specially acknowledge to my advisor Cecilia Bastarrica for her great human and professional support that I received: patient, right words, relevant questions, humanity, and unconditional dedication and collaboration. Importantly, I would like to thank Alexandre Bergel for his advice and support in this research. Also, I would like to acknowledge my research team, especially to Sergio Ochoa, Daniel Perovich, Pedro Rossel and Alcides Quispe for their valuable contributions.

Contents

List of Figures

Chapter 1

Introduction

Counting on a well defined software process model is an important factor for achieving software quality and process productivity. Therefore, many companies have prioritized initiatives to specify and improve their software process. As a result, organizations usually need many different kinds of processes of different magnitudes and for different purposes. The organization would also generally need more than one development process. This set of the processes implicitly compose a software process family (Rombach, 2005) and it in the best cases include a reference process, a library of reusable assets, and tailoring and enacting mechanisms.

Different software development life cycles suggest specific activities to be carried out in a particular order, from traditional models such as the Waterfall, to more modern ones such as RUP (Kruchten, 2003), Scrum (Schwaber, 1995) or XP (Beck & Andres, 2004). Also, if a company aims to certify or evaluate its software development process, it should be strictly defined as prescribed to comply with some models and standards such as CMMI (SEI, 2006) and ISO/IEC 12207 (ISO/IEC, 2008). This organizational process definition normally requires an enormous effort and it still needs to be adapted to satisfy the specific characteristics of different project situations (Mirbel & Ralyé, 2005). Also, there are still no standard mechanisms for determining the quality of the specified process, and thus of software process definition could be risky to the organization.

This thesis has been motivated by the observations collected for the past five years on software companies in Chile to define their development processes in an effort to improve national industry standards[1]. As part of this practical experience, the process adaptation problem arose as an industrial need. On the other hand, some typical recurrent errors in software process modeling showed that the quality of the process models is also problematic. But both, adaptation and analysis are not easy tasks because there is an

[1]Tutelkán: Achieving High Quality in National Software Industry by Applying Reference Processes (`www.tutelkan.org`).

enormous amount of process elements involved, there are multiple views, and informal notations that may sometimes introduce ambiguity are usually used. This thesis deals with the reuse of reference processes, the tailoring of software process models to specific contexts and the verification of software process models to achieve quality in their definition. Consequently, this thesis assesses how a model driven approach may result useful and practical to perform a meta-process (Kontio, 1998) to mainly implement:

- An MDE approach to software process tailoring.
- A visual recovery approach to analyze software process models.

This chapter introduces this thesis including the problem, the challenges, the hypothesis, the goals, the used research methodology and the document content outline.

1.1 Problems and Challenges

The more oriented towards their processes organizations intend to be, the more conceptual and technological support they require. There are several challenges when an organization is process oriented. Among them, there is a huge variety of models for different purposes including quality, reference process, assessment and improvement, a variety of software process modeling languages, diverse implementations for software process execution, and a complex relationship among all the above. Managing these relationships among models with different goals, languages and technologies requires a deep knowledge about software processes in order to put them into practice and being able to reach all the advantages they may provide.

One of the challenges in the area of process engineering is to research for mechanisms that allow taking advantage of the variety of supporting technologies for process definition, tailoring, execution, verification, validation and evolution including process modeling languages (PMLs) (Zamli, 2004). Process model solutions should be independent of changing aspects such as a particular project or the execution platform, and in this way their operation, evolution, reuse and adaptation would be easier (Bretón & Bézivin, 2001). A key strategy for reaching a competitive software industry is to apply the concepts and technologies of process engineering in the most context-suitable way possible. However, this strategy requires a big and disciplined effort for the organizations for defining, tailoring and analyzing software process models.

1.1.1 General Questions

Process models are major knowledge assets for mature organizations and communities. When a process model is defined and applied many errors could be introduced. Some errors correspond to underspecifications or misspecifications, however others are related to conceptual and design problems when a software process model is defined, tailored, interpreted or used. Defining a software process model with quality requires the use of techniques for building, validating and verifying software process models. As in software, two complementary approaches are the detection and prevention of defects. Each practice for achieving quality requires effort. The challenge is achieve a good trade off between quality and practicality. Following this two approaches, this thesis has two major components:

- *Prevention of defects* introduced by human errors during the tailoring step. Because tailoring must be executed in the context of a software project, the time, knowledge and effort to achieve a suitable software process model is limited. Figure 1.1 shows the tailoring step within a basic meta-process. The relevant question is : how the tailoring step can be conduced in an effective and expert independent way?

- *Detection of defects* introduced by human errors in different stages, in Figure 1.1 points 1,2, 3 and 4. A way for detecting errors in process models is to analyze their correctness and practical issues. Therefore, analyzing software process models is in general a transversal activity for achieving quality process models. Particularly, in situational method engineering and tailoring approaches, process analysis has a great value for assessing general and derived process models. However validating and verifying software process models result either impractical or limited. The relevant question is : how the software process model analysis can be conduced in a effective and practical way?

1.1.2 Software Process Tailoring

Process tailoring is the task of defining project-specific processes by adapting the activities and their related process elements from a base process model to develop high-quality software efficiently (Washizaki, 2006). Within software organizations, the standard software process is usually applied directly (without any tailoring at all) o tailored in an inadequate way as a consequence of the difficulty required for it. As a consequence, this situation increases the costs and risks of development projects (Pedreira *et al.*, 2007). On the other hand, if the software process is not rigorously applied, the organization could

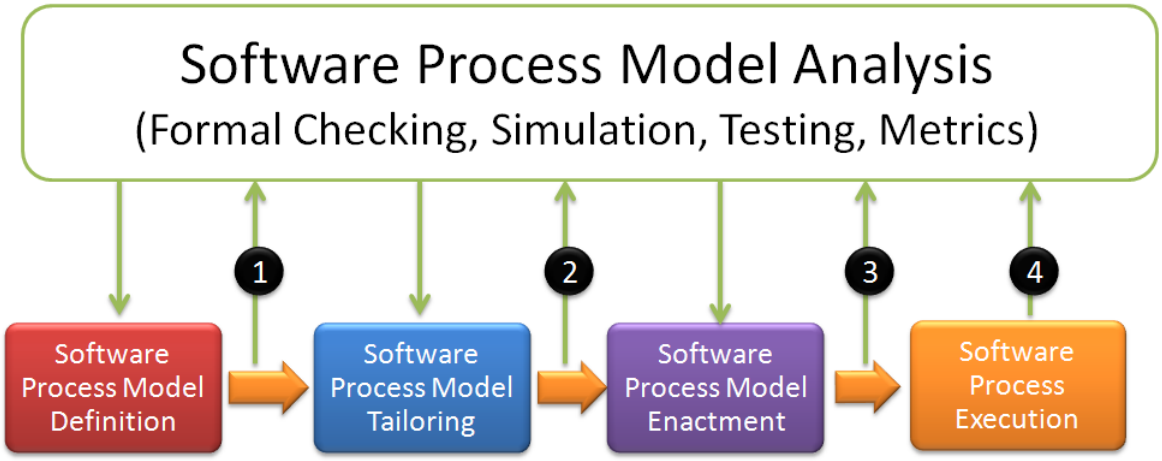

Figure 1.1: Basic Software Process Engineering Meta-process

be losing most of the effort invested for defining the software process as a competitive strategy. For that reason, tailoring is a trade off required to have both, a repeatable software process and a software process suitable for the particular needs of each project.

Process tailoring is a difficult task because it involves intensive knowledge in software process engineering or in situational method engineering (Mirbel & Ralyé, 2005; Rolland, 2009). Additionally, it normally occurs in a reactive way, and therefore it is time consuming (Ocampo *et al.*, 2005). According to Bai *et al.* (2010), process engineering stakeholders and software engineers (project stakeholders) are usually not clearly distinguished. The process tailoring is generally performed by the project manager, i.e., a process user and not a process designer. Thus, a process modeling approach should provide ways to cost-efficiently adapt a general (or combined) process model into a project-specific process model (Armbrust *et al.*, 2009). The problem is how a process adapter tailors a process model just defining a specific situation, because the main concern for software engineers (Bai *et al.*, 2010) is to build software and not to built software processes.

The Software Process Line (SPrL) approach has emerged to resolve the reactive approach in software process tailoring. The SPrL approach facilitates the planned reuse of process assets, while classic tailoring re-actively integrates unanticipated variability in the process model (Armbrust *et al.*, 2009). A SPrL seems to be an ideal way to define, tailor and evolve a set of related processes as it can be evidenciated by the work on process variability (Simidchieva *et al.*, 2007; Washizaki, 2006). However, to build software process lines requires methodological approaches and technological support, particularly to perform the process engineering considering situational information both, as part of the planned variability and as a tailoring strategy itself.

1.1.3 Software Process Analysis

Software process models are sophisticated and huge specifications and building them demands an enormous effort. However, once a process model is specified it is not evident how to determine if it is well defined, or if it can be improved in any sense before it is enacted (Osterweil, 1987). The software process meta model SPEM 2.0 (OMG, 2008) proposes wellformedness rules for software process definition but their scope is limited. For example, SPEM 2.0 does not determine if for a given process some of the roles are overloaded, if there are work products that are potential bottlenecks, or if there is no clear termination condition for a task cycle. Furthermore, these issues do not always constitute errors, but they are indicators that something may not be completely right (Jacobs & Marlin, 1996). Even though there are some metrics defined for measuring software processes (Cánfora *et al.*, 2005), they are general metrics over the software process model and they provide a limited intuition about what may be wrong, where errors are, and how to find opportunities for improvement when there is no apparent error. So, a question arises: how can we present a rich software process on mere flat diagrams? Software process models should represent a complex, dynamic and multidimensional world. Moreover, available tools are usually intended only for software process visual specification, and not for analysis, let alone visual evaluation.

For the last five years several Chilean software companies have been observed while they specified their software processes and some recurrent problems have been identified indicating the potential presence of misconceptions or misspecifications. However, it is a recurrent situation that none of them are easily identified, let alone localized, because of the huge amount of process elements involved, multiple views required, and informal notations that may sometimes introduce ambiguity.

1.2 Thesis hypotheses

The hypothesis has been formulated around the fundamental problem of achieving software process models of quality. However, this hypothesis has been focused to achieve suitable and correct software process models, as Figure 1.2 shows.

1. *Hypothesis:* a software process can be built and tailored as a software process line in a systematic way using an MDE production strategy and situational information formally specified, introducing the advantages of the planned reuse of the software process lines approach and the automatization of the MDE approach.

- *Derived Hypothesis:* the software process derivation from a software process line using the MDE approach is simpler and faster than a non planned approach. Thus, a major cost-effectiveness could be achieved and the software process could be improved in practice.

2. *Hypothesis:* a software process model can be analyzed using a visual approach. Thus, an intuitive mechanism could simplify the software process model validation by identifying syntactical, semantical and pragmatical problems

   - *Derived Hypothesis:* a software process model can be analyzed using a visual approach without requiring a great experience. Software process model verification and validation could be applied in practice to identify missconceptions, underspecifications and practical issues.

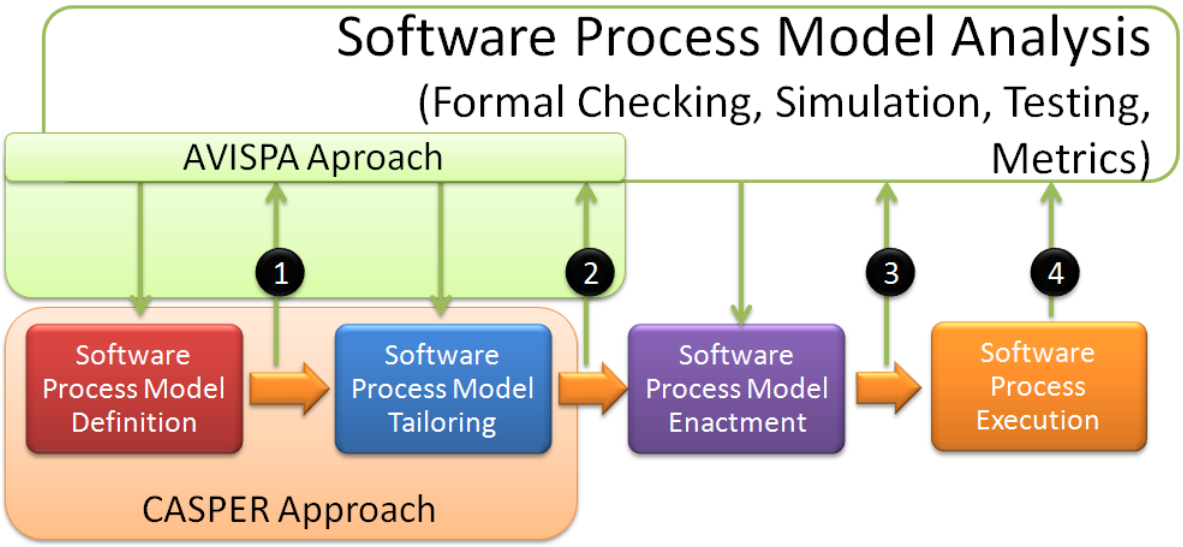

Figure 1.2: Hypothesis in the Basic Software Process Engineering Meta-process

Although both hypotheses address to the same problem line, it is important to considerer the independendence presented in this work. To validate each hypothesis requires to define a unique technological context. The tailoring hypothesis requires counting with technological maturity of the software modeling tools; the ideas behind of this hypothesis are to improve the current tools for modeling software process models including new concepts (i.e. contexts, process features and tailoring rules). In this thesis, the process models in tailoring research are based on eSPEM a simplified version of SPEM and for the complementary models, specific metamodels have been defined. On the other hand, the analysis hypothesis requires work on available industrial process models; this is achieved

defining mechanisms to analyze process models using standard information. Hence, in this thesis, the process models in analysis research are based on UMA the metamodel behind Eclipse Process Framework, a SPEM based process modeling tool used by the software industry and the process community. Actual and available UMA software process models are used to validate this approach. On the other hand, in the software process model tailoring hypothesis, the technological maturity is lower than that required by the process analysis hypothesis. Therefore, a basic infrastructure was developed for validating the tailoring proposal itself and the cases introduced(normally specified in textual way) are modeled with this infrastructure using its process sources modeled in different metamodels.

1.3 Thesis goals

1.3.1 Main goal

The main goal is to build a meta-process based enabling context suitable and correct software process models. It includes a general and abstract process process line from which derived processes can be created through a production strategy based on MDE and verified in a visual way.

1.3.2 Specific goals

1. To create and validate a meta-process including a set of meta-process assets for supporting it. These assets include a set of practices and strategies for supporting model representation and model transformation including an MDE tailoring strategy as its central asset.

2. To define and validate a visual approach to statically analyze software process models formally specified in order to identify specification problems before the process models are executed.

1.4 Research Method

1.4.1 Case Study Method

The research developed in this thesis is based on the case study method. According to Yin (1984) *case study is an empirical research method that investigates a contemporary phenomenon within its real-life context.* Yin (1984) suggests this method specially when the

boundary between the research object and its context is not well delimited. The case studies have been criticized as less valuable than analytical and controlled studies, including its powerlessness to generalize hypothesis. This corresponds to a one of the misunderstanding claimed by Flyvbjerg (2006) where it is not possible to generalize from individual cases. Thus, experimental and formal generalization are considered as valid tools of the scientific method, whereas *the case study* is used as an exploratory study (Rowley, 2002).

According to Eisenhardt (1989), the case study is also useful to validate theoretical models. The case studies can be generalizable if they are strategically selected from the universe of cases (Rosch, 1978). For instance, as reports Rosch (1978), *if the objective is to get the greatest possible amount of information on a given phenomenon, a representative case or a random sample may not be the most suitable strategy*, because an extreme case could offer the richest information to validate the research hypothesis. The case study adds to existing knowledge more knowledge to established theory or to go down into an established theory. A suitable method in empirical software engineering is the use of controlled experiments. However the control of the all variables in software engineering is very difficult (Shull *et al.*, 2002) because they include the human factor. A case study is an appropriate method in this research because it is difficult to controller all of the variables to understand the process tailoring and process analysis phenomena in industrial settings as Runeson & Höst (2009)suggest. Furthermore, in industrial settings there are few available cases to conduce experiments with a representative number of samples.

According to Klein & Myers (1999) there are three types of case studies: positivist, critical and interpretative. Following this classification, this thesis has used positivist case studies, where evidence for testing hypotheses is searched. The case studies in this thesis define their units of analysis in a determinisistic way. Flyvbjerg (2006) classify the case selection according to the information required as: *extreme/deviant*, *maximum variation*, *critical* and *paradigmatic*. The purpose of the selection in this thesis was driven by its representativity (Benbasat *et al.*, 1987). However, the case selected following a general availability criteria.

The complete research method description used in this thesis is presented from three points of view. Figure 1.3 depicts the phases of the research method and it refers to a management view. The three phases (and their milestones) are: exploration (problem defined), formulation (model defined) and validation (hypothesis validated). At the end of the exploration phase, the project proposal was accepted (Thesis I), by the end of the formulation phase a model (or validation mechanism) was completed (Thesis II and III) and when the validation was performed, the research was completed (Thesis IV).

In the exploration phase the main objective is to establish the problem statement. In the formulation phase the technical infrastructure was developed including methods,

techniques and tools associated to the different parts of the meta-process. In the validation phase a set of study cases for validating hypotheses 1 and 2 are designed and performed.

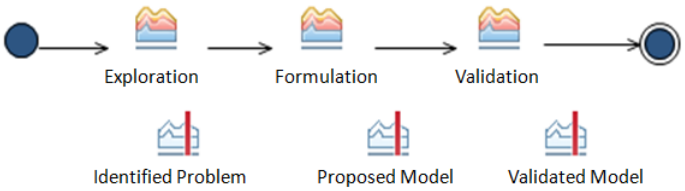

Figure 1.3: Research Method Phases

The second point view refers to the research disciplines (main activities): state of the art review, problem formulation, hypothesis formulation, validation design, model validation and documentation. These disciplines were executed many times with different levels of effort according to the phase in execution as it is shown in Figure 1.4.

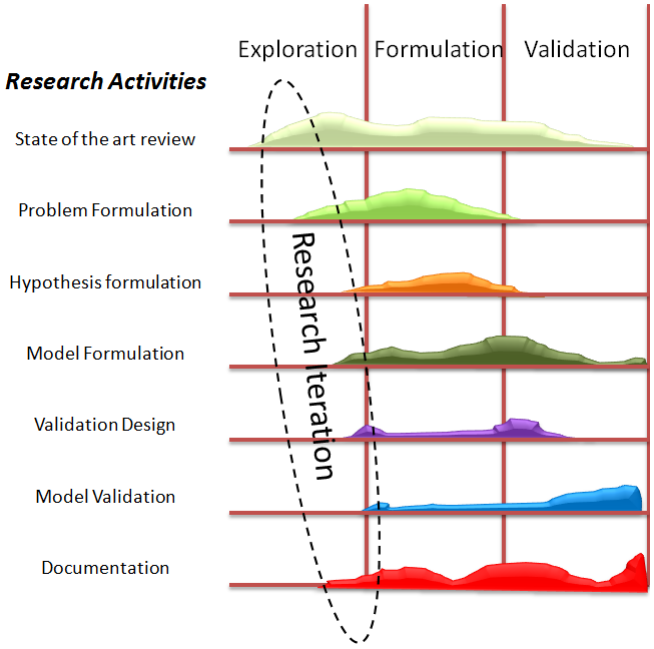

Figure 1.4: Research Disciplines

1. *State of the art review activity:* the relevant literature and industry was observed, studied and organized.

2. *Problem formulation activity :* a well defined problem statement was achieved. This activity motivates and defines a relevant problem for both, the industrial and scientific software process community.

3. *Hypothesis formulation activity :* the hypotheses were iteratively formulated and re-formulated according to the scientific and industrial review, and the problem statement.

4. *Model Formulation activity :* mechanisms were defined according to the hypotheses and the validation needs. The problem, the state of the art and the hypotheses were used to create the models (meta-process and meta-models) and tools to facilitate the evaluation of the hypothesis.

5. *Validation design activity:* validation instruments were defined and implemented. The instruments refer to non formal (test and initial cases) and industrial study cases according to the iteration performed. The software process line study cases were selected as *extreme* and *critical* (Requirements engineering), whereas the process analysis case studies were based on the availability of the EPF process models (both, community and industrial).

6. *Model Validation activity:* validation was performed according to the study cases defined at validation design. The case studies were performed according to the specified design.

7. *Documentation:* it was generated while the project advanced. So, technical reports, papers, process models and this thesis have been defined in an incremental way.

The above two points view, are related in a new third point view: the research iteration. A research iteration organizes the research disciplines in a workflow as it is presented in the Figure 1.5. In each iteration, the variable emphasis by phase is represented in Figure 1.4. In this thesis three iterations were executed with the following results:

1. First Iteration: software process improvement projects were observed and an initial version of the problem and the state of the art was established. The hypothesis was initially stated. Process models in EPF were defined and process assessments were conducted by the organizations. Two publications were obtained: *Software Process Improvement in Small Enterprises* published in ICSP 2009 (Pino *et al.*, 2009) and *Software process tailoring as a means for process knowledge reuse* published on Kreuse'009 (Hurtado & Bastarrica, 2009).

2. Second Iteration: a more complete description of the problem and the state of the art was established. The hypothesis was stated with a better definition than in the previous iteration. The models were defined and initial versions of the tools were implemented. Additionally, example process models were defined as software

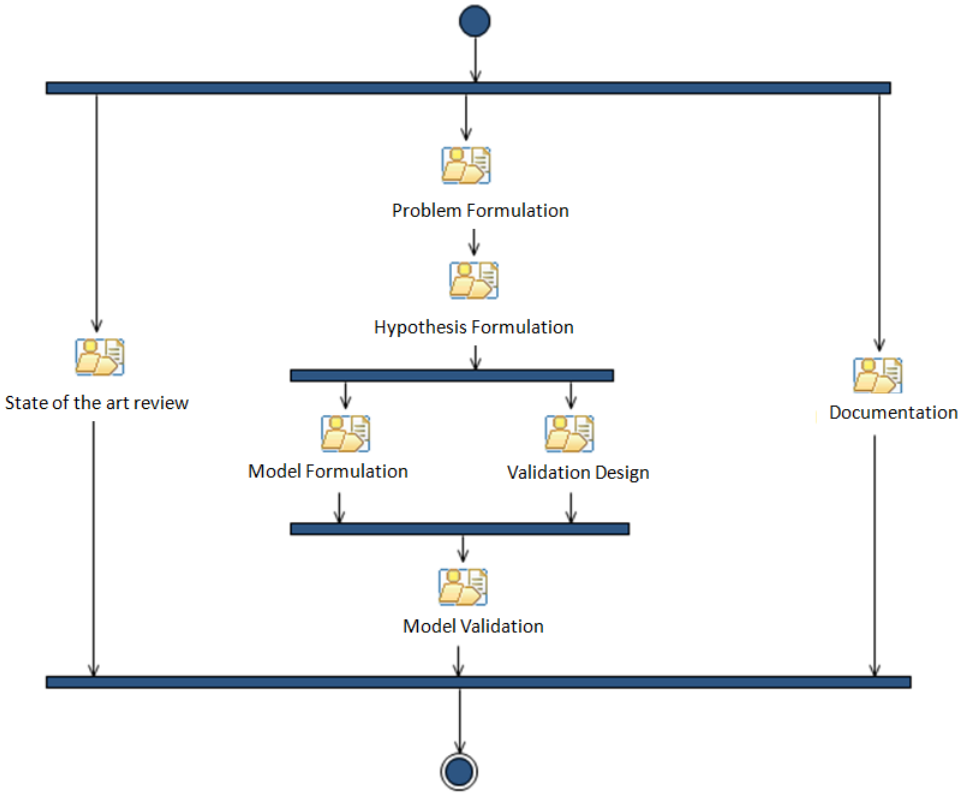

Figure 1.5: Research Iteration

process lines and available software process models were analyzed. A first proposal to process adaptation and analysis was defined and validated using small cases and a small industrial case in DTS enterprise. Two publications were presented: *Software process models blueprints* published on ICSP' 2010 (Hurtado *et al.*, 2010b) and *Analyzing Scrum with* AVISPA on the SCCC 2010 (Hurtado *et al.*, 2010a).

3. Third Iteration: industrial cases were defined to validate the hypothesis. Mechanisms were improved and applied in the study cases. More concrete results were obtained and published in this iteration. Four publications were presented: *Toward Lean Development in Formally Specified Software Processes* on the EuroSPI 2011 (Bastarrica *et al.*, 2011), *Analyzing software process models with* AVISPA (Hurtado *et al.*, 2011a) on the ICSSP 2011, *An MDE approach to software process tailoring* on the ICSSP 2011 (Hurtado *et al.*, 2011c), *Is it Safe to Adopt the Scrum Process Model?* on the CLEI Electronical Journal (Hurtado *et al.*, 2011b) and *Building Software Process Models with CASPER* on the ICSSP 2012 (Hurtado & Bastarrica, 2012).

1.5 Document content

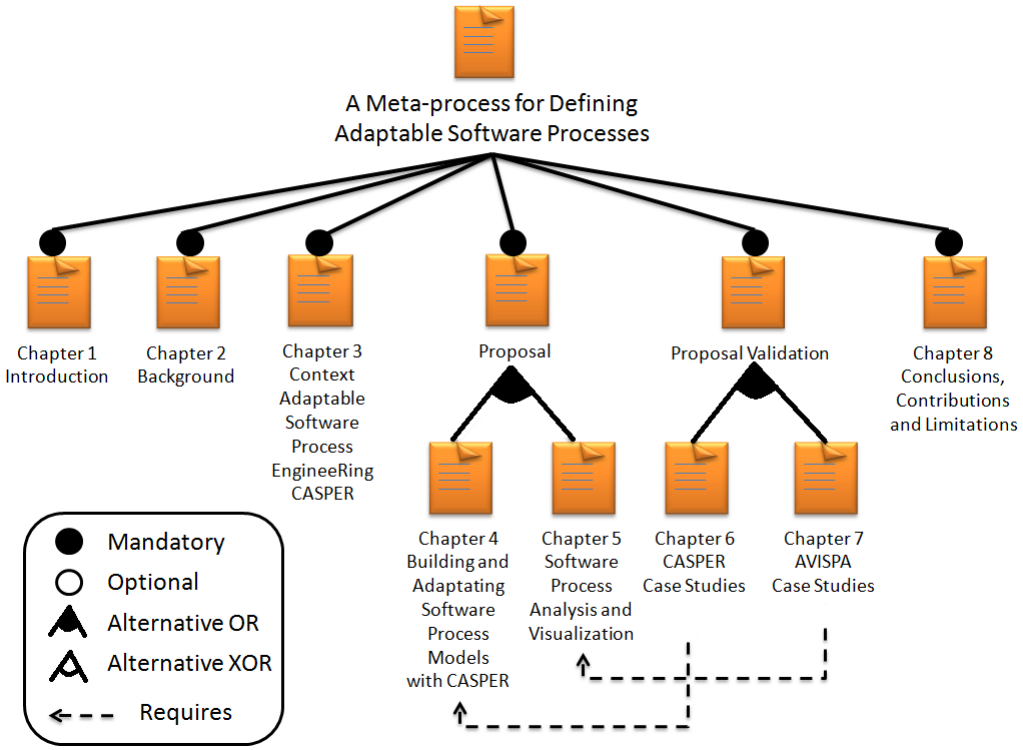

Figure 1.6: Thesis Feature Model

The rest of this document details the structure of the thesis. Chapter 2 includes the state of the art and the related work about software process tailoring, software process lines and software process model analysis. Chapter 3 presents the methodological framework: CASPER (Context Adaptable Software Process EngineeRing). Chapter 4 details how CASPER is applied, particularly how software process lines are developed and a process is produced. Chapter 5 explains software process blueprints and AVISPA, the approach and technology to software process analysis. Chapter 6 and Chapter 7 present the study cases where CASPER and AVISPA are respectively validated. Chapter 8 presents the main contributions and limitations of this thesis and some further work is explained. This document could be read of three ways. The feature model in Figure 1.6 shows these alternatives according to the reader preferences. Basically three initial review scenarios have been considered, a review on software process lines (Chapters 4 and 6), a review on software process model analysis (Chapters 5 and 7) and a complete review (Chapters 4, 5 ,6 and 7). The remaining chapters may be necessary to understand the thesis.

Chapter 2

Background and Related Work

Software processes have been recognized as a important asset for software development (Humphrey, 1989). However, for suitably defining, applying and analyzing software processes a great effort is required. This chapter defines the relevant concepts involved in this thesis. They range from basic concepts as what a software process is to specific concepts such as software process tailoring strategies including software process families.

2.1 The Software Process Concept

According to Humphrey (1989), *software process is a set of tools, methods, and practices for producing software products.* For Feiler & Humphrey (1993) and Conradi *et al.* (1993), *a software process is a set of partially ordered steps intended to reach a goal.* On the other hand Lonchamp (1993) defines the process as *a set of partially ordered process steps, with sets of related artifacts, human and computerized resources, organizational structures and constraints, intended to produce and maintain the requested software deliverables.* Additionally, some authors suggest that the software process is a complex entity (Fuggetta, 2000) that requires to use a staged and multiview (Jacobs & Marlin, 1996) approach.

Definition 1. *A Software Process is an entity explicity or implicity defined for representing, tailoring, enacting and executing a set of partially ordered software tasks with their associated performers, inputs, outputs and knowledge in a software development or maintance context*

Software processes are defined from simple elements called process elements or entities. The element names depend on the vocabulary used to define a software process. For example performer, agent, actor, stakeholder or role can be used to represent somebody performing a task, activity or step at a specific stage. This document uses a standard terminology based on the SPEM 2.0 language, described bellow, as a way of adhering

to a unique language for this thesis. Normally these entities can be considered key elements for reuse, tailoring, measurement, enactment or improvement. So, process elements can be considered as method elements or chunks as in Situational Method Engineering - SME (Ralyté *et al.*, 2003). A *method fragment* is defined as an indivisible part of a methodology according to Brinkkemper *et al.* (1998). These authors discriminate between two kinds of method fragments: process fragments (e.g. tasks) and product fragments (e.g. artifacts). Additionally, ther define a *method chunk* as a combination of a process fragments and product fragments very related (Ralyté & Rolland, 2001). A similar situation is presented in SPEM 2.0 (OMG, 2008): a process component includes *process elements* and *method content elements*. A process is defined as a special kind of reusable *ProcessComponent*.

A software process as an artifact, is normally expressed as a software process model. A model describes, at different detail levels, an organization of the elements of a process for designing, enacting, using, evaluating and improving. This process model can be analyzed, validated, and executed or simulated when it is an executable artifact. The process models can be used for software process control (evaluation and improvement) in an organization and for experimenting with software process theory and to ease process automation.

Definition 2. *A Software Process Model is an abstract software process description (Acuña & Ferré, 2001), (Lonchamp, 1993) to certain level of abstraction and a particular view on the process (Lonchamp, 1993) used for representating, tailoring, enacting, simulating or executing a software process.*

According to Lonchamp (1993) *An abstract description hides some aspects of the system or item being described: either definitional details, or some properties, or the way to implement the system or item.* Different process models can describe different points of view giving priority to particular concerns of a complex system (Humphrey, 1989) (Jacobs & Marlin, 1996). Some formalisms as suggest (Lonchamp, 1993), define views as perspectives, such as the functional, organizational, and behavioral facets. Process models could be described as a set of views when they are defined or analyzed.

2.2 Software Process Engineering

Software development organizations have found that count with defined processes can improve the organization performance and product quality (Humphrey, 1989). Provided that software process definitions' make high-quality software easier and more economical to produce, they have become widely valued and used. This means that software process

definitions are both useful for practitioners and reasonably economical to produce. Experience to date, however, demonstrates that the development of a comprehensive software process definition can be very expensive and time consuming. Thus, process design requires developing general purpose process definitions together with techniques for reusing, tailoring, enhancing and analysing them. Just as with software, this implies that large scale software processes should be carefully designed, constructed and validated.

According to (Feiler & Humphrey, 1993), *Process engineering is the research area involved with the practices for the definition, application, improvement and evolution of the software process.* Process engineering follows a meta-process, a conceptual framework for defining and building software process models (Lonchamp, 1993). *Software processes are software too* (Osterweil, 1987), so, a meta-process has many of the same artifacts and require quite similar disciplines and methods as software engineering. For example, one possible simple life cycle can start with clear process requirements followed by a process architecture and detailed process design stages. Processes must be validated against users' needs, and process prototypes (over pilot projects) may be needed before full scale development is undertaken.

Definition 3. *A software meta-process is a process describing the engineering of software processes. Meta-process artifacts are generic process models, tailored process models, instantiated process models, and model driven software processes among others. A model driven process is a process execution influenced by the interpretation of an instantiated process model, for example by a workflow machine.*

2.2.1 Software process engineering life cycle

Meta-process steps are intended for modeling, analyzing, supporting, and improving software process. Meta-process agents are either humans (e.g. process model engineers, process model managers, process model performers) or process support technology (Conradi *et al.*, 1992). A meta-process is highly related to the kind of software process model to be delivered. In Figure 2.1, we enumerate several main meta-process activities: process model design, tailoring, instantiation, enactment, validation, verification, monitoring, evolution and improvement (Madhavji, 1991), (Conradi *et al.*, 1992), (Humphrey, 1989). Actual meta-processes do not necessarily include all these meta-steps. Therefore, according to Madhavji (1991) *the boundaries between some steps may vanish in approaches with intertwined model design / customization / instantiation and enactment: even during enactment a guidance-oriented or enforcement-oriented model must be changeable.* Software process management is based on measurement, assessment and improvement activities

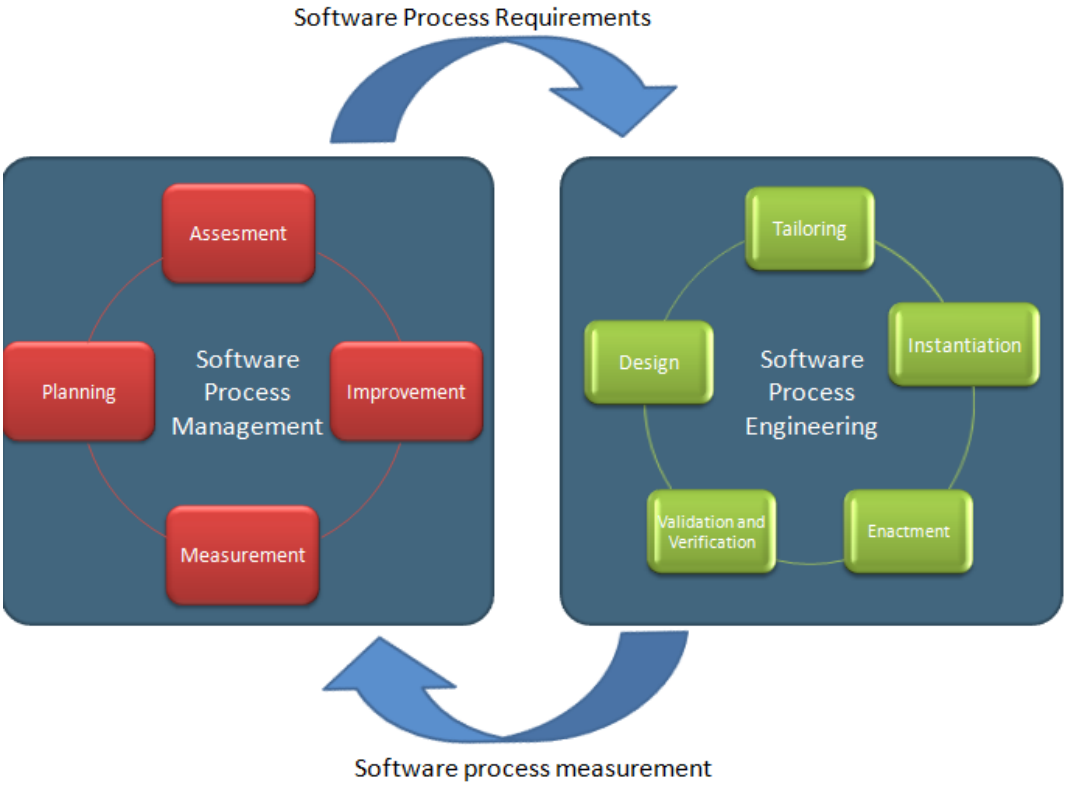

Figure 2.1: Software Process Engineering and Process Management Cycles

whereas software process engineering is based on analysis, design, tailoring, instantiation and enactment activities.

2.2.2 Software Process Modeling

The modeling of the software process is a meta-task to define a software process as a software process model (Acuña & Ferré, 2001). Different model representations describe, at different detail levels, an organization of the elements of a process (actual or future). Further, they provide definitions of the process to be used, instantiated, enacted or executed. So, a process model can be analyzed, validated, simulated or executed if it is suitably defined with these goals.

Curtis *et al.* (1992) present some benefits of software process modeling:

- *Ease of understanding and communication:* a process model must contain enough information for its application. Additionally, it formalizes the process providing a basis for training and knowledge reuse.
- *Process management support and control:* a process model facilitates project-specific software process adaptation, monitoring, management and coordination.

- *Provision for automated orientation for process performance:* a process model facilitates to define an effective software development environment, providing user orientation, instructions and reference material.
- *Provision for automated execution support:* a process model facilitates identifying automated process parts, cooperative work support, a set of metrics and process integrity assurance.
- *Process improvement support:* a process model facilitates reusing well defined and effective software processes, the selection or comparison of alternative processes and process development support.

One of the main challenges in the area of process engineering is to be able to take advantage of the huge variety of research and supporting technologies for process construction, tailoring, execution and evolution, including process modeling languages as Zamli (2004) state. A Process Modeling Language (PML) is feined by Fuggetta (2000) as *a particular language for modeling and describing software processes.* Examples of PMLs are E3 language (Jaccheri *et al.*, 1998), APEL (Dami *et al.*, 1998), Little-JIL (Osterweil, 1998), PROMENADE (Franch & Ribo, 1999) and SPEM 2.0 (OMG, 2008). SPEM 2.0 is a standard proposed and maintained by the Object Management Group (OMG). It is based on MOF (Meta Object Facility) and it has been the most popular language used to specify software processes (e.g., Scrum, XP and OpenUP). It also defines a UML profile in order to provide a mechanism to model processes using the UML language.

2.2.3 SPEM 2.0

Software and Systems Process Engineering Meta model SPEM 2.0 OMG (2008) is the OMG standard for process modeling. SPEM is a process engineering meta-model as well as a conceptual framework. This metamodel provides fundamental concepts for modeling, documenting, presenting, managing, interchanging, and enacting development methods and processes. An implementation of this meta-model would be oriented to different stakeholders such as process engineers, project leaders, project and program managers. In general any stakeholder responsible for implementing and maintaining processes for their development organizations or individual projects.

SPEM conceptual model, as depicted in Figure 2.2, allows identifying roles that represent a set of related skills, competencies, and responsibilities in the development team. Roles are responsible of specific types of work products. For creating and modifying work products, roles are assigned as performers in tasks where specific types of work products are consumed (inputs) and produced (outputs). Roles, work products and tasks could

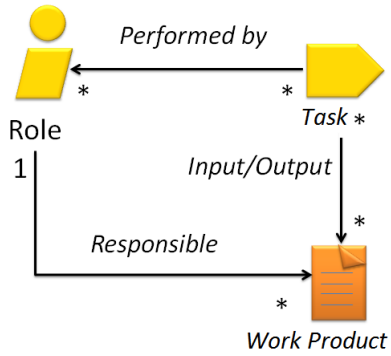

Figure 2.2: Conceptual Model of SPEM

have associated information. This information could be classified as fundamental and complementary. Fundamental information is described in the same element whereas complementary information is described as associated Guidance. Guidance is defined in SPEM as an element that provides additional information related to Describable Elements (this includes task, roles and work products). There exist specific kinds of guidance: Guidelines, Templates, Checklists, Tool Mentors, Estimates, Supporting Materials, Reports, Concepts, etc. To obtain a complete list of guidance kinds and detailed description refer to OMG (2008).

SPEM provides a standardized and managed representation of *method libraries* in order to allow reuse of method content. It aims to support development practitioners in defining a knowledge base for software development. It allows them to manage and deploy method library content using a standardized format. Such content could be licensed, acquired or reused. Examples of method content are tasks, work products, roles, examples, method definitions, white papers, guidelines, templates, principles, best practices, internal procedures and regulations, training material, and many other general descriptions of how to develop software. Details of these concepts are presented in OMG (2008).

SPEM provides a standardized representation of *process structure* for defining an independent life cycle, allowing method content to be placed into a specific process life cycle. Such processes can be represented as workflows and/or breakdown structures. Within this process, the reused method content can be then refined for its specific context. SPEM 2.0 also provides the conceptual foundation for process engineers and project managers for selecting, tailoring, and rapidly assembling processes for their concrete development projects.

The SPEM 2.0 meta model separates reusable method contents and their application in specific processes to promote reusability. Method content provides step-by-step explanations, describing how specific development goals are achieved independently of the

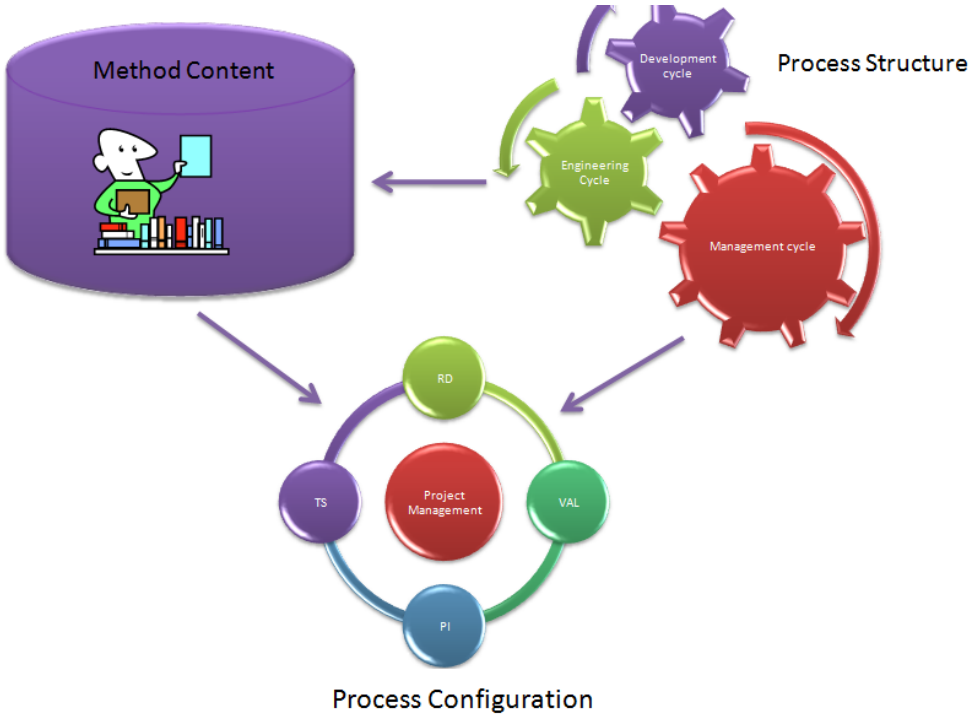

Figure 2.3: Conceptual Model of SPEM2.0

placement of these steps within a development life cycle. Processes take these method content elements and relate them into partially-ordered sequences that are customized to specific types of projects. SPEM 2.0 is structured in seven packages:

- *Core Package:* contains classes and abstractions that build the basis for all other packages.
- *Process Structure Package:* defines the basis for defining process models as a breakdown of nested *Activities* with the related performing *Roles*, as well as input/output *Work Products*. It also provides mechanisms for process reuse such as dynamic binding of process patterns that allow users to assemble processes with sets of dynamically linked Activities.
- *Process Behavior Package:* extends the static structures of the process models with externally defined behavioral models, e.g. UML state and activity diagrams.
- *Managed Content Package:* introduces concepts for managing content of development processes documented and managed as natural language descriptions. These concepts can either be used as standalone or in combination with process structure concepts.

- *Method Content Package:* adds concepts for defining life cycles and process-independent reusable method content elements that provide a basis of documented knowledge of software development methods, techniques, and concrete realizations of best practices. *Method content* describes how to achieve fine-grain development goals, by which roles, with which resources and results, independently of the placement of these elements within a specific development life cycle. The basic concept is *Method Content Element* (method fragment).

- *Process With Methods Package:* facilitates integrating processes defined with *Process Structure* with instances of *Method Content*. Whereas *Method Content* defines fundamental methods and techniques for software development, *process structure* places these methods and techniques into the context of a life cycle model.

- *Method Plugin Package:* introduces concepts for designing and managing maintainable, reusable, and configurable libraries of method content and processes. The concepts introduced in this package allow arranging different parts of such a library based on different layers of concern. Using concepts such as *Method Plugin*, *Process Component* and *Variability Types*, processes with a variety of capabilities can be defined. Variabilities can be used for defining variants of a method to choose among different choices at tailoring time.

For a detailed description of SPEM 2.0 meta model refer to OMG (2008).

2.3 Software Process Tailoring

There is no standard software process suitable for all development situations since appropriateness depends on various organizational, project and product characteristics, and what is even worse, all these characteristics evolve continuously. A one-size fits-all approach does not work for software development (Firesmith, 2004). Each project has its own characteristics and requires a particular range of techniques and strategies, and selecting a set of practices and integrating them into a coherent process should also be aligned with the business context (Cusumano *et al.*, 2009). In their process improvement approach, Dörr *et al.* (2008) suggest that the right set of practices for a project can be better found if we understand the context of the company. So, each project context should dictate the definition of the process that best fits it. Moreover, the particular process applied should not vary dramatically from one project to the other, so that process knowledge acquired by the development team could be reused. Tailoring is the process through which a general software process is configured for adapting it to the particularities

of the project at hand. Empirical studies show that process tailoring is difficult because it involves intensive knowledge generation and deployment (Rolland, 2009). If tailoring is carried out manually, it would involve an enormous effort and still be error prone, and the knowledge necessary for appropriately tailoring the process may be lost from one project to the following one.

Definition 4. *Software Process Tailoring is a step inside of a software meta-process for adapting a general software process to a specific context. This step generally occurs before process enactment: in this case the general process is an organizational software process and the specific process is a project context adapted process.*

Tailoring methods proposed by Dai & Li (2007) and Yoon *et al.* (2001) allow making tailoring decisions about deleting of merging process elements but the proposed formalisms and the generalization they propose make more complex the work of the project manager at tailoring time. Situational Method Engineering (SME) is a related area to software process tailoring focusing on the project specific method construction from a method repository (Ralyté *et al.*, 2003). In general SME defines three general techniques. The most common one is for assembling method chunks or process fragments. Another strategy is for extending an existing method, and the other one is for generating a method by abstraction/instantiation of a model/meta-model. Nevertheless, in most cases the effort for tailoring the process is huge, especially when the assembly is carried out at method construction time, as in Mirbel & Ralyé (2005). This is a big problem because process tailoring is normally not a responsibility of process engineers, but of the project manager. During the organizational process definition, an adaptable structure and a guide for process tailoring by situational knowledge can be defined, as proposed by Aharoni & Reinhartz-Berger (2008).

Process tailoring is a required practice for most processes. The Unified Process (Jacobson *et al.*, 1999) uses an *adjust guide tailoring approach* where tailoring rules are defined as recommendations to adapt phases, iterations and disciplines according to project specific situations. This approach requires a huge effort, knowledge and it is error prone. Agile methods, e.g. XP (Beck & Andres, 2004), define principles, values and practices where the process results as an emergent entity adapted to the project specific needs. So, agile methods follow an *auto-adaptable tailoring approach*. This approach is oriented to people, so the knowledge mainly flows in an oral way. Particularly, Crystal Methodology (Cockburn, 2000) is defined as an agile methodology family with four members: Clear, Yellow, Orange and Red. This classification can result insufficient for suitably covering each specific project. The ISO/IEC 29110 standard "Life cycle profiles for Very Small Entities" (ISO, 2011) defines a set of profiles, where VSEs selecting one of them according to Business

Models (commercial, contracting, in-house development, etc.), situational factors (such as criticality, uncertainty environment, etc.), and Risk Levels. According to O'Connor & Laporte, creating one profile for each possible combination of values of the various dimensions introduced above would result in an unmanageable set of profiles. Crystal and ISO/IEC 29110 tailoring follow a *template based tailoring approach*. Commercial processes such as RUP (Kruchten, 2003), use a *framework based tailoring approach*, where a general process is defined and a specific configuration is created by each specific project. Still, in this approach knowledge about processes is required at tailoring time. This approach requires using composition tools such as Method Composer of Rational IBM or Eclipse Process Composer. The framework strategy presents understandability problems and an overloaded model, and the template strategy presents the inherent difficulty of defining the adequate set of templates for the satisfaction of different types of projects (Bustard & Keenan, 2005). In this latter strategy, reuse is partially considered, but adaptation is not.

Martinho *et al.* (2008) present a two-step tailoring approach based on controlled flexibility. The process engineer expresses which, where and how changes can be applied onto process elements. The other process participants (process users) can easily identify which changes they are allowed to perform, and act accordingly. To support this approach, they have proposed a flexibility meta model, a modeling language, and a software process modeling tool called FlexEPFC. This approach is practical because each participant tailors the process fragment that he/she is interested in, however general inconsistencies could appear because to incompatible local decisions can be taken by the participants. So, this approach is still time consuming and error prone.

The above approaches do not facilitate mechanisms for reusing the tailoring knowledge. An *intelligent tailoring approach* has been proposed using case based reasoning (Xu, 2005), (Henninger & Baumgarten, 2001) and neuronal networks (Park *et al.*, 2006). In these cases, tailoring is based on an incremental set of previously tailored processes, so the benefits are achieved after various processes have been adapted. So, the knowledge and experience required at tailoring time is smaller when a good experience base has been defined. In an intelligent tailoring approach the specific situation is explicitly defined as a context encapsulated as a set of values for a set of context variables. When a new tailoring decision is required, it is manually implemented, however it improves the knowledge base for subsequent cases. The main difficulties in this approach correspond to the set up cost and the non planned change and evolution of various processes. So a change to the process may require training again the adaptation mechanism. So, this approach consumes a huge effort and time too.

Killisperger *et al.* (2009) propose an *instantiation based tailoring approach*. However there are few processes formalized up to an enactment level. So, this approach may result in little benefit. If it were possible, still some problems would arise. For example, decisions about replication of tasks, resource allocation and scheduling of events are added on the top of adaptation decisions. This approach makes very complex the tailoring process.

2.3.1 Process tailoring based on process families

Software Process Lines (SPrL) could be considered are a special kind of Software Product Line (SPL) (Clements & Northrop, 2001) (Osterweil, 1987).

Definition 5. *Software Process Line - SPrL is a set of related software processes defined from a set reusable of process assets in a planned way. SPrL enable a* planned software process tailoring approach.

2.3.1.1 The Early Age of the SPrLs

The first work widely known in SPrL was published by Sutton & Osterweil (1996b). In this work it is apparent the usefulness of viewing the software process as a process family. Osterweil based his work on the observation that the Booch Object Orientation Design (BOOD) method is a framework for describing different specific processes instead of only one unique specific process. Although a software process framework is not necessarily a SPrL, this observation and the analogy with the product engineering have been key concepts for following this approach. Following a SPrL approach, BOOD was used by the Programmable Design Process (PDP) project (Sutton & Osterweil, 1996a) for generating a variety of processes. The PDP project used a combination of mechanisms for selecting, specifying and tailoring processes. Rather than to show the process flexibility and adaptability, this work has risen key questions for this research line such as, how can the process family members be differentiated?, can one member be tailored for becoming other? So, this work suggests an integrated vision of product families and process families. It also states the need for characterization, use and management of commonalities and variabilities among family members. So, process families is a promising area where some interesting concerns are the identification, specification, support and adaptation of the SPrL. Osterweil's work is a first step in the identification of SPrL but it does not present a concrete proposal for dealing with SPrL, as this thesis intends.

A configuration management approach is proposed by Belkhatir & Estublier (1996) for the software process reuse and configuration. The idea is based on using a combination of an object-oriented view and the concept of process interface. For explaining this approach, the authors use the Process Interconnection Language (PIL), which was

built for process reuse and composition. The language provides version management characteristics for controlling the specific unit versions within a particular software process. The complete software process model is considered as a configuration, which can be automatically generated and may support multiple configurations simultaneously. The most relevant contributions of Belkahir's work are suggesting that the modeling language must be separated from the composition language, and that the configuration management can support process reuse and configuration at large scale. For theses reasons, the model defines a configuration model and includes the process unit concept (interface and implementation), the family concept (interface and variant group) and the configuration concept. The model makes a separation in three levels: meta class, family and instance. This work addresses the technical problem about the control and management process models as a family; however, it does not offer a specific approach for defining and applying software processes as the framework as is proposed in this thesis. The work of Belkhatir & Estublier (1996) does not make emphasis on the way of adapting process models to specific contexts such as projects and execution machines, which is the production strategy proposed in this thesis; however, a process configuration strategy as that shown in their work must be included in any meta-process proposal.

2.3.1.2 The emergence of the SPrLs

Simidchieva *et al.* (2007) present an approach for defining a process and a set of variants as a process family. This work proposes a formal approach for defining process families by characterizing them. They also use the language Little-Jil for implementing an example. The purpose of this work is showing that the language and the approach are appropriate for defining process families. The key elements for process family management lie on the language used that allows a separation of concerns (specification, coordination, agents and artifacts), visual representation, and an experimental platform for modifying the process allowing variants. Generating variants is achieved by the technique of process components reuse and combination, based on the user specifications. In the example used, the authors create process variants by changing an agent's behavior, a task elaboration and an artifact. The work neither considers any particular technique for SPrL, nor it is closed to one particular definition of a SPrL. The scope of this work is process lines in general and not SPrL, so its scope is wider than ours. On the other hand, our thesis is centered around software processes as a specific domain, and it intents to define a conceptual framework for SPrL and based on this framework, to define a systematic meta-process for building SPrLs. However, some ideas suggested by Simidchieva *et al.* (2007) will be taken and adapted for our work.

Rombach proposes the SPrL concept as a systematic mechanism for managing a process and its variants (Rombach, 2005). This work is a motivation for SPrL work because it strengthens the relation between software product lines and software process lines stated by Sutton & Osterweil (1996a). However, the main contribution of his work is about establishing possible further work for SPrL. The work highlights that PMLs must include variability management mechanisms, effective methods for creating empirically supported processes as evidence-based software engineering methods or value-oriented software engineering methods; it also includes theoretical and engineering foundations for an integrated vision of SPrL and software product lines. Rombach's work has generated great interest, but it neither includes a particular approach for modeling SPrLs, nor includes a concrete meta-process. This thesis uses and extends his perspective, and intends to take the following step by proposing a concrete methodology for building and applying SPrLs.

In Hanssen *et al.* (2005), the tailoring of a software process based on the SPrL concept is presented. The method uses a SPrL in a specific domain (automobile industry), it follows a top-down adaptation approach, and it supports a bottom-up refinement of the generic process based on tracing the instantiated processes. The work shows through an informal validation study that tailoring, within the SPrL approach, is efficient and it yields adherence to a generic process at adaptation time. This method is about the SPrL concept, but it does not define a specific meta-process or tailoring strategy. This thesis proposes a generalization of this approach defining a meta-process based on a generative tailoring strategy.

Following Rombach's path, Washizaki (2006) proposes a concrete work in SPrL: the SPrL architecture construction via an ascendant approach. Washizaki's work proposes a technique for tailoring the process through SPrL. A SPrL is understood as a set of common processes within the same problem domain, and a SPrL architecture that specifies commonalities and variabilities. The architecture is the basis for deriving the SPrL member processes using a global optimization view. Furthermore, a conceptual framework and concrete definitions are provided; also a practical example is presented where a software process is tailored. However, the conceptual framework is limited in terms of methodological issues for defining the scope, the architecture, the process components, commonalities and variabilities. Although this work defines a special notation for representing commonalities and variabilities (a SPEM 2.0 extension), the work is centered in factorization and reverse engineering, and it does not prescribe any method for process engineering. Another characteristic is that the work considers the process composition as the only mechanism for process customization to generate a new process instance and it does not consider that complete models could also be defined from adaptations either.

Other work following the bottom-up strategy to SPrL is presented by Ocampo *et al.* (2005). The approach is based on a tool for factorizing commonalities among several software process models. To do so, the work shows an initial evaluation in the Wireless Internet Services domain. The steps followed by this work are: define pilot processes, execute pilot processes, observe and model processes, identify and evaluate processes and derived practices, analyze commonalities and variabilities, and finally, create a reference model. Again, it is a work where the SPrL is based on the factorization strategy. Although this alternative is feasible for building SPrLs, the process asset composition (in a context of a huge number of components) is poor and the adherence to a general process is hard to achieve (Zhu *et al.*, 2007). The work proposed in this thesis takes the ideas of these other work for representing variabilities and commonalities, but, it will follow a generative approach based on model transformations. This work will use a descendant approach based on reusing and adapting process models, and using transformations as part of the production strategy in the meta-process for SPrL. So, reuse is applied in a systematic way instead of an opportunistic way. Similarly, model adaptation can be reused too and therefore, the process assets associated to these models could also be reused. Although the SPrL is created as a whole, it could use some factorization strategies for its definition. The software process architecture will be considered as the core asset of the SPrL as proposed by Washizaki (2006).

2.3.2 Model Driven Engineering in software process tailoring

According to France & Rumpe (2007) the Model-driven engineering (MDE) is a software development approach where abstract models are defined and systematically transformed into more concrete models, and eventually into source code. This approach promotes reuse through a generative strategy. MDE has been originally created for software engineering, but following the Osterweil's premise (Osterweil, 1987), MDE can also be applied to software process engineering as Bretón & Bézivin (2001) propose. In particular, transformation techniques have been used as instantiation strategies in process engineering (Killisperger *et al.*, 2009), (Bézivin & Bretón, 2004). This thesis is based on the ideas of formally defining process models and using transformations to manipulate them. For this reason, MDE will be used for separating the process definition in two abstraction levels: organizational process level and project specific process level. In the organizational process model a general and abstract process model is defined, that is the core of the SPrL. Via transformations the next level model is produced. A transformation is an automatic activity based on contextualized information about project and the environment respectively.

This thesis follows a software process line approach for supporting the software process tailoring. Process domain engineering facilitates the planned reuse anticipating the variability in the process model (Armbrust *et al.*, 2009), so a model transformation is proposed as a tailoring strategy to produce project-specific process models (Bretón & Bézivin, 2001) (Hurtado & Bastarrica, 2009). Context information modeling (Kajko-Mattsson, 2010) is achieved using a Process Model Context Meta model. The thesis approach searches a separation of Software Process Engineering and Software Engineering Domains where process modeling stakeholders and process enactment stakeholders (project stakeholders) are usually not clearly distinguished in other approaches (Bai *et al.*, 2010). Additionally, software process tailoring is complex, requiring the reuse of intensive tailoring knowledge reuse (Hurtado & Bastarrica, 2009). Thus, a process modeling approach should provide ways to cost-efficiently instantiate a general (or combined) process model into a project-specific process model (Armbrust *et al.*, 2009). The process adaptation should be reduced to defining a specific situation, because the main problem for software engineers is to build software and not software processes. So, adaptation should result appropriate and not time consuming to software engineering stakeholders and the software process models and software process tailoring rationale could be reused between different projects.

2.4 Validating Software Process Models

Software quality is a key issue in software engineering, and it highly depends on the process used for developing it (Humphrey, 1989). Quality of process models is also a key issue in process engineering (Lonchamp, 1993). Software process model quality can be addressed from different approaches: metrics (Cánfora *et al.*, 2005), testing (ISO/IEC, 1998; SEI, 2006), simulation (Gruhn, 1991) and formal verification (Ge *et al.*, 2006).

2.4.1 Software Process Testing

Process testing is an effective way to evaluate a process model; assessments and audits are based on data of executed projects, but executing the process is generally expensive. Our approach provides a means for a priori evaluation of the software process quality. Cook & Wolf (1999) present a formal verification and validation technique for testing and measuring the discrepancies between process models and actual executions. The main limitation of testing is that it can only be carried out once the process model has already been implemented, tailored and enacted. When process models are tested according to their specifications, they are validated in a similar way to software testing (Osterweil, 1987).

An example of process testing is a software process assessment, where a process and its corresponding model are evaluated based on a capability maturity framework. This kind of approach is present in CMMI (SEI, 2006) and ISO/IEC15504 (ISO/IEC, 1998). This kind of testing activities can only be carried out once the process model has already been implemented, possibly tailored and enacted. This approach checks for the adherence to a standard but it does not evaluate the appropriateness of the process for the organizational or project needs. Process testing requires very long time cycles. Gruhn (Gruhn, 1991) has proposed a verification technique based on simulation results and execution trace evaluation. Simulation has a shorter cycle, but it still requires enactment data. This is an appropriate verification technique if the process is known to be suitable for the environment, but if the model is incomplete, underspecified or it is not suitable for the organization, then the process model simulation will not reflect the expected results.

2.4.2 Software Process Simulation

Simulation has an usefull shorter cycle to validate proposals, but it still requires enactment data to be a reliable execution. Software process simulation is useful for instance to project managers for supporting management decisions Raffo & Kellner (2000). To analyze process models using simulation sophisticated and complex techniques are required, simulation requires historical data normally unavailable when a process is defined the first time. However, with data available and a suitable simulation approach, simulation is an effective tool to analyze software process models, taking in count dynamic elements also. *It is hard to adopt a simulation technology because of the difficulty in developing a simulation model* Park & Bae (2011). Park & Bae (2011) propose reduce the complexity of a simulation model resolving the lack of historical data, deriving the simulation model from a descriptive process model widely adopted.

2.4.3 Software Process Model Metrics

Cook & Wolf (1999) propose formal verification and validation technique for identifying and measuring the discrepancies between process models and actual executions. Cánfora *et al.* (2005) have defined some ratio metrics for measuring the maintainability of overall software process. However metrics for partial portions of the process or individual process elements have not defined. These mesurements besides being generals, are usually not a suitable presentation for a reviewer. Based on this general data it is hard to know what is wrong, where the error is located, and how to find opportunities for improvement when there is no apparent error. Pérez *et al.* (1996) suggest to evaluate the congruence between the process model and a given environment based on past data. However, obtaining these

measurements is hard and the results are not necessarily precise. Johnson & Brockman (1998) use execution histories for predicting process execution in order of estimates a project. The above metrics do not address suitability, completeness or consistency of the process model, however these metrics were any way a key starting point in the AVISPA development.

2.4.4 Software Process Formal Verification

Formal specifications and checking based on Petri Nets in a multi-view approach are presented by Ge *et al.* (2006). Because the particularity and the complexity, this views include activity view, product view and role view, which are considered in AVISPA too. However the formal checking covers sintactic issues but has practical and semantic limitations. Atkinson & Noll (2003a) state that *data-flow analysis can uncover specification errors, such as misspelled resource names, that can exist in otherwise syntactically correct process specifications.* They suggest that analysis is requieres, to detect these problems before the process be executed. The analysis includes verifying that the resource flow specified by the process execute correctly. The process engineer can validate that the process inputs are used to generate the respective outputs, identifying potential errors in a process specification. Additionally, the flow analysis aids to identify opportunities of redesign of the analized process.

2.4.5 Software Process Analysis

Software reviews, such as inspections, walkthroughs, and technical reviews, are widely used to improve software quality, so they are promising techniques for software processes too (Osterweil, 1987). The main idea is that a project team of software engineers, in order to review and improve a software product, can to detect defects that the product designer or developer could have introduced. According to Sauer *et al.* (2000), the review process has three main tasks: defect discovery, defect collection, and defect discrimination. The defect discovery is the one that demands the highest expertise and knowledge. Scacchi (2000) employs a knowledge-based approach to analyzing process models identifying problems related to consistency, completeness, and traceability of a process model. Based on the evidence about usefulness of the visualization in data analysis and problem detection presented by (Lee *et al.*, 2003), visualization may be also applicable for analyzing different concerns about process models using the human visual capacity for interpreting the process in context and to evaluate its consistence and suitability as in software visualization (Larkin & Simon, 1987). So, this thesis is based on these assertions and it proposes and uses a visual mechanism for process model validation.

Using visualization to identify error patterns is not new. A lot of research use visual approaches to identify positive or negative properties of software systems (Lanza & Ducasse, 2003; Perin *et al.*, 2010; **?**). However, none of the related work we are aware of elaborated on visual patterns for identifying problems in software process models. We have developed and applied AVISPA, a visual approach and tool for analyzing the process models defined in three different Chilean software companies.

Another technique similar to AVISPA is presented by Atkinson & Noll (2003b) for analyzing the flow of resources through a process. This technique, derived from research in data-flow analysis of conventional programming languages, enables a process designer to analyze outputs, waste work products, the coherence between control work flow and data work flow, improvement opportunities in the resources flow (e.g. parallel tasks). The analysis is oriented to search specification errors, indicating re-design needs. Furthermore the analysis highlight flaws in the underlying process, indicating a potential process improvements. However, AVISPA metrics and visualization facilitates analyzing not only syntactic aspects from process specifications but also semantic errors (in conceptualization).

Soto *et al.* (2009) present a case study that analyzes a process model evolution using the history of a large process model using the configuration management system with the purpose of understanding model changes and their consequences. Soto's goal is oriented toward the evaluation of the impact of the changes in software process models whereas AVISPA's approach is oriented to early analyzing a new or changed process model just before the model is used in a specific project. Soto's approach analyzes the changes on the process elements comparing many XML files, while in the AVISPA tool the analysis is realized using a unique and static process model using XML files exported from the EPF tool.

Analysis of SPEM models, in particular to community models such as Scrum, is limited. A close work by Osterweil & Wise (2010) presents an analysis of Scrum using Little-JIL (Osterweil, 1998), a graphical language for defining processes where autonomous agents are coordinated.the language includes the use of resources during a task. In Little-JIL executable programs, the agents are conduced through a process achieving that their actions satisfy a process constraints. Little-JIL programs can be statically analyzed to ensure that process requirements are satisfied for all executions. A Little-JIL process program defines a variety of ways of accomplishing tasks that can work with varying resource requirements and varying agent capabilities. Agents may be human or automated (software or robots). Little-JIL process programs conduce to agents through a set of alternatives and facilitate their communication and resource sharing. Preconditions and postconditions are used to dynamically verify that the process is being applied correctly.

Resources are defined using a resource model and are reserved and locked during the execution of a process.

Little-JIL has been applied to analyze the Scrum Software process Osterweil & Wise (2010). This analysis determines a weak point when the product is integrated in each development work. So, provided that continuous integration is not part of Scrum, it can fall in long periods of development without integration, generating a bottleneck. However Avispa based analysis, additionally detected other potential bottleneck problem; guidance missing, multi purpose tasks, and too demanded work products. Whereas Osterweil & Wise (2010) follow a formal and agent-oriented approach, Avispa focuses on analyzing SPEM 2.0 process models. The Avispa approach could be implemented to analyze other languages including Little-JIL. In Avispa approach, the process designer can gain some insights about the process model design and potentially find some problems.

2.4.6 Software Process Model Analysis by Visualization

Considering that software engineers use UML models as inputs and outputs of its development tasks, Lange *et al.* (2007) propose an approach to analyze system properties using metrics of the UML model or from its context information (version control system). On these metrics, they propose a set of views to visualize the information to support the tasks and they validate its proposal on a large-scale controlled experiment to validate the usefulness of the proposed views. The views were implemented in the MetricView Evolution tool. This approach analyze the requirements for performing the tasks, but it does not include tacit and explicit knowledge of the process, for instance the participant roles, the produced artefacts (furthermore models) or the associated knowledge to the task (guidance). Sadiq & Orlowska (2000) present a visual verification approach where a set of graph reduction rules are used for identifying structural conflicts in process models providing a well-defined correctness (deadlock free, lack of synchronization free) criteria in a process model. The Sadiq & Orlowska (2000) work use Petri Nets and workflow graphs, however, this technique could be applied to extend the analyzis capabilities of Avispa, introducing new problematic patterns.

2.5 Synthesis and Discussion

The works described above show the need for defining a practical framework for software process engineering for building adaptable software processes. None of the works here presented takes the SPrL in an integrated way as the Software Engineering Institute did by creating a framework for software product lines (Clements & Northrop, 2001). This work is

a starting point for intending to unify concepts and for establishing a specific path to build and apply adaptable software processes in a way close to real organizations. This work proposes and implements a generative strategy for automatically tailoring organizational processes, modeled as SPEM 2.0 instances to particular project contexts, based on MDE techniques so that appropriate processes are achieved rapidly and with little effort. The SPEM 2.0 models are recovered and visualized independently using some basic metrics for a practical evaluation. The validation approach includes a tool that supports the software process model review based on visual mechanisms.

Chapter 3

Context Adaptable Software Process EngineeRing - CASPER

3.1 Introduction to CASPER

Suitable software process models are required for reaching quality and productivity in software projects. The most suitable process for a particular project depends on the organizational context, the project context, and the rigor required. Due to both, high costs and the time consumed, to define a specific process for every project results unfeasible. So, software process models need to be built for reuse and adaptation.

A Software Process Line - SPrL is a special Software Product Line - SPL in the software process engineering domain. Families of software processes share common features as well as exhibit variability (Sutton & Osterweil, 1996b). Consequently, a process family is a promising approach for achieving a high level of reuse and adaptation (Simidchieva *et al.*, 2007), (Rombach, 2005), (Washizaki, 2006). In a SPrL, process domain engineering develops and maintains process assets promoting planned reuse, while classic tailoring re-actively integrates unanticipated variability in the process model (Armbrust *et al.*, 2009).

This thesis propose CASPER (Context Adaptable Software Process EngineeRing), a meta-process to define context adaptable software process models. CASPER facilitates situational method engineering (SME) or software process tailoring defining a software process line instead of just a software process. CASPER uses coherently three key approaches as shown in Figure 3.1: software process lines, context and process modeling and MDE based tailoring. CASPER has been building on the follow four principles:

- *Principle 1. Separation of Software Process Engineering and Software Engineering Domains*: process modeling stakeholders and process enactment stakehold-

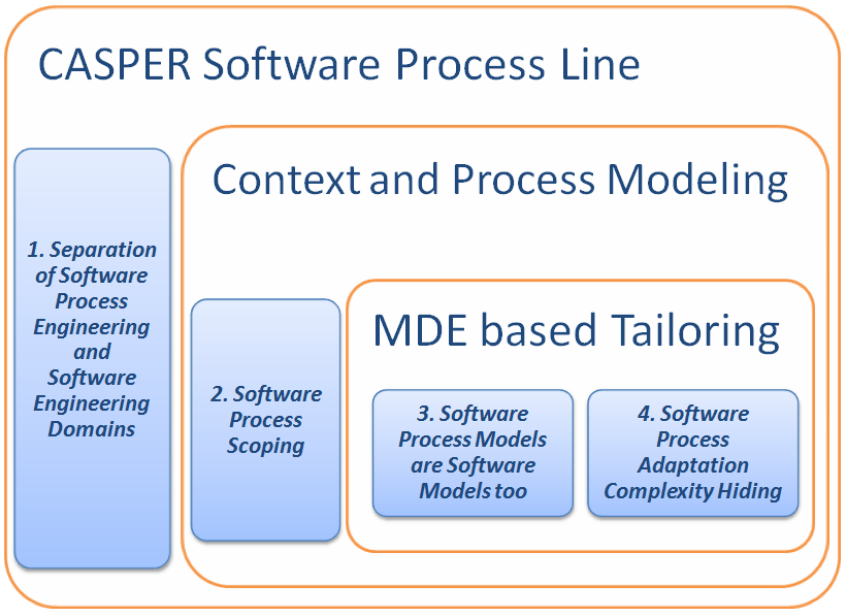

Figure 3.1: Principles of CASPER

ers (project stakeholders) are usually not clearly distinguished (Bai *et al.*, 2010). Software process engineering stakeholders' required capabilities are different from software engineering stakeholders' capabilities. Consequently, the software process model definition task (including its adaptation mechanisms) should be assigned to the process engineering group, and the process adaptation task should be a responsibility of the project management group before project planning. Thus, CASPER is a SPrL approach that separates process domain engineering from process application engineering activities. A SPrL facilitates a planned reuse and evolution of software process models and their elements.

- *Principle 2. Software Process Scoping*: it is in principle unclear which decision models can help determining which process can be applied to a specific project (Pérez *et al.*, 1996), (Boehm *et al.*, 1995). Particularly to SPrL, it is necessary to determine which processes and process elements should be part of the process line, and which should not (Armbrust *et al.*, 2008). Software process engineers need to define the limits of the SPrL. Scoping the SPrL will determine situations where the process will be used and which process elements (common and variable) will be required in each situation. Moreover, planned situations should be related to process features for determining suitable process configurations for each situation. A CASPER process line is formally scoped using a Context Model that defines all different possible project circumstances, a Process Feature Model that identifies all possible process elements that may vary and their relationships.

- *Principle 3. Software Process Models are Software Models too*: it is a direct conclusion of the fact that software processes are software too (Osterweil, 1987). So, similar to software models, process models could use the MDE approach for different goals: design, adaptation, simulation and enactment of process models (Bretón & Bézivin, 2001), (Killisperger *et al.*, 2009), (Bézivin & Bretón, 2004). Hence, software process models and software process tailoring rationale could be reused along different projects. CASPER uses an MDE approach for tailoring software process models including context models to define where these process models will be applied. Each specific situation can be represented before project planning and used to automatically produce a specific process model.

- *Principle 4. Software Process Adaptation Complexity Hiding*: process tailoring involves intensive knowledge and it is usually time consuming mainly when it is done manually (Rolland, 2009) (Ocampo *et al.*, 2005). A process modeling approach should provide ways to cost-efficiently tailor a general (or combined) process model into a project-specific process model (Armbrust *et al.*, 2009). The process adaptation should just consist of defining a specific situation, because the main problem for software engineers is to build software and not software processes. So, adaptation should not be time consuming for software engineering stakeholders. A CASPER process line uses an MDE production strategy. The software process engineering group, according to the SPrL scope, defines an adaptable software process model, an adaptation context (organizational context) and a set of tailoring rules. The software engineering group defines a context model configuration (project context) and automatically executes the transformation and validates the specific process model.

3.1.1 CASPER in a Nutshell

CASPER follows three basic key tactics: tailoring is context sensible (context-adaptable), the process reuse and adaptation is planned (software process line), and rules relating context information to process model variants are programmed (tailoring strategy). In Figure 3.2, a project context has been defined by two context attributes (Time Constraints), each attribute includes two values. Furthermore, Figure 3.2 shows a process including an optional Activity (Use Case Realization) and an activity with two non-exclusive options (Database Design, Persistence Framework Design). Context attributes and process variants have been related (blue arrows) according to past adaptation information. These relations define what and how variants will be resolved at tailoring time and so these relationships determine the scope of the software process line. These relationships could

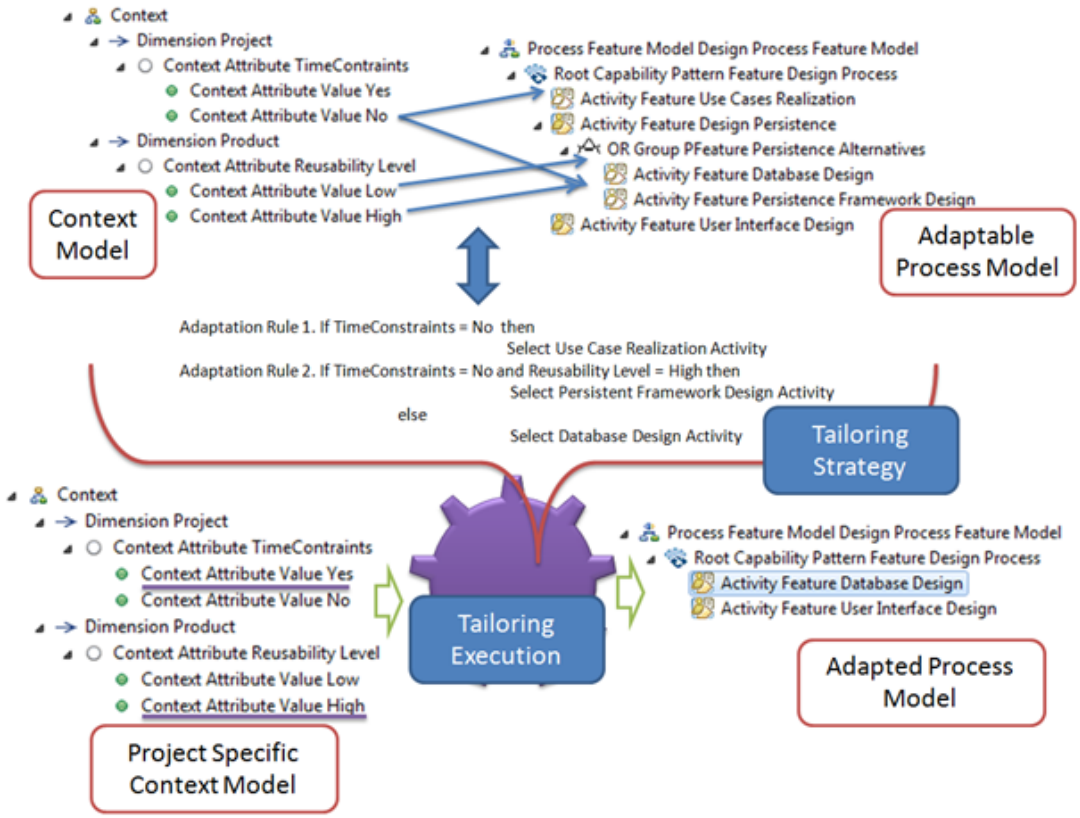

Figure 3.2: Software Process Engineering for SPrL

be programmed reusing the tailoring knowledge and thus achieving an automatic tailoring strategy. For instance a specific project with a high reusability level but with time constraints will produce a process like the one presented as adapted process model, by applying the rules described in the tailoring strategy. So, the tailoring task complexity is reduced to only defining a specific context.

3.1.2 CASPER Subprocesses

CASPER meta-process is presented in Figure 3.3. It embodies two main subprocesses:

Process domain engineering: this is an iterative process focused on capturing the software process domain knowledge and developing the process model core assets (contexts, process features, process components, reference process models, tailoring rationale and concrete tailoring rules) for enabling the implementation of each context-adapted process model.

Process application engineering: a context-adapted software process model is automatically produced according to the requirements of a specific project using a reference process model. The production strategy is an MDE implementation where the tailoring

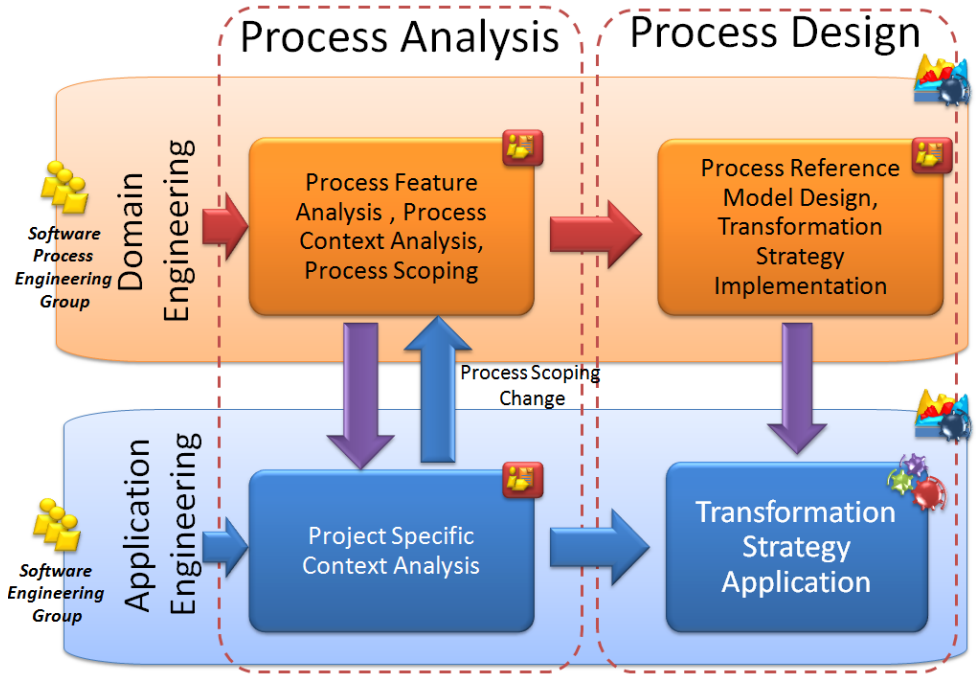

Figure 3.3: Software Process Engineering for SPrL

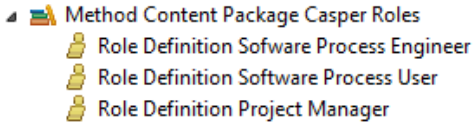

Figure 3.4: CASPER Roles

rationale is expressed as a set of transformation rules.

These subprocesses are described in more detail in Chapter 4. Each subprocess consists of different tasks, produces specific artifacts, and defines the roles to perform these tasks.

When CASPER is adopted within an organization, this organization should include two different groups of engineers: one for domain software process engineering(Software Process Engineering Group - SPEG) and another for software process application engineering(Project Team - PT). The first group corresponds to a SPEG in a Software Process Improvement(SPI) approach. The second one corresponds to people in charge of applying the software process to specific projects. The SPEG takes care of the development and evolution of the process family. The project team produces and applies a family member according to the project-specific context. The CASPER roles are shown in Figure 3.4.

A roadmap of CASPER including stages, deliverables, languages and activities is presented in Table 3.1

Table 3.1: CASPER Road map

Stage	Deliverables	Representation	Activities
Domain Analysis	Context Model Feature Process Model Scope Model	SPCM PFMM SPSMM	Context Analysis Process Feature Analysis Software Process Scope Analysis
Domain Design	Organizational Process Model MDE Production Strategy	SPEM SPEM ATL	Software process architecture design Software process model design Software process model transformation implementation
Application Analysis	Configuration Context Model	SPCM	Project Specific Context Analysis
Application Design	Adapted Software Process Model	SPEM	MDE production strategy execution Manual tailoring(optional)
Verification and Validation	Software Process Blueprints	AVISPA Visual Model	Visual process model analysis

3.2 CASPER Domain Engineering

Process domain engineering is an iterative process that captures the knowledge gained in the software process domain. This knowledge is expressed as context models, process features, process components, software process models and tailoring rationale. This process, as defined in Figure 3.5, includes five general activities: process context analysis, process feature analysis, scoping analysis, reference process model design, and production strategy implementation. These activities are performed by both, the software process engineering group and the software engineering group; however, the software engineering group mainly participates as a source of requirements and experience, and for validating the resulting models.

The process context analysis activity allows the understanding of process requirements, where a context model is defined to cover the possible situations of a project in the organization. The process feature analysis activity identifies and defines process features that establish the general process elements and their classification as common, optional or alternative. The scoping activity validates the context characteristics, the process features and defines their relationships. The process reference model design activity includes the process features in a reference process model. The reference process model design builds a process model according to the identified process features using the variability and reuse mechanisms available in the process modeling language. The strategy production implementation activity defines and implements a set of decisions that adapt (tailoring rationale) the process model in specific variation points using situational or contextualized information represented as a configuration context model. These decisions are implemented as a set of model transformations based on information of the process

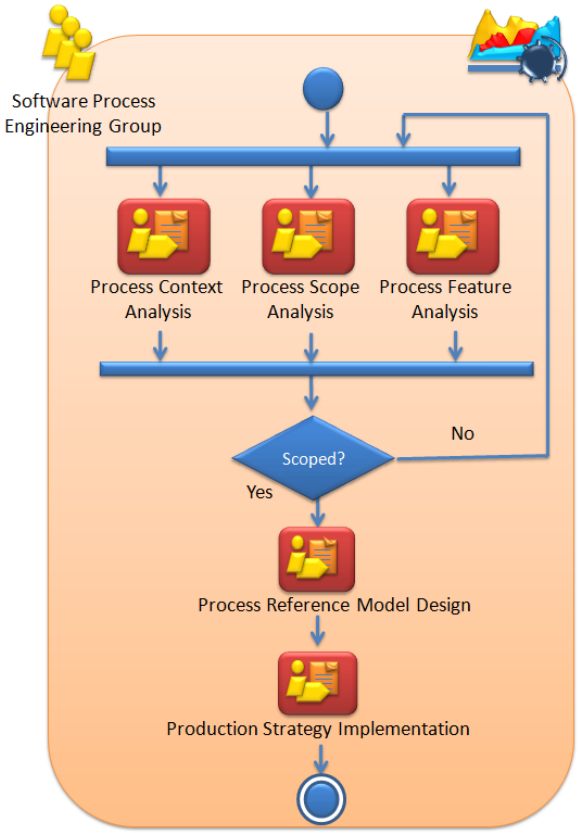

Figure 3.5: Domain Engineering Meta-Process

feature model, the reference process model, the context model and their relationships.

3.2.1 Software Process Context Analysis

In the process context analysis activity the set of situations where the SPrL will enable to produce suitable software process models is defined. A context is modeled by dimensions, context attributes and their possible values (context attribute values). This activity requires to determine the dimensions covered and their potential variable and constant attributes. The attributes and their possible values should be delimited using past projects, future projects or using the knowledge and experience of the project managers and process engineers (Pérez *et al.*, 1996), (Armbrust *et al.*, 2008), (Boehm *et al.*, 1995), (Koolmanojwong & Boehm, 2010b). First, the dimensions are defined. Dimensions are sets of related attributes to achieve the separation of concerns in context modeling. Next, for each dimension, relevant attributes are defined including their possible values. If there is only one possible value for a certain attribute then the context attribute is a constant, otherwise the attribute is variable and it may take one of a set of possible values. This information is presented in a Context Model. The context model, including

its possible configurations, should be defined using the Software Process Context Meta model - SPCM detailed in Chapter 4.

3.2.2 Software Process Feature Analysis

Different process models are analyzed in order to identify process variants and justifications for them (Ocampo *et al.*, 2005) recognizing the differences between some canonical situations (e.g. software process templates). This information will enable the creation of a reusable (and adaptable) reference process model. Therefore, the process should be analyzed as a process feature model. Process feature analysis is used to identify the common and variable characteristic of the process-line members. In the domain design, the points and ranges of variation captured in process feature models are implemented as part of the reference process model. Software process features are a special kind of software features. Process properties (life cycle type, maturity level, etc), method elements (method fragments), process elements (process components, process fragments), process with method elements (chunks) and method plug-in elements (reusable components, processes and configurations) are examples of process features. We use the feature model proposed by Kang *et al.* (1998), but using SPEM 2.0 stereotypes for modeling software process variabilities.

3.2.3 Software Process Scope Analysis

In the software process scope analysis activity process it is decided which features should be supported, and which should not (Armbrust *et al.*, 2008). The process scope in CASPER is explicitly defined through a cross reference (or mapping) between process features and context characteristics as Pérez *et al.* (1996) propose. This cross reference is an abstract view of the tailoring decisions to be implemented during the production strategy implementation.

3.2.4 Software Process Reference Model Design

A process reference model is designed for supporting commonalities and variabilities specified in the process feature model from one or more, explicit or implicit, current or future software process models (Rombach, 2005). A reference process model is designed according to the process features using variability mechanisms (e.g. extend, contribute and replace in SPEM 2.0) and reuse mechanisms (e.g. process components, patterns, configurations, plug-ins and links between process elements and method elements in SPEM 2.0) available in the process modeling language.

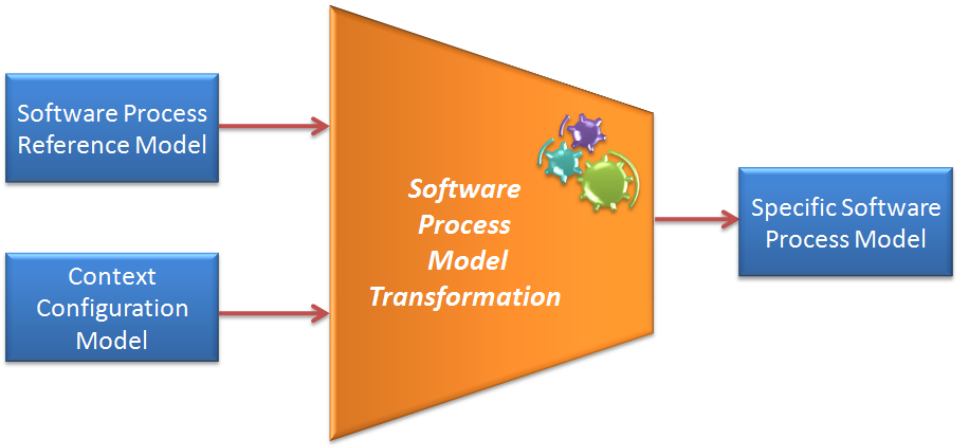

Figure 3.6: Process Production Strategy

A reference process model may be structured (Washizaki, 2006) according to one or more reference models or a life cycle (internal or external). For example, a process could be built using the life cycle of the Unified Process (Jacobson *et al.*, 1999), according to phases, disciplines and iterations. However, other process components (e.g. project management, support processes) can be organized according to categories of CMMI (SEI, 2006). A hybrid approach could also be used for achieving the goals of the implementation. So, a set of process patterns, life cycles, commercial process models (e.g. RUP (Kruchten, 2003)), and community process models (e.g. Scrum (Schwaber, 1995)) can be used as a starting point. In this thesis, both, the reference process model and the project-specific process model, are specified in SPEM 2.0.

3.2.5 Production Strategy Implementation

As Figure 3.6 shows, the production strategy implementation activity defines a model transformations using as input a reference process model and a context configuration model, and as output a project-specific process model (tailored model). A project-specific process is suitable for a specific context configuration (1), the context attributes have been previously related to variabilities of the feature process model during scoping (2) and these variabilities have been implemented in the reference process model during design (3). Consequently, a transformation rule set can be specified (from 1 to 3) in order to create a project-specific software process model. The transformation rules follow an MDE approach on process models (Bretón & Bézivin, 2001), (Bézivin & Bretón, 2004), (Killisperger *et al.*, 2009). This transformation is implemented in a transformation language such as Atlas Transformation Language - ATL, Epsilon Transformation Language - ETL

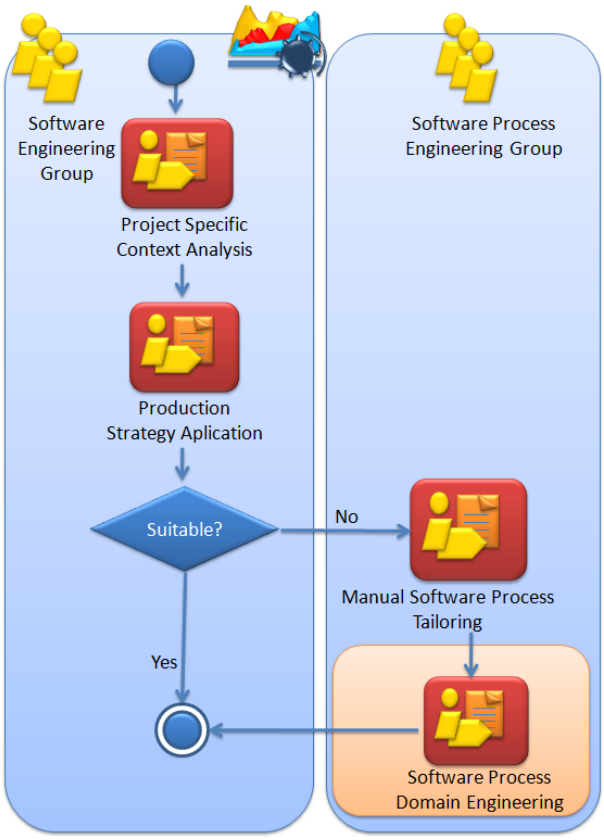

Figure 3.7: CASPER Application Engineering

or Query/View/Transformation - QVT, but it could also been eventually implemented in a general purpose language.

3.3 CASPER Application Engineering: Context-Based Software Process Adaptation

The application engineering embodies the activities depicted in Figure 3.7. Application engineering defines a project specific context configuration according to project information and executes the transformation strategy. This strategy has been previously defined in the domain engineering process to generate the project-specific software process model. If it is not possible to model the situation with the context model, the most approximated context configuration should be defined by the software engineering group. Then, the obtained model can be manually tailored by the software process engineering group. In this case, a scoping decision must be made by this group refining the SPrL:

1. The SPrL scope will not be extended and the new software process model will not belong to the software process line.

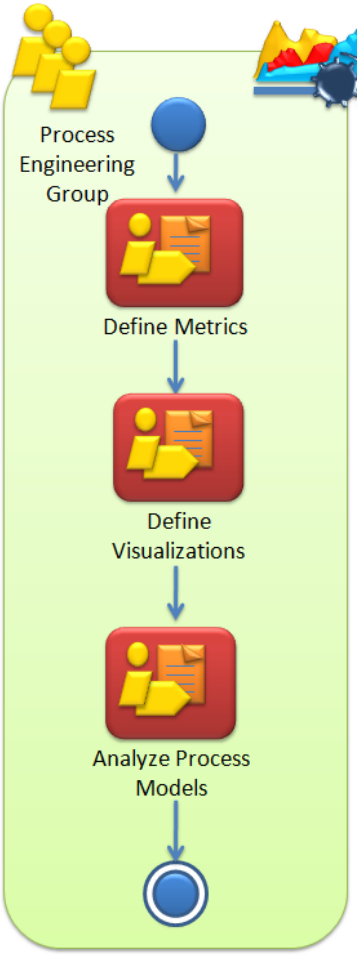

Figure 3.8: Software Process Validation based on AVISPA

2. The SPrL scope will be extended, thus the new process is part of the family. Hence, the domain engineering activities must be re-executed so that the new process requirements are also considered (Scope Analysis).

3.4 Process Model Analysis using AVISPA

The process models specified in SPEM 2.0 should be analyzed because the investment realized in the domain models is significant. The validation process is presented in Figure 3.8. The proposed model validation follows an architectural recovery approach to evaluate different process characteristics. Three process architectural view types - ROLE BLUEPRINT, TASK BLUEPRINT and WORK PRODUCT BLUEPRINT- have been defined to support the analysis. These blueprints are process partial views that make use of metrics computed from the process model for helping understanding and evaluating different aspects not directly available in SPEM 2.0 and its associated tools.

These visualizations have been implemented in AVISPA[1], a tool that builds blueprints and highlights error patterns (Hurtado *et al.*, 2011a) . Counting on this tool, the process engineer only needs to analyze highlighted elements, not demanding much experience and also requiring little previous knowledge for effective process model analysis, and adding usability as well. AVISPA's implementation is a based on Moose technology and the Mondrian tool. Visualization in general aids in data analysis and problem detection (Lee *et al.*, 2003). Using different concerns about process models in a graphical manner, we make use of the human visual capacity for interpreting the process in context and to evaluate its coherence and suitability (Larkin & Simon, 1987). The analysis is realized via a visual inspection of the software process model by the process model engineer, however some insights are offered by AVISPA for localizing some recurrent problems. The AVISPA approach and tool are presented in Chapter 5.

3.5 Synthesis and Discussion

This chapter showed an overview of a practical framework for software process engineering for building adaptable software processes called CASPER. This framework is based on four principles: *Separation of Software Process Engineering and Software Engineering Domains, Software Process Scoping, Software Process Adaptation Complexity Hiding* and *Software Process Models are Software Models too.* Based on these principles the CASPER process is built based on three basic concepts: software process lines, context and process modeling, and MDE based adaptation.

Armbrust *et al.* (2008) define four requirements (R) to adequately scoping software processes and two constraints (C). CASPER meets these needs as follows:

1. *(R) Support software product development*: context modeling in CASPER helps determining which attributes are relevant for the organization and specific software processes. So, a product dimension, including attributes as complexity and size, should be defined supporting product concerns.

2. *(R) The approach provides ways to characterize software products, projects, and processes accordingly*: context modeling in CASPER is based on dimensions, so characteristics can be of different nature and they can be classified and organized in these dimensions. So, product dimensions can be defined including complexity and size characteristics. In a similar way characteristics of project, process and other dimensions can be added.

[1]Freely available at http://www.moosetechnology.org/tools/ProcessModel.

3. *(R) Distinguish stable process parts from variable ones*: process feature models in CASPER aid to identify common and variable process features explicitly (process components, process elements, method elements, chunks and others). These features are implemented in SPEM 2.0.

4. *(R) Incorporate unanticipated variability in a controlled manner*: the software product line approach in CASPER allows to produce a project specific process in a planned way because an MDE strategy on a scoped infrastructure is implemented during domain engineering.

5. *(C) Provide ways to store stable and variable parts within one process model*: the process models are defined in SPEM 2.0. In this language the variabilities are implemented, however in a process feature model the variation points should be identified, stored, related and managed.

6. *(C) Provide ways to cost-efficiently instantiate such a combined model into a project-specific process model*: provided that the tailoring is planned, each project-specific process is obtained in an automatic way from a situation specification. The situation specification is realized by configuring the context model according to the specific needs in the new project.

Chapter 4

Building and Adapting Software Process Models with CASPER

4.1 Introduction

This chapter details the main activities defined by CASPER including: software process context analysis, process features analysis, process scoping, reference software process design, production strategy implementation and software process tailoring itself. The production strategy is the essence of CASPER and it is one the main contributions of this thesis. All CASPER activities and artifacts are defined around this strategy. The activities and the resulting work products are illustrated with an actual academic requirements process of an advanced software engineering course. Section 4.2 presents the example process (*CC51A-RE Requirements Process*). Sections 4.3, 4.4, 4.5 and 4.6 detail the main activities and techniques of the domain engineering to obtain the context model, the feature model, the scope model and the process model. To complete the domain engineering, section 4.7 details the production strategy. Section 4.8 details the application engineering, i.e., the software process model adaptation. Section 4.9 presents the preliminary results of the case study.

4.2 Example Problem: CC51A Requirements Engineering Software Process Line

A general requirements engineering process is used by advanced computer science students of the University of Chile while developing a real world project as part of a regular 5th year course. The projects developed involve teams of 5 to 6 students and an advisor. The advisor and most of the students have some professional experience. The projects also

involve real world problems and real world clients representing the work environment of a real world small software company (Hurtado & Bastarrica, 2010). Nowadays, this type of company represents most of the software development task force around the world (Neumuller & Grunbacher, 2006; Von Wangenheim *et al.*, 2006). However the process is not applied exactly in the same way for all cases in the course. The lecturer normally introduces the software process with the idea that students tailor it to their specific project. However, some students apply the general process just as it is and others spend long hours trying of adapt it. Additionally, when control points are applied, the auxiliary professor found that the processes are either missing elements or have more than necessary. Therefore, this introduces learning problems, missconceptions about the project or a major development effort. Therefore, a tailoring solution is required to allow the projects to work with the most suitable software process in each case.

4.3 Software Process Context Analysis

The context of a project may vary according to different project variables such as: product size, project duration, product complexity, team size, application domain knowledge, and familiarity with the involved technology, among others. The study case of Pérez *et al.* (1996) defines the characteristics of a software process model and its environment, and determines how congruent the process model is in the given environment using project information of a specific organization. On the other hand, COCOMO II (Boehm *et al.*, 1995) defines a cost model for projects based on situational information. This information is defined as a fix set of factors and their scaling factors. The context dimension is introduced by Mirbel & Ralyé (2005) as a way to separate context concerns in a matching strategy for selecting chunks in a road map approach to SME. In the work of Bucher *et al.* (2006) a context engineering method is proposed where context factors are identified and analyzed to enable the engineering of contextual methods. Royce (1998) presents an approach to tailor the Unified Process based on two dimensions of characteristics: technical complexity and management complexity. The characteristics in the literature cited above are discriminating factors such as scale, stakeholder cohesion or contention, process flexibility or rigor, process maturity, architectural risk and domain experience. The unified Process is a horizontal process framework, and the idea of CASPER is to define a process at the organizational level (vertical reuse), so the context dimensions and factors are those defined by the organization nature. According to Aharoni & Reinhartz-Berger (2008) a specific context can be defined as a vector of characteristics that relate to the organization, the project, the developing team, the customer, etc. However, defining the context as a formal model enables us to automatically tailor the organizational process

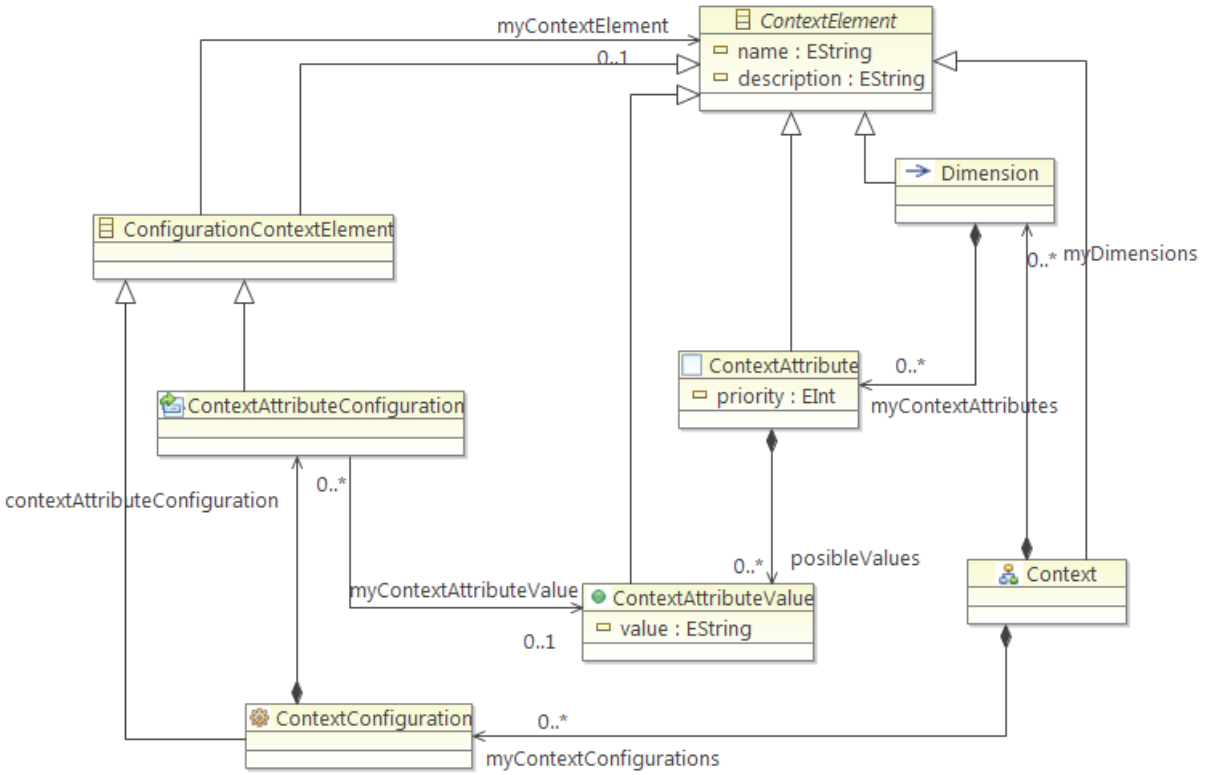

Figure 4.1: Software Process Context Meta model

according to it. The SPCM (Software Process Context Meta model) has been defined as part of this thesis by (Hurtado *et al.*, 2011c) in order to express both context models at the organizational level and the context configuration models at project specific level. This meta model is depicted in the Figure 4.1.

4.3.1 Software Process Context Meta model - SPCM

SPCM is a meta model defined in CASPER to express software process contexts. SPCM is based on three basic concepts: *Context Attribute*, *Dimension* and *Context Attribute Configuration*. Every element in SPCM extends a *Context Element* that has a name and a description. A *Context Attribute* represents a relevant characteristic of the process context that may be required for tailoring. The *Context Attribute* includes a priority (used when a trade-off between context attributes is required) and it can take one of a set of values defined as *Context Attribute Value*. An example of a *Context Attribute* is the Project Size. A *Context Attribute Value* represents a specific value for qualifying a *Context Attribute*. Examples of *Context Attribute Values* for the Project Size Context Attribute are Small, Medium and Large. A *Dimension* represents a collection of related *Context Attributes*. A *Dimension* eases the separation of concerns applied to Context Attributes. An example

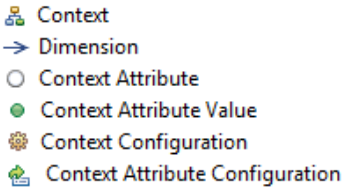

Figure 4.2: Context Model Visual Stereotypes

of *Dimension* is Team dimension, referring to team attributes such as team size, team capabilities, team skills and team cohesion, among others. A *Context* is represented as a collection of *Dimensions*. A *Context* represents the whole context model. To represent possible specific process contexts, a *Context Configuration* can be defined from the context model. A *Context Configuration* is a collection of *Context Attribute Configuration* that is set to one of the possible *Context Attribute Values* for a *Context Attribute*. Therefore, a *Context Attribute Configuration* is associated to a *Context Attribute* and to one unique *Context Attribute Value*. An example of a *Context Attribute Configuration* is the Project Size Configuration for a small project, where its *Context Attribute* is Project Size and the *Attribute Value* associated is Small. A graphical notation has been defined to express each of the context concepts as shown in the Figure 4.2.

4.3.2 Context Modeling with SPMC

Requirements engineering refers to the process of elicitation, analysis, specification and validation of real-world goals for software systems, functions of software systems, and constraints on software systems (Zave, 1997). It is also concerned with the connection of these requirements to particular specifications of software behavior, and to their evolution over time and across software families. The context model could be used as one of the requirements artifact in the process engineering domain. This fact is supported by the principle that first it should be determined the context before designing or comparing processes (Armbrust & Rombach, 2011). Context definition could be achieved by applying software requirements techniques to the software process domain.

- *Context Elicitation:* elicitation starts identifying the information sources including past projects, process experts, previous process adaptations, identified kinds of projects, process templates, process configurations and tailoring guides. According to the specific source, the information is analyzed in order to determine the relevant context attributes. For example, the elicitation work could be conducted as a workshop, so an initial list of the context attributes could be obtained using brainstorming, and then an analysis is conducted by the process engineering team.

Context Attributes	Description	Fixed	Value
Developers	Number of people in the development team	X	5–6
Duration	Project duration (months)	X	3
Training	Project team education in Computer Science (years)	X	5
Complexity	Size of the problem or the technical solution		Medium or High
Domain knowledge	Familiarity with the problem domain		YES or NO
Project type	Software development, evolution or reengineering		D, E or R

Table 4.1: Context attributes of CC51A-RE Process

Otherwise an initial proposal of context model (prototype) could be developed first and analyzed during the workshop. Some other techniques that could be used in this step are: develop a focus group with the project managers or an interview with a process expert. This information could be obtained incrementally as the process domain engineering continues (See Chapter 3). So, different context characteristics could be defined and exemplified. Each characteristic is defined including its possible values (domain). General sources of relevant context information can be found in (Boehm *et al.*, 1995; Pérez *et al.*, 1996; Royce, 1998). In the CC51A RE-Process case, the elicitation was conducted with an interview to the course lecturer (expert in the process). In an open interview, attributes and their possible values that would be used for adapting the general process to specific situations were identified.

- *Context Analysis:* in practice not all combinations of context attribute values are relevant for adaptation. According to Bucher *et al.* (2006), a process model must address only those combinations occurring with a certain frequency in practice. To extract these combinations, a survey or a workshop can be conducted. These techniques could aid to determinate all potentially relevant context attributes and their values in the context set for which the process model should be applicable. The result of this analysis should be groups of context attribute values and specific combinations that are relevant. In CC51A-RE case study and using an initial context model, a workshop was conduced. The expert stated that, with respect to *people*, there were no relevant differences among projects, but the *complexity of the solution*, the *project type* and the *domain knowledge* could vary substantially from one project to another. Therefore, these context attributes were analyzed considering their possible values. The resultant context variables are presented in the Table 4.1.

- *Context Specification:* the context model is formally specified as an instance of SPCM. So, the related context attributes are defined and grouped by dimensions. For each attribute the domain is defined as a set of possible values. The context model of CC51A-RE is depicted in the Figure 4.3 according to the previous elicitation. The context includes three dimensions: project, product and team. Some

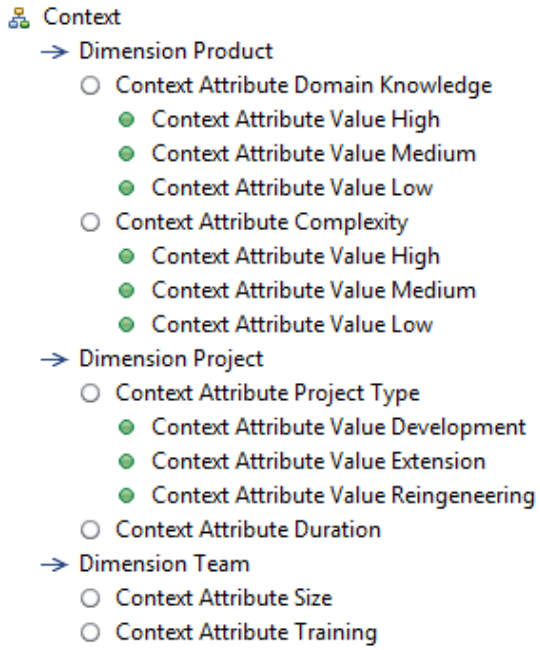

Figure 4.3: Context Model of CC51A-RE Process

other attributes were identified but their domains were not specified because these attributes were modeled as fixed according to the expert. In the identified dimensions a high detail was required to define the possible values of *Domain Knowledge* attribute (High, Medium and Low) and the possible values of the *Complexity* attribute (High, Medium and Low). A priority could be introduced to the context attribute if a trade-off between two o more context attributes is required at tailoring time. The priority could also be specified afterwards when the scope is analyzed and defined.

4.4 Software Process Features Analysis

Feature modeling was proposed as part of the Feature-Oriented Domain Analysis (FODA) method (Kang *et al.*, 1998), and since then, it has been applied in a number of different domains (Antkiewicz & Czarnecki, 2004). Feature modeling is used to model the common and variable properties in product-lines. At early stages, feature modeling enables product-line scoping, i.e., deciding which features should be supported by a product line and which should not. In domain design, the points and ranges of variation captured in feature models need to be mapped to a common product-line architecture. Cardinality-based feature modeling (Kang *et al.*, 1998) extends the original feature modeling from FODA with feature and group cardinalities, feature attributes, feature diagram references,

and user-defined annotations. A feature diagram is a visual notation of a feature model, which is basically an *alternative* (OR) or *inclusive* (AND) tree of features (represented by a tree symbol) where each sub-tree can be optional (represented by a white circle) or mandatory (represented by a black circle). *Requires and excludes* constraints between nodes can also be defined (represented by unidirectional arrows or bidirectional arrows respectively).

In CASPER software process features are considerer as a special kind of software features. Examples of process features are process properties (life cycle type, maturity level, etc), method elements (method fragments), process elements (process components, process fragments), process with method elements (chunks), method plug-in elements (reusable components, process patterns, processes and configurations), process packages, method packages and categories. We use the feature model notation proposed by Czarnecki & Antkiewicz (2005), but using SPEM 2.0 stereotypes. Because a software process has multiple views, these perspectives such as role view, process view and work product view among others, could be defined in one or many process feature models. Elements in the same and different perspectives can include relevant constraints for a coherent process assemble, for example the *requires* constraint. In general a process feature could be implemented with many process pieces and a process piece could implement many features. However, SPEM 2.0 stereotypes could be suitably used to identify the process and method points where variation (and their variants) will be required.

4.4.1 Process Feature Meta model PFMM

PFMM is a meta-model defined in CASPER to define the features of a software process line. As Figure 4.4 shows, PFMM is based on basic the concept of feature Czarnecki & Antkiewicz (2005) extended as a process feature to be applied to software processes: *PFeature*. Every *PFeature* could be decomposed into more process features. A *PFeature* is sub classified as a *Group Feature* and as a *SpecificPFeature*. The former in turn is sub classified as *OR GroupPFeature* or *XOR GroupPFeature* to facilitate alternative process features. The *SpecificPFeature* is classified according to implementable variation points in a software process (the SPEM concepts have been used): *WPFeature*, *TaskFeature*, *GuidanceFeature*, *TaskUseFeature*, *RolFeature*, *ActivityFeature*, *CapabilityPatternFeature*, *DeliveryProcessFeature* and *PluginFeature*. A *ProcessFeatureModel* is a tree including a *RootPFeature*, where its children are references as its *PFeature members*. *MethodLibraryFeature*, *RootPluginFeature*, *RootCapabilityPattern* and *RootDeliveryProcessFeature* are specific features of *RootPFeature*. Furthermore, a *ProcessFeatureModel* includes a set of Constraints about dependencies between *SpecificProcessFeatures* defined as *Requires*

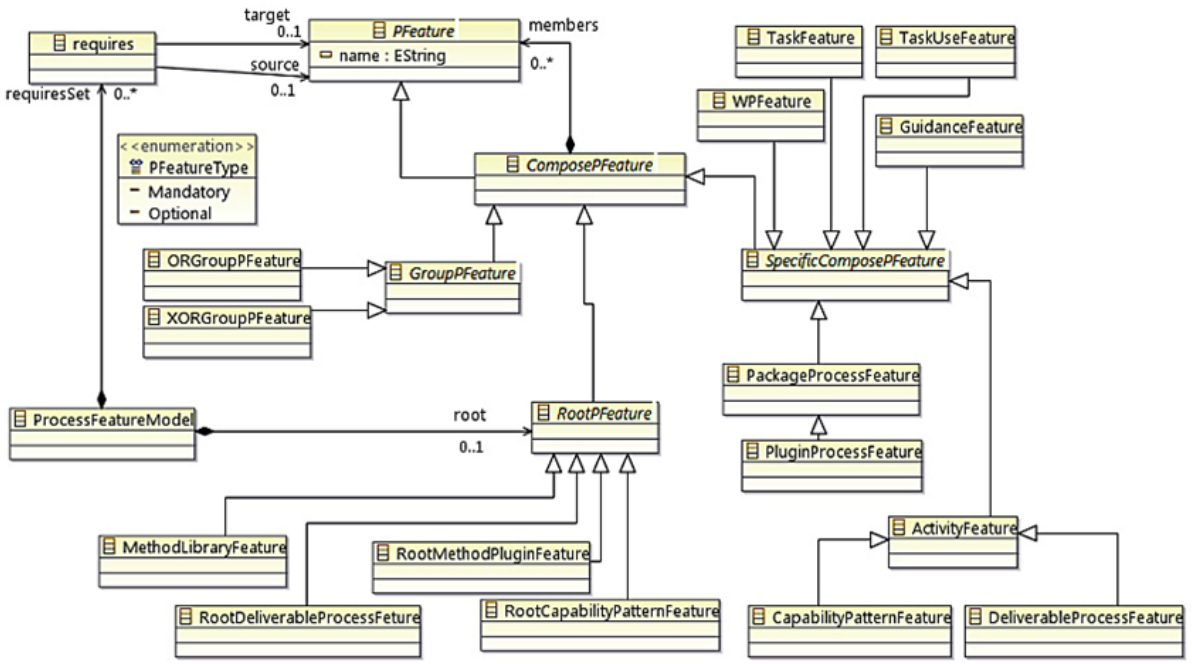

Figure 4.4: Process Feature Meta model PFMM

elements, where two *SpecificProcessFeatures* are related when a dependence appears between them.

4.4.2 Process Feature Modeling with PFMM

Modeling the process features is a task concurrently executed with context analysis and scope analysis tasks. Thus, elicitation, analysis and specification form a recurrent pattern.

- Process Mining: similar to context elicitation, this task starts by identifying the information sources including past projects, process experts and process adaptations, identify kinds of projects, process templates, process configurations and tailoring guidelines. According to the specific source, the information is analyzed in order to determine the relevant process and consider the family members. The mining work could be conducted as knowledge exploration based on interviews to the process engineer and the process users. This information is obtained incrementally as the process domain engineering continues (see Chapter 3). In the CC51A RE-Process case, the mining was conducted by interviewing to the course lecturer (expert in the process). In a work session the process owner presented the general software process defined for the course and some examples about tailored specific processes.

- Commonality and Variability Analysis: Similar to Ocampo *et al.* (2005) proposal, a commonality analysis is conducted in CASPER. It identifies common and variant

process components. In the elicitation task a set of software process models are identified. Similar process components are identified among these processes using a process walk through, evaluating the projects, interviewing to users or conducting a focus group. The common and variant components could appear in any granularity level, so a decomposition of the software process as is suggested by the modeling language could be applied. For instance, SPEM plug-in and package structure, the breakdown structure and links between process elements and method elements are used to obtain the structure of the process model of CC51A-RE, and so it is applied to identify process features. During a 2 hours work session, the process owner showed the process model and he was asked about what things ever happen; and about what things must happen. Thereby, process features as *User Requirements Specifications and Validation*, and *System Requirements Specification and Validation* were identified as mandatory parts in any project. Similarly, the process owner was asked about the components of the process not always applied. Thus, the *Exploration* feature was identified as optional. The same approach was applied to each process feature, if the process feature was not a variability, the recursive process stops in this feature, because its detail concern to the process modeling instead of to the process feature identification. In the CC51A-RE process, the process feature *Software Requirements Specification and Validation* includes in turn the *Software Requirements Validation* process feature with two alternatives process features: *Prototype based Validation* and *Internal Validation*, and an optional process feature: *Operational Prototyping*. Further, to ensure that a well composition is achieved, a consistence analysis is conduced. Therefore, dependencies between features must be analyzed. This analysis restructures the defined process feature model because an optional feature could introduce alternative features where one requires the optional process feature but not the other way around. In the CC51A-RE process, in order to include the *Prototype Based Validation* process feature, it is required to select the *Operational Prototyping* process feature. The process engineer has to check the description of processes, activities, work products, tasks, roles, and guidance in order to determine commonalities and variabilities. Each task feature includes the related method elements (roles and work products). The identified commonalities and variabilities must be validated with the process performers related to the specific process features.

- Process Feature Specification: the process model depicted in Figure 4.5 is formally specified as an instance of PFMM. So, a tree of the features is used to characterize a software process. A root type is identified as the whole process and a recursive

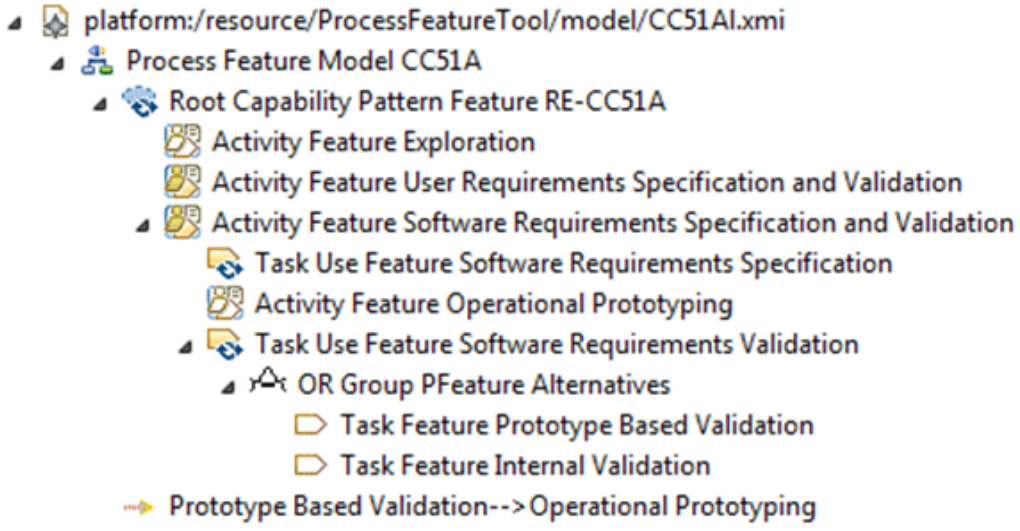

Figure 4.5: Process Features Model of CC51A-RE Process

decomposition strategy is applied. The process features notation is much closer to the implementation language but implementation issues could be relaxed in the modeling.

4.5 Software Process Scope Analysis

Product line scoping determines what is inside and what is outside of the software product line (Clements & Northrop, 2001). This identifies those context entities with which products deal (the product line context), and it also establishes the commonalities and sets limits on the variability of the products in the product line. Getting the scope right is important: if the scope is too large, the core assets will have to accommodate so much variation that they will be too complex to be useful and cost-effective in any product. If the scope is too small, the product line may not have enough opportunities to recover the investment in the core assets and having either a unique process or just a couple of process would be a better strategy. And if the scope bounds the wrong products, the product line will not satisfy the target market. A software process line may result useful when a reference process is involved but some manageable differences between projects are relevant for productivity and quality issues.

Armbrust *et al.* (2008) define Software Process Scoping as the systematic characterization of products, projects, and processes and the subsequent selection of processes and process elements, so that product development and project execution are supported efficiently and process management effort is minimized. Software process optional and

alternative features determine the possible process models. However it is important to determine for each process what the most suitable context is. A specific context configuration could determine a project specific context, but the relationship between processes and configuration contexts is a complex many to many relationship. A set of processes could be defined and associated with a set of context configurations. One specific experience is presented in the Incremental Commitment Life Cycle Process proposed by Boehm (2010) where a set of decision points based on risks are defined. For this model a set of patterns for Rapid-Fielding Projects has been developed (Koolmanojwong & Boehm, 2010b). A pattern is selected according to the situation (suitability). Because each organization is unique, the process model is also unique, so a more general approach than that of Koolmanojwong & Boehm (2010a) is required. Nevertheless, a process or pattern selection is less flexible; process models are limited and complex to maintain. Consequently, when the features tree grows it is critical to produce each process model. In contrast, partial decisions about each feature according to specific situations facilitates producing only one and the most suitable process to this configuration context.

The scoping in a CASPER SPrL represents a preliminary way to define early tailoring rules, such as *selectOptional (ContextAttributeSet, ProcessFeature): Boolean* and *selectAlternative (ContextAttributeSet, ProcessFeature): ProcessFeature* operations, to obtain a global suitable software process model. Lee & Kang (2010) propose a method for scoping SPLs in which contexts of use are related to product features. This method includes three main models: Usage Context Variability Model (UCVM), Quality Attribute Variability Model (QAVM) and Product Feature Model (PFM). The method is based on the mapping between these models: UC-QA mapping and QA-FD mapping. This method is similar to CASPER, however, CASPER is simpler because the Process Features are directly related with the context information using a mapping table or other mechanisms as model mapping or abstract rules among others.

CASPER could be combined with PuLSE-Eco, a method for determining the scope of a product line (Bayer *et al.*, 1999). This method is based on mapping too. First, product candidates are mapped with characteristics about the system domain and stakeholders. Candidates include existing, planned, and potential systems. The result is a list of potential characteristics for products in the product line. Products and characteristics are combined into a product map a kind of product/attribute matrix. In parallel evaluation functions are created, using stakeholder and business goals as input. These evaluation functions allow to predict the costs and benefits of including a particular product with a particular characteristic (such as a feature). Next, potential products are characterized, using product maps and the evaluation functions. Finally, benefit analysis determines the scope of the product line.

4.5.1 Software Process Scope Meta model - SPMM

When the process line scope is defined, the relationship between context attributes and process features could result in a one to one, one to many, many to one, and many to many relationship as explained below:

- *One to one relationship*: is the simplest decision, a process feature variable depends on only one context attribute.
- *One to many relationship*: in this case a context attribute defines many process features. This kind of context attributes are very relevant because certain congruence between the decisions of many related features is required, e.g., features related by the *requires* relationships. In this case many simple rules must be defined for the same context attribute. Maybe these process features correspond to a crosscut software process concern as the one presented by Mishali & Katz (2006) and thus, they could be modeled as a process aspect.
- *Many to one relationship*: this case is very complex. Some context attributes together define the process feature selection. This combination requires an order in a static decision rule and additionally a priority if it is required at tailoring time. In this case it is possible to define when an element must be selected or not selected by one specific combination context attributes. A color code could be used, e.g., coloring green the cross between the involved context attribute value and the process feature if the process feature must be mandatoriy selected for this attribute value, red if the context attribute value implies eliminating a process feature and yellow when a combination of context attribute values of different attributes will define to select or to eliminate a process feature.
- *Many to many relationship*: this case is the most complex. The complexity of the previous case is increased by the complexity of the second case.

According to the above analysis the scope metamodel presented in Figure 4.6 has been built as a set of relationships between process features and context attributes (and its values). A *relationship* ranges from a simple relationship (one to one) to a complex relationship (many to many), and so it could be decomposed into many matches. A *Match* includes links to meta-classes defined in the PFMM and SPCM metamodels. A *Match* has associated a *Context Attribute Value* that belongs to a *Context Attribute*. An *Optional Match* is associated to a Specific *PFeature* whereas a *Selection Match* is associated to a *Group PFeature*. A *Match* can have priority over others. These matches could be used to guarantee a consistent tailoring and to define deterministic tailoring rules.

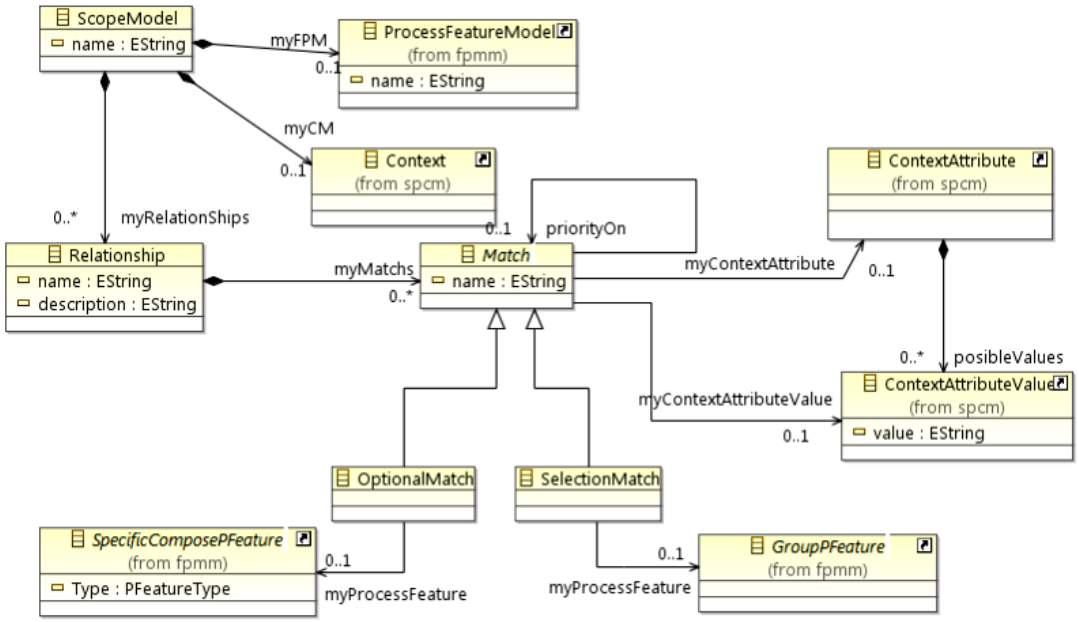

Figure 4.6: Software Process Scope Meta model

4.5.2 Software Process Scope Determination with SPSMM

The scope should be determined based on empirical evidence. Past context projects and their adaptations should be correlated. A frequency table should aid identifying the stronger relationships between process features and context attributes. A simple relationship table could be enough to map process features and context attributes when there are only a few elements to match. However it could be difficult to manage when numerous elements are included in the relationship table. To determine the scope the process engineering should work with the different process users using empirical evidence provided by the project registers and tacit knowledge. The steps could be conducted as follows:

1. Source identification and information collection: past projects, specific contexts and variants of process features are identified. Sources include project documents, process users and processes experts.

2. A simple analysis of correlation could be enough to establish the relationships, but if the match is more complex, a frequency analysis of the correlation between context attributes and process features should be conducted.

3. A consolidation of the strong relationships is obtained. Then, this consolidation is analyzed by the process engineers and process users.

4. Based on the previous analysis, a scope model (match model) is defined.

Context Attributes	Process Features →	Exploration	Operational Prototyping	Software Requirements Validation
Domain Knowledge	High	-	False	Requirements Validation
	Medium	-	False	Requirements Validation
	Low	-	True	Prototype Based Validation
Complexity	High	-	-	-
	Medium	-	-	-
	Low	-	-	-
Project Type	Development	True	-	-
	Extension	False	-	-
	Reingeneering	True	-	-

Figure 4.7: CC51A-RE Process Scoping

For the CC51A-RE process, the scope definition was obtained in a work session with the process owner and the process engineering team. Provided that there are only a few context attributes and a few variants, this session was enough to establish the scope. Figure 4.7 depicts a scope solution for the CC51A-RE process based on a matrix where the rows are context attributes (and values) and the variable features are the columns. In this case only a *one to one* and a *one to many* relationships appear. For example the *project type* attribute determines when the *Exploration* process feature will be selected. The other case is *one to many*, because the *Domain Knowledge* attribute determines if a prototype will be used, i.e., selecting the kind of requirements validation and according to this, the *requires* relationship makes or not mandatory to select to *Prototype Based Validation* task. However, the *Operational Prototyping* process feature in a SPrL evolution could be selected independently from other situations, e.g., when product complexity requires to define a prototype for risk management or verification issues.

4.5.3 SPrL Scope Change

Similar to the SPL Framework of the SEI (Clements & Northrop, 2001), the process scope definition is used during software process tailoring to know if the expected process model is a feasible member of the process line. Sometimes a process clearly falls within the defined scope and others it will be clearly out of scope. However the complex situation is when the new process is located on the boundary. In that case, a scope analysis may help to determine if the organization requires this new process as part of the software process line, and then the scope (and the software process line) can be extended appropriately. If

it is frequent to have processes that fall on-the-boundary, it is perhaps an indication that the scope should be expanded to include these processes.

In CASPER, the scope could be extended when:

1. A new situation is added with a new context attribute or a new context attribute possible value.
2. A new optional/alternative process feature is added.
3. A new constraint on a context attribute-feature relationship is added.
4. A set of old context attributes, process features or a context-feature relationship are modified because a new specific situation must be considered. For example, the *team size* starts to be relevant for selecting the *Operational Prototyping* process feature.

4.6 Implementing Software Process Models Variability with SPEM 2.0 and the CASPER meta-process

In CASPER, process models are defined using SPEM 2.0 (OMG, 2008), that is the OMG standard for process modeling. This thesis actually uses a subset of SPEM 2.0 that is enough for research purposes (eSPEM). SPEM provides some primitives for specifying variability as shown in Figure 4.8. A compliant SPEM complete process model is modeled as a *Method Plug-in* including *Process Elements* and their linked *Method Content Elements*. *Method Content Elements* specifically correspond to *Task Definitions* having *Work Product Definitions* as input and output, and performed (or participate) by *Role Definitions*. An *Activity* is a *Work Breakdown Element* and *Work Definition* that defines basic units of work within a Process as well as a Process itself. An *Activity* supports the nesting and logical grouping of related *Breakdown Elements* forming breakdown structures. The concrete breakdown structure defined in an *Activity* can be reused by another *Activity* via the *used Activity* association which allows the second *Activity* to reuse its complete sub-structure. So, *Role Use, Task Use and Work Product Use* are *Work Breakdown Elements* that refer to activity-specific occurrence of the respective *Method Content Element*.

A *Variability Element* is a SPEM element that can be modified or extended by other *Variability Element* of the same kind according to a *Variability Type* (extends, replace, contributes, extends-replace). To understand the variability types in SPEM refer to its

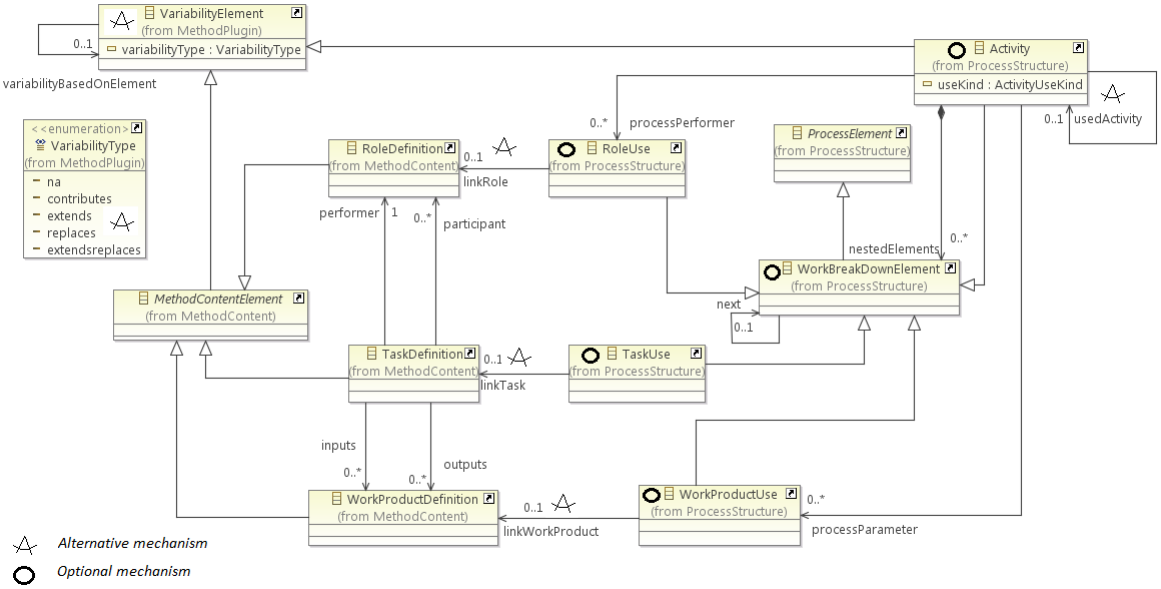

Figure 4.8: Partial view of experimental SPEM (eSPEM) highlighting where variability is specified.

meta model specification (OMG, 2008). Thus, each *Method Content Element* (*TaskDefinition, RoleDefinition and WorkProductDefinition*) and the *Activity* meta-classes are *Variability Elements*.

CASPER could use *Variability Elements* when alternative variants require it (labeled as an alternative symbol used in feature models) and a general condition or constraint must be accomplished. A set of alternatives can be defined from the same *Variability Element* (maybe abstract). When a *Process Element* is linked to the *Variability Element* one of these alternatives could be selected without assembly problems. For example, a *Task Use* can be linked to one of many available *Task Definitions*. Moreover, each *Work Breakdown Element* can be considered as optional or not according to its *isOptional* value.

4.6.1 Evaluating SPEM 2.0 variability mechanisms

In general, SPEM 2.0 offers variation points based on associations to be specified at tailoring time and mechanisms for modeling alternative features when it is required to guarantee the process model integration. An evaluation of the mechanisms offered by SPEM 2.0 for modeling variability in CASPER is presented as follows:

1. *Process Structure Package:* defines the basis for defining process models as a breakdown of nested activities with the related performing roles, as well as input/output work products. It also provides mechanisms for process reuse such as dynamic

binding of process patterns that allows users to assemble processes with sets of dynamically linked activities; CASPER uses this decomposition for defining optional and alternative elements in different decomposition levels.

2. *Method Content Package:* includes concepts for defining process-independent and reusable method content elements that provide a basis of documented knowledge of software development methods, techniques, and concrete realizations of best practices. Method content describes how to achieve fine-grain development goals, by which roles, with which resources and results, independently of the placement of these elements within a specific development life cycle. CASPER uses the relationships between method elements to make optional method element or to select a method element among a set of alternatives.

3. *Process with methods Package:*facilitates integrating processes defined with elements of the *Process Structure* package with instances of the *Method Content* package. Whereas *Method Content* package defines fundamental methods and techniques for software development, processes place these methods and techniques into the context of a life cycle model. CASPER uses method elements as a method content repository to be associated to process structure at tailoring time. A alternative activity is resolved using the *usedActivity* link.

4. *Method Plugin Package:* introduces concepts for designing and managing maintainable, reusable, and configurable libraries of method content and processes. The concepts introduced in this package allow arranging different parts of such a library based on different layers of concern. Using concepts such as *Method Plugin*, *Process Component*, and *Variability Types*, processes with increasing capabilities can be defined. CASPER use this complete variabilities capacity for defining variants on a method library(method content and process structure) to be selected at tailoring time.

4.6.2 Software Process Architectural Model in CASPER with SPEM 2.0

According to Washizaki (2006) a Process Line Architecture - PLA is a process structure which reflects the commonality and variability in a collection of processes that make up a process line from the perspective of overall optimization. The *overall optimization* refers to preparing a PLA with general usefulness rather than defining separate but similar optimized processes. By deriving individual processes from the PLA, the fixed amount of additional effort required in the future can be reduced, and timeliness of completion

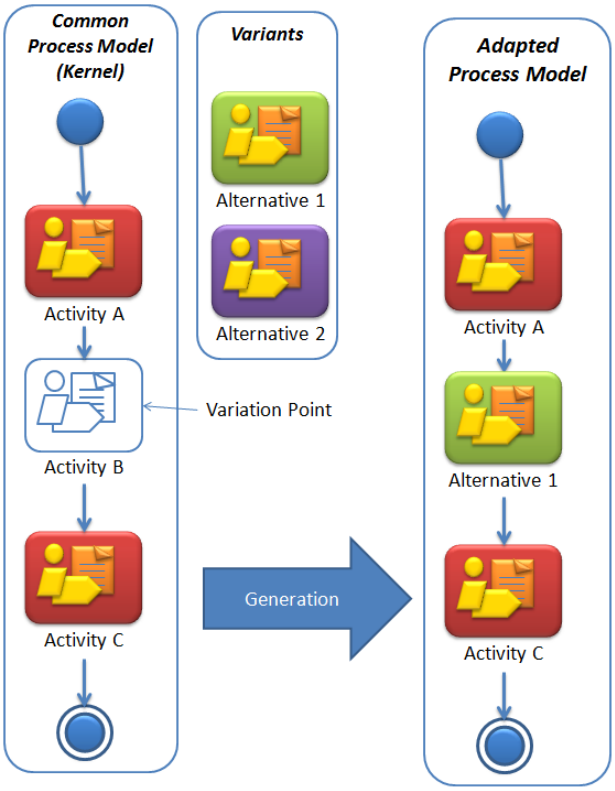

Figure 4.9: An example of variation points and variants

can be improved. CASPER production strategy uses the idea of a SPEM PLA as an organizational (or reference) process where commonality is represented by the core process (kernel process). Variability is represented by the variation points in the kernel process and process variants are available as plug-ins, process components, process patterns, processes, activities, method elements and managed content elements.

SPEM processes in CASPER follow the idea of a Software Process Line Architecture with variation points as presented above which can be changed according to the particularities of a specific project. Process variants corresponding to optional or alternative process or method elements are applied to the variation points as it is exemplified in Figure 4.9 in a model transformation (generation). Processes that are specialized for particular but similar projects can be defined and applied effectively by combining, extending and reusing the core process and variants following a production strategy. When a generative strategy is used, a set of rules relating variation points with project characteristics is required. These rules embody tailoring decisions that could take place to different granularity level. Specific examples follow:

1. A complete process configuration (the template tailoring approach).

2. A complete process pattern could be selected to one discipline or an iteration.

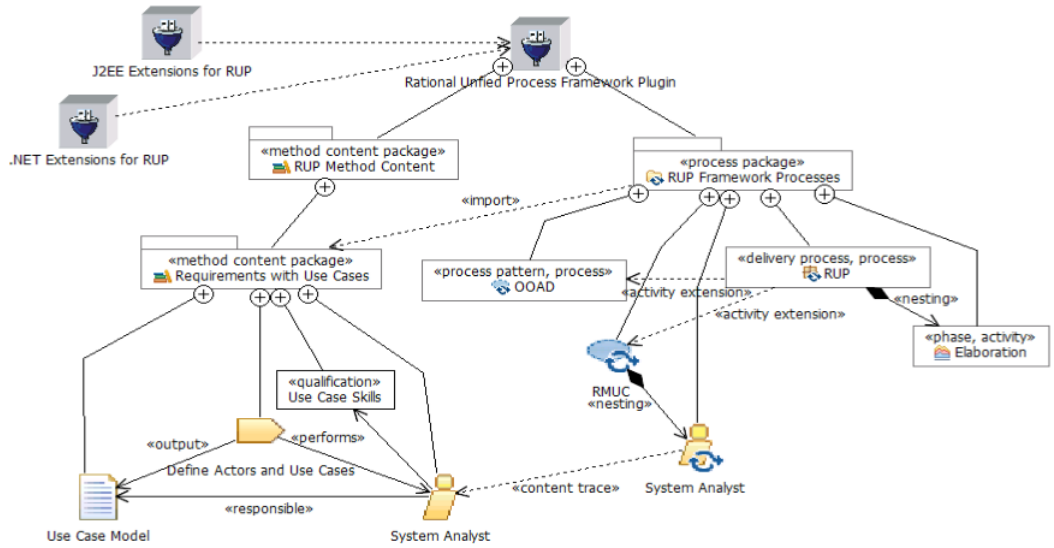

Figure 4.10: A partial view of the RUP architecture in SPEM2.0 taken from SPEM2.0 Specification (OMG, 2008)

3. A complete set of document templates according to what is rigorously required.

4. A complete method plug-in could be selected because specific technology methods or processes are required.

5. A complete method/process package or process component could be selected because a third party's specific interface is required.

6. A complete phase could be selected according to a management criterion.

7. A medium-size activity could be selected according to a technical criterion.

8. A set of tasks, work products or roles could be selected according to a technical/management criterion.

To different granularity levels, variation point modeling depends on the feature models defined in the domain analysis, as the software process model architecture is and the constructs of the used software process meta model. Thus, a process feature could be implemented by a unique element (e.g. a *task definition*), by a set of elements (e.g. a set of *Templates*), a process pattern or an activity among others as depicted in Figure 4.10 where the Rational Unified Process is sketched. The SPEM 2.0 architecture itself provides a framework for structuring a software process model. The basic steps for structuring a process supporting variability are:

- *Step 1.* Create a *Method Library*. A *Method Library* is the complete container of the organizational software process model and the adapted software process models.

- *Step 2.* Identify common (e.g. RUP *Method Plugin*), optional and alternative (e.g. *J2EE Extensions for RUP*) method plug-ins. Create and reuse the respective method plug-ins. Relate the process features to the respective method plug-ins. Define the variation points in the kernel process and their associations to variant method plug-ins.

- *Step 3.* For each *Method Plug-in* create a *Method Content Package* (e.g. Requirements with use cases *Method Content Package* in RUP). Structure each *Method Content Package* according to the corresponding design decisions (e.g. RUP *Method Content Packages* corresponds to *Disciplines*). Create and identify the *Method Elements*. Relate the *Process Features* to their respective *Method Elements* identifying variants. For a set of variants of a variation point, a hierarchy of *Method Elements* could be defined using the respective types of the variability types defined in SPEM 2.0 as depicted in Figure 4.11 where a set of alternative *Tasks* has been defined as a hierarchy of *Tasks*.

- *Step 4.* For each *Method Plug-in* create the respective process packages (e.g. RUP process framework package). Structure each process package according to the corresponding design decisions. Create and identify process patterns (e.g. a *Process Pattern* is defined by each discipline such as OOAD *Process Pattern*) and *Delivery Processes* (e.g. RUP *Delivery Process*) both core and variants. Relate the process features to the respective process elements (including *process patterns*). For each alternative feature that has previously been identified to be part of an *Activity*, a hierarchy of activity elements could be created using the respective types of the variability types defined in SPEM 2.0. When an alternative feature has been identified in step 3 to a *Method Element in Use* then the element with the variation point must be set to a default method Content Element as shows Figure 4.11. This link considers the variation point to be fixed at tailoring time. Similarly, the *usedActivity* link of an *Activity* to another *Activity* could be considered as an alternative (similar to Figure 4.9) variation point (extended activity in SPEM2.0). In this case this link must be set to a default *Activity* and fixed at tailoring time to the corresponding *Activity* identified as a variant.

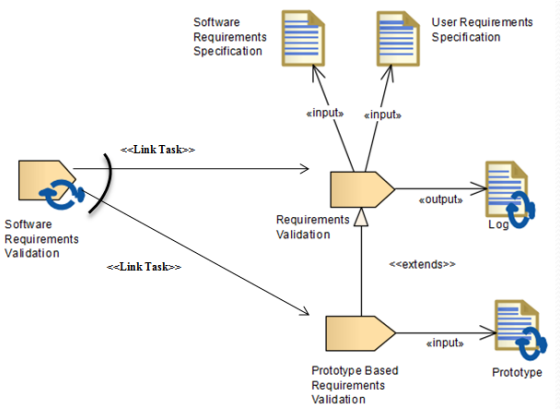

Figure 4.11: Implementing an alternative task with a hierarchy of tasks and the task link as a variation point

4.6.3 General Requirements Engineering Process

In the general requirements engineering process presented so far, three main process components have been identified: *Exploration*, *User Requirements Specification and Validation*, and *Software Requirements Specification and Validation*, as shown in Figure 4.12. A complete view of the CC51A-RE Process Model as an instance of eSPEM is presented in Figure 4.14 where a method library, a method plugin and their respective elements (*Packages, Patterns, Process Elements and Method Elements*) have been defined. It is a simple process because it is related to real small software companies that typically develop small software projects and they have little resources to conduct sophisticated RE activities (Vergara, 2008).

The *Exploration* activity is in charge of identifying the business problem to be addressed and the project context. Typically, clients, analysts and the project manager participate in this activity. The *Exploration* activity does not have formal inputs; however it produces a detailed problem and project context specification. Exploration activity matches the Exploration Activity which has been defined as optional (shadowed in Figure 4.12). This activity could be either included or not at tailoring time. The *User Requirements Specification and Validation* activity is in charge of determining the project's goals and scope in terms of user requirements. Performing this activity may involve the use of simple prototypes that help clients and team members to validate the user requirements and conceive the high-level solution. The activity input is the problem definition and the output is the user requirements specification.

In the last stage of the RE process, *Software Requirements Specification and Validation* activity, each user requirement is translated into one or more software requirements. The resulting list represents the software requirements specification for the project. Provided

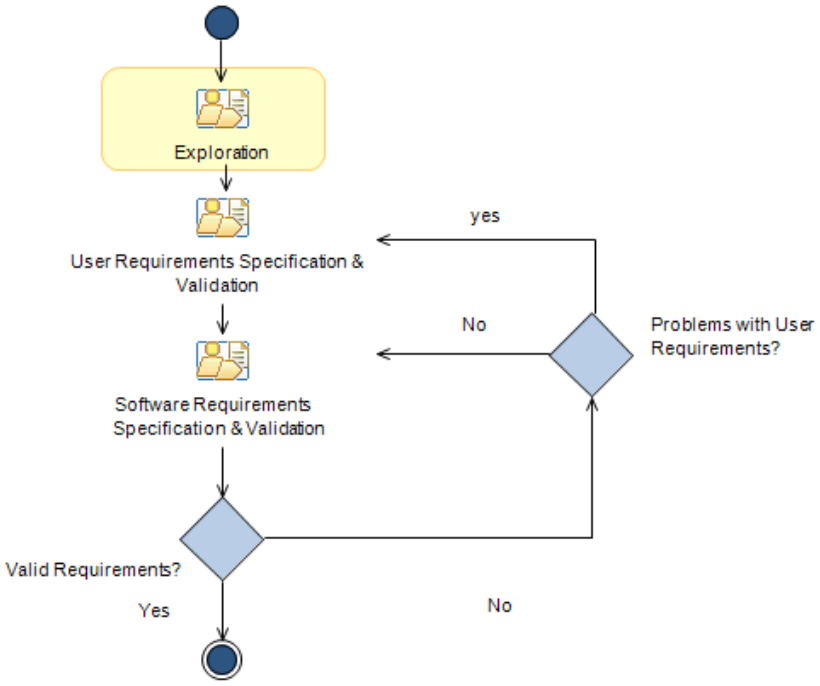

Figure 4.12: Requirements Engineering Process

that this translation is done by members of the development team, a validation with users and clients is required. Such a validation is carried out in two steps and it may involve operational prototypes as is shown in Figure 4.13 (this variation point was implemented as Figure 4.11 shows). One option is to have an internal validation in which just team members participate. Its goal is to check if the translation is good enough and also to determine if the resulting requirements list is addressable by the development team.

The first alternative involves using an operational prototype to validate the requirements translation with users and clients. In the *Software Requirements Specification and Validation* activity, analysts, designers and eventually some programmers participate in the translation process. Once the software requirements specification is approved by the development team, an operational prototype is implemented in order to validate the translation with clients and users. This second validation process typically produces feedback that is recorded and used to refine software requirements and also to adjust the prototype. This is an iterative activity that concludes when the translation of all priority user requirements have been validated; therefore the resulting product is the validated software requirements specification.

On the other hand, if the development team is familiar with the application domain, the *Software Requirements Specification Validation* activity is carried out internally without the user intervention, and without the need of an operational prototype. However, as

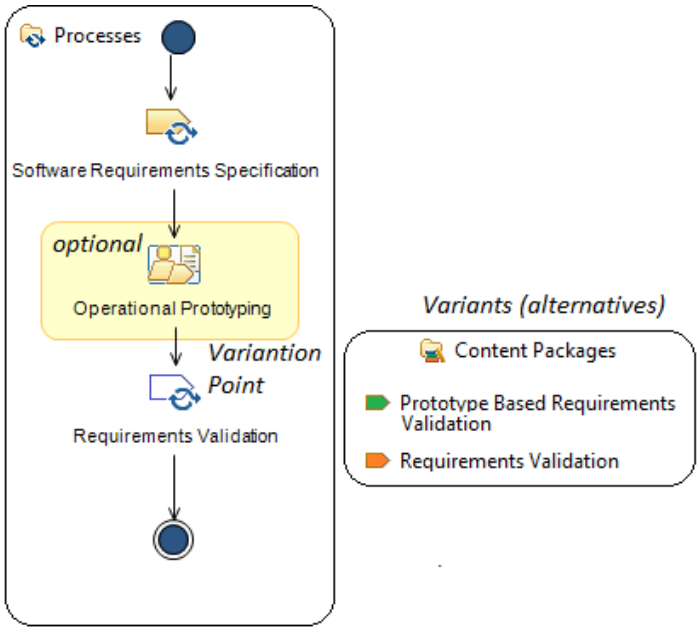

Figure 4.13: Software Requirements Specification and Validation Alternatives

validation is required, it is realized anyway internally by the team members who are in charge of checking and refining each requirement translation through an iterative process.

4.7 A MDE production strategy of CASPER

CASPER follows an approach for automatically tailoring organizational processes to particular project contexts, based on MDE techniques so that appropriate processes are achieved rapidly and with little effort. Tailoring is implemented by means of a model transformation whose inputs are the organizational process model including variabilities and a model of the project context, and whose output is the context-adapted process. Previously the meta models were formalized and this section shows how transformation rules could be defined and implemented using ATL (Atlas Transformation Language).

4.7.1 Environment Implementation Definition

The CASPER tool[1] implementation was developed in Eclipse Modeling Framework - EMF 3.4[2] and the ATL plug-in 2.0[3].

[1]CASPER tool website https://sites.google.com/site/softwaremetaprocess/
[2]EMF website http://download.eclipse.org/tools/emf
[3]ATL website http://www.eclipse.org/downloads/

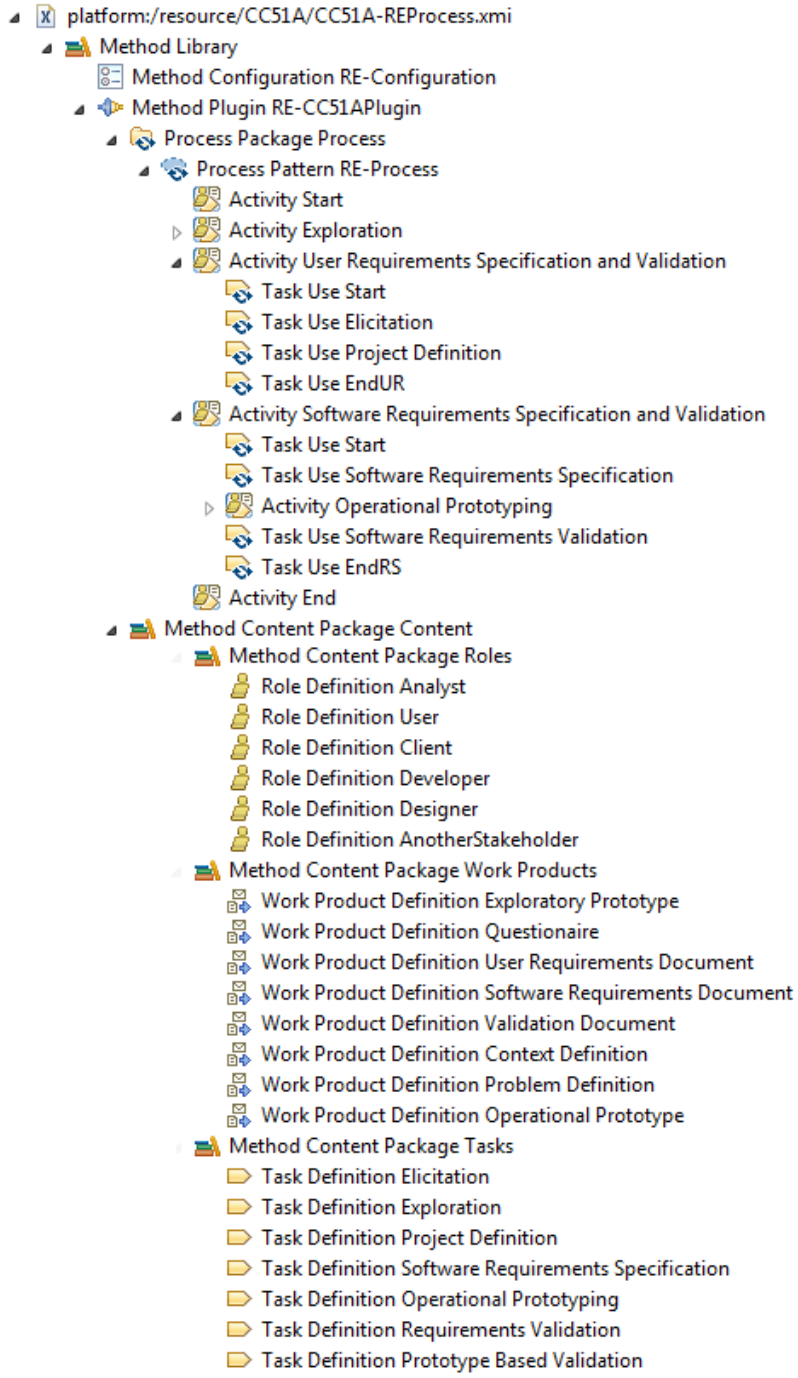

Figure 4.14: Requirements Engineering Process as an instance of eSPEM

4.7.1.1 Modeling Platform

The EMF project is a modeling framework and code generation facility for building tools and other applications based on a structured data model (Budinsky *et al.*, 2003). EMF is based on two meta-models: the Ecore and the Genmodel meta models. The Ecore meta model contains the information about the defined classes. The Genmodel contains

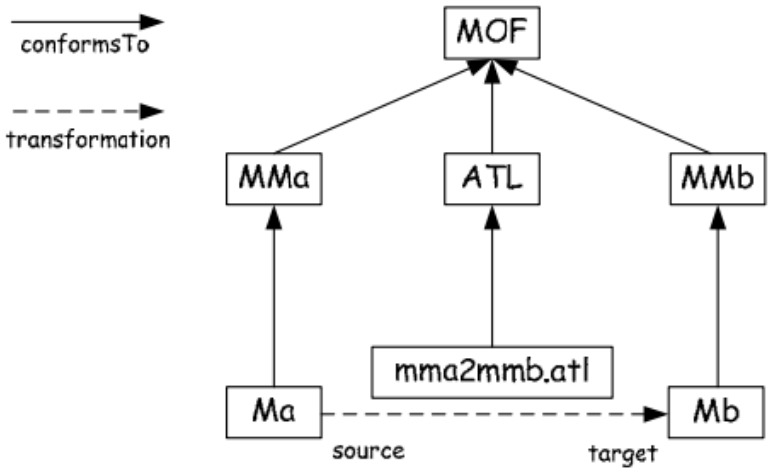

Figure 4.15: ATL Transformation Approach

additional information for the code generation and the control parameter about how the coding should be generated. The Ecore model shows a root object representing the whole model. This model has children which represent the packages, whose children represent the classes, while the children of the classes represent the attributes of these classes.

Meta models, eSPEM and SCPM were defined as Ecore meta models in EMF and the transformations were implemented as ATL rules. Models were implemented as instances of defined meta models and edited using Exeed (Extended EMF Editor) an extended version of the built-in EMF reflective editor that enables customization of labels and icons by adding annotations to Ecore meta models.

4.7.1.2 Model Transformation Language

ATL is applied in the context of the transformation pattern shown in Figure 4.15. In this pattern a source model Ma is transformed into a target model Mb according to a transformation definition mma2mmb.atl written in the ATL language. The transformation definition is a model conforming to the ATL meta model, while all meta models conform to the MOF.

ATL is a hybrid transformation language: it mixes declarative and imperative constructs (Jouault *et al.*, 2008). Declarative is the main style of specifying transformations. However, sometimes is difficult to provide a complete declarative solution for a given transformational problem, this is the case of tailoring rules. In that case some imperative features of ATL are required. ATL transformations are unidirectional, operating on read-only source models and producing write-only target models. During the execution of a transformation the source model may be navigated but changes are not allowed to it. The target model cannot be navigated, it is a limitation when SPEM elements of the

target model require to be linked after or before a transformation rule, e.g. in a *WorkBreakDownElement* when the *nextElement* is optional and it is not selected. Source and target models for ATL are expressed in the XMI OMG serialization format using Ecore.

An ATL transformation can be decomposed into three parts: a header, helpers and rules. The header is used to declare general information such as the module name, the input and output meta model and potential import of needed libraries. Rules are the heart of ATL transformations because they describe how output elements (based on the output meta model) are produced from input elements (based on the input meta model). The ATL rules express a mapping between an input element and an output element. In CASPER, because the transformation produces a delta of the input, the rules basically are copiers with small changes at the variation points, and the helpers are called to fix the variants according to a specific context configuration when a variation point is found. Variation points are located on rules whereas variants are recovery from helpers. Helpers functions are called from rules to fix the selected variant.

4.7.2 MDE Software process tailoring

Adaptation in CASPER is the process to resolve variation points to specific variants according to a context configuration (specific situation). The SPrL scope model defined above is the main input to the transformation implementation, because all the process features have been related to context attributes (as preliminary tailoring rules). In CASPER, the adaptation ocurrs during the domain engineering following a planed approach, where the project manager should only provide the project characterization at hand, and a project-adapted process is rapidly and automatically generated. In CASPER the tailoring decisions are encapsulated as rules. Thus, rules about tailoring the general process model according to the context attributes can be composed incrementally. In this way we can configure new process models through a generative strategy by recombining partial tailoring transformation rules, and thus reusing the knowledge they embody. All and only the required roles, activities and work products will therefore be present in the adapted process, and no extra work would be required. The adapted process is thus more efficient, and the tailoring process is more reliable as well, since it is computed automatically.

4.7.3 Defining transformation rules

Each variation point identified and associated with a process feature requires one o more simple or complex rules. To determine a decision, the scope table aids to identify when a variant (optional or alternative) is selected. To set an optional variation point requires a rule returning a *Boolean* value, whereas to set an alternative variation point requires a

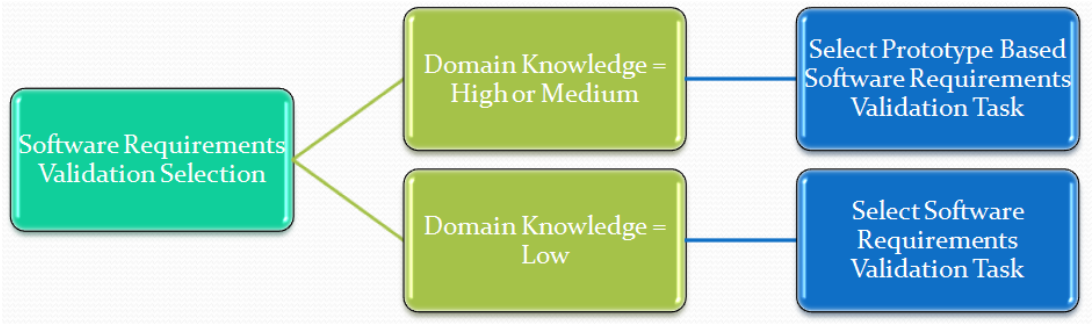

Figure 4.16: Select Rule defined as Tree Decision

rule returning a *Kind* value(according to the Kind defined in the selection). For instance, a Task alternative variation point requires a rule returning a *Task* value and a Activity variation point requires a rule returning a *Activity* value. In the case of the CC51A-RE Process the select rule shown in Figure 4.16 is defined as a decision tree from the scope definition table presented early in Figure 4.7. A Tree decision structure has been selected for simplicity and scalability reasons.

4.7.4 Implementing transformation rules

The ATL tailoring transformation developed is endogenous (Czarnecki & Helsen, 2006) because its output conforms to the same meta model as the input, eSPEM. However, it is not *in place* since we want to preserve the organizational process model for future transformations. CASPER uses *ATLCopier*[1] for duplicating the general process, while rules executed modify the copy according to the values in the context model. Matched rules constitute the core of an ATL declarative transformation since they allow us to specify: (i) which target elements should be generated for each source element, and (ii) how generated elements are initialized from the matched source elements. In CASPER tailoring rules make decisions for identified variation points in the process model. Each variation point has an associated helper called from the matched rule. Figure 4.17 shows rule *TaskUse*. The source pattern *MM!TaskUse* is defined after the keyword *from*, meaning that the rule will generate target elements for each source element matching the pattern. For instance, in order to select only those source elements that are relevant for the specific project, an extra condition is added: an optionality rule implemented as a *helper function*. When this rule returns false, the element needs to be removed from the process, so the copier rule is not applied to this element. Attribute initialization uses the values in the source process model element. However, and provided that we use eSPEM variability mechanisms, a process element (e.g. *TaskUse*) could be linked to several variants of method elements

[1]ATL Transformation Zoo. `http://www.eclipse.org/m2m/atl/atlTransformations/`

(e.g. *Task Definition*). Therefore, we define an *AlternativeTailoringRule* as a rule that returns the selected method element according to the helper rule. The *AlternativeTailoringRule* chooses the most suitable *TaskDefinition* variant, according to the *Domain Knowledge* Value in the context. If there were more variability points, a combination of rules would be applied. Additionally if priorities to make trade-offs were required the respective helpers would be used.

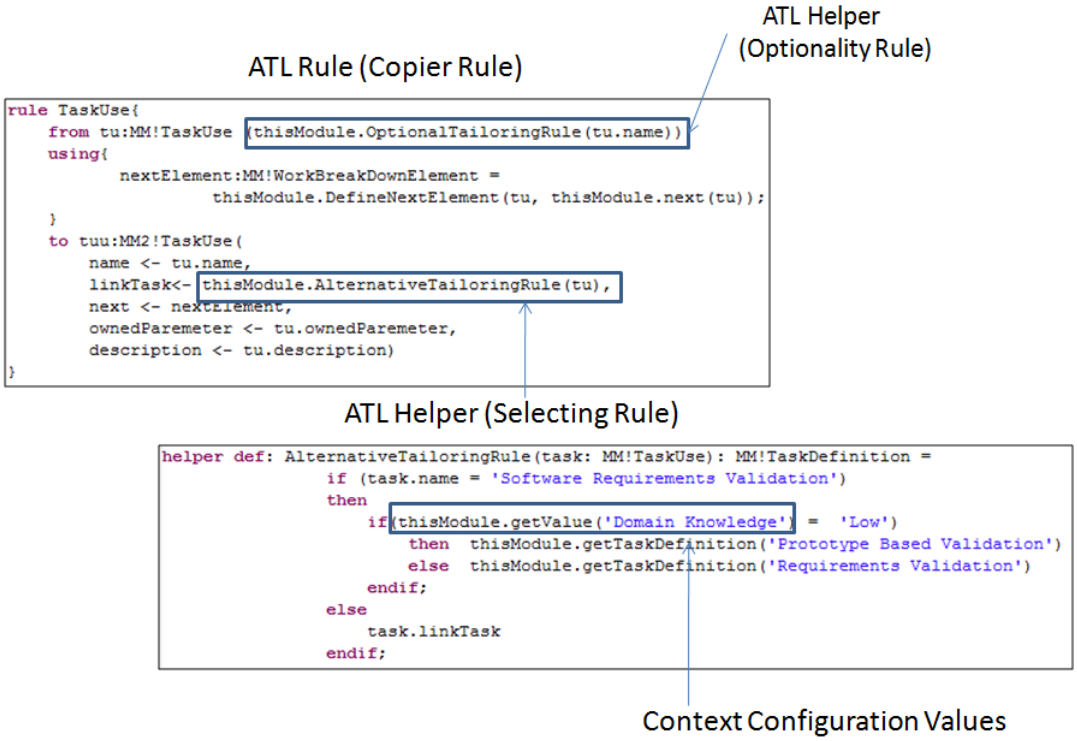

Figure 4.17: ATL Tailoring Transformation

4.8 CASPER application engineering: Context-Based Software Process Adaptation

Tailoring in CASPER requires to understand the specific project context and to define a suitable context configuration. The contextual framework was defined by the general context model during the domain engineering activities. Each attribute must be fixed to a specific *Context Attribute Value*. Therefore, new attributes will not be taken into account because they would be out of the scope. However, an approximated context configuration could be defined and the resultant process model could be tailored manually (delta). In this case the new process manually defined is a possible input to a new

domain engineering cycle. Also, the tailoring could be applied many times according to the available knowledge of the project context while the project progresses.

4.8.1 Generating context-adapted process models

In the SPMC meta model, the *Context Configuration* is key to define a specific configuration of the process model. Thus, a *context configuration* is a set of *Context Attribute Configurations* where the value is fixed during project planning when the process normally is tailored and enacted. At that time, the project context information is available and it could be refined as the project progresses if a new process tailoring is required. In an iterative way the project manager could execute the following steps:

- *Step 1.* Analyze each dimension defining the conditions of the actual project. This analysis depends on the context dimensions. In this step it is very important to have a suitable idea about the real context. For instance, the project manager could elicit requirements from different stakeholders. However, the information could be incomplete (but planning will be incomplete too anyway). In any case when planning is required, tailoring must be executed with the available information at hand. This case is different from the case where the context model is insufficient to express the context for a new project.

- *Step 2.* Determine for each attribute the specific value in the context configuration. In this case the project manager chooses an alternative of the possible attribute values.

- *Step 3.* Generate the process model running the ATL transformation with the context configuration and the organizational process model. Use the resulting process model to enact it in the project.

Tailoring transformation execution of *CC51A-RE Process* on the context defined in Figure 4.18 is shown in Figure 4.19. The context model configuration corresponds to an *extension project* for a *low domain knowledge*. We can see that the *Exploration* activity is not included as part of the process and the *Operational Prototyping* activity was included, and therefore *Software Requirements Validation* task in use is realized through its *Prototype-Based Validation* task option. This result matches what was expected. All others configurations were used as well and the adapted processes generated matched what was expected too.

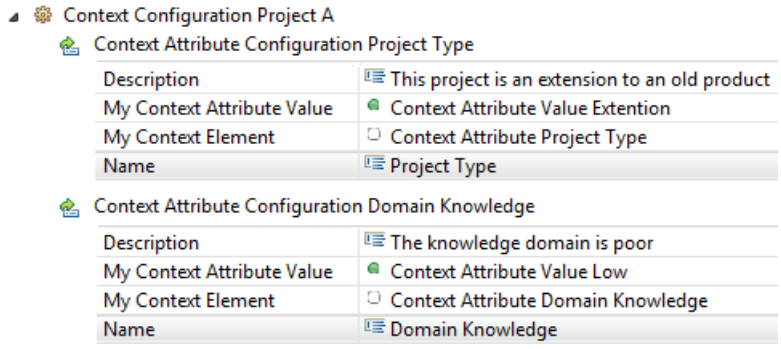

Figure 4.18: A Context configuration to an user project of CC51A-RE Process

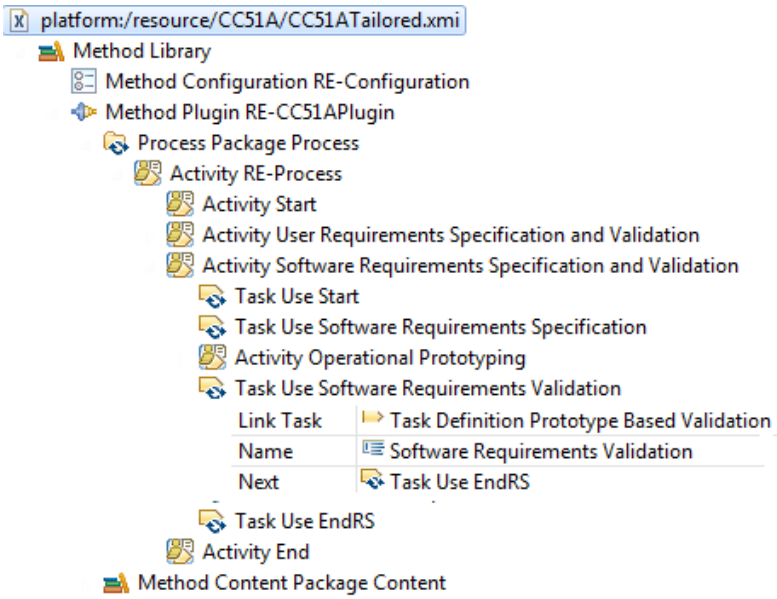

Figure 4.19: An adapted software process model of CC51A-RE

4.8.2 Manually Tailoring

When the adaptation is out of scope (the specific project context cannot be modeled with the defined context model), a non-member of the SPrL could be manually created. If the new situation will happen frequently in the future, the SPrL scope could be extended. A scope analysis could be realized to determine if the new process could be a member or not of the SPrL, if a new member must be included then a new domain engineering cycle should be executed. Furthermore, this scenario could be the starting point to create a SPrL. Therefore, a set of the manually created members are transformed into a SPrL. In this case, according to Washizaki (2006) the process architecture should be recovered and in the same way the specific context should be recovered from past specific situations to define the process context of the SPrL.

4.9 Preliminary Validation

Before applying the proposed approach to an industrial case, a preliminary validation was conduced to confirm its consistency, feasibility, and applicability. This preliminary validation was presented in this Chapter where the CASPER prototype was presented and a small academic case study was used as an example. The academic case study refers to a requirements engineering process used as part of a software engineering course CC51A. The following subsections provide details on the particular forms of validation and present the obtained results.

4.9.1 CASPER Tool Prototype

As was explained above, CASPER tool is based on the definition of the CASPER meta-process and its MDE strategy. A successful implementation of the meta models and transformation rules support for CASPER demonstrates that the meta-method is applicable (in particular the production strategy) and it has achieved a suitable level of definition, consistency and determinism. CASPER tool was used to validate CASPER in academic and real world case studies.

4.9.2 CASPER Academic Case Study

CASPER was initially used in an academic context. The partial process model of the course Software Engineering II (CC51A) at the University of Chile was used as a small initial but complete case to validate the suitability of the whole approach. The case only includes the requirements engineering process. This case has been presented as example to introduce CASPER in this Chapter. According to the case description, the *CC51A process* model consists of 2 optional elements, 1 variation point, 2 variants and 1 *requires* constraint producing 6 different processes. Additionally, the context model covered two dimensions with three context attributes and 9 possible attribute values, resulting in 27 possible contexts. The ratio between contexts (27) and process models (6) was of a 22.2 (6/27) percent (with a unique process model would be 3.7(1/27) percent and in a template approach with three processes a 11.1 (3/27) percent). However, the solution could give better results if the requirements process is improved with some requirements practices introducing rich variability. This example shows that the approach is suitable for the problem; consequently, the tailoring rules are reusable, and the tailoring is significantly less time-consuming and simpler than a configuration based approach. Because the manual intervention is not possible, the process is not human error prone. Still, the human error could happen when the context configuration model is defined, but in this case the error

does not correspond to the tailoring process itself but to the way how the project context is established. Although this is an interesting and valid problem in software process modeling, it is a problem out of the scope of this thesis.

A relevant question arises about the effort involved in tailoring. In Casper the tailoring effort is very low, however the domain engineering effort cannot be ignored. Thus, the domain engineering effort (to build software process line models) should be distributed among the possible instances of the process line (at least among used instances), and also the context specific modeling effort. Even so, in this academic case, the effort is lower than a reactive approach because the setup effort is divided among 6 processes. In the Casper validation in Chapter 6 the effort is analyzed in detail.

4.10 Synthesis and Discussion

Casper proposes an MDE-based meta-process for automatically generating processes by tailoring a general process applying a set of transformation rules defined during the definition of the organizational process as a process line. Casper has the potential to enable improvement with respect to project productivity and quality of the resulting software products. Provided that the adapted process will include all process elements that are required for the particular project context, no extra work will be needed and only the essentially required effort and resources will be spent. In addition, high quality work products can be expected, because the process is adjusted with this goal in each particular project context. Since this tailoring process is automatic, and it eventually uses already validated transformations, it is expected to achieve a reduction of the tuning time and cost, and also the number of adaptation errors. The case study developed in this chapter shows how the approach works and shows that it is possible to apply tailoring transformations built for adapting a general RE process to different project contexts in a planned manner. An industrial case will be presented later in Chapter 6. Being able to validate the transformations for particular known cases gives us confidence on their validity for the general case. Therefore, whenever unanticipated scenarios happen, a combination of already built (and potentially already validated as well) tailoring transformations can be applied; and as a consequence, an appropriate context adapted processes can be obtained quickly and easily. The experience has allowed us to conclude that: (1) Casper techniques are an effective tool to achieve process tailoring; (2) the Casper approach is useful and practical because it was easily implementable in the MaTE Laboratory. However (3) the prototypical tool still needs be more usable, in particularly to define the transformation rules.

Chapter 5

Software Process Models Analysis and Visualization

5.1 Introduction

Software process models are sophisticated and large specifications aimed at organizing and managing software development. An approach as CASPER in particular requires formalized software process models. Their formal specification demands an enormous effort, but once specified there are few approaches and even fewer tools that aid the process engineer to analyze the quality of the process. For the last five years as part of the Tutelkan project, ten Chilean software companies were assisted in specifying their software processes. A serie of problems that indicate the potential presence of misconceptions or misspecifications in their process models were found.

As a first step, process model blueprints (Hurtado *et al.*, 2010b) were developed as a means for visualizing and analyzing different perspectives of a software process model. The three blueprints considered (ROLE BLUEPRINT, TASK BLUEPRINT, and WORK PRODUCT BLUEPRINT) were applied to software process models specified with SPEM 2.0 OMG (2008). The software process blueprints enable the identification of *exceptional entities* (Demeyer *et al.*, 2002), *i.e.,* exceptions in the quantitative data collected. Software process blueprints were successfully used to identify a number of flaws in an industrial process model, but a lot of experience from the process engineer was required for identifying these flaws. Since then, several other industrial process models were assessed, and a set of *recurrent patterns* ranging from suboptimal modeling to misconceptions and misspecifications were discovered. The next step was to characterize kinds of process modeling errors as error patterns, and to detail how errors could be localized within a software process model. To assist process engineers to analyze the quality of their processes AVISPA was developed. AVISPA is a tool that graphically renders different aspects of a process

model and highlights potential errors as intuitive and comprehensible indicators. This chapter presents both, software process blueprints and AVISPA tool[1]. The approach and the supporting tool are illustrated by applying them in two software process models.

5.2 Software Process Blueprints

There is a generalized agreement among software practitioners about the relevance of counting on a well defined process model for systematizing development. There have been several efforts in aiding organizations toward this goal: maturity models and standards that describe which elements should be part of a software process, notations that allow rigorous specification, and tools that support these notations. However, having a well defined software process does not necessarily mean having a good process. It is not apparent how to determine if a software process is good, or if it can be improved in any sense before it is enacted (Osterweil, 1987).

The convenience of specifying the software architecture with multiple views has been agreed upon (Clements *et al.*, 2002). Different stakeholders require different information for evaluations, thus it seems natural to deal with multiple process architectural views (Jacobs & Marlin, 1995). Process architectural views are built following an architectural recovery process such that they allow us to evaluate different process characteristics. In this thesis, three blueprints were proposed to analyze processes: ROLE BLUEPRINT, TASK BLUEPRINT and WORK PRODUCT BLUEPRINT. These blueprints are process architectural view types that make use of metrics computed from a process model for understanding and evaluating different aspects not directly available in SPEM 2.0 and its associated tools. These visualizations have been implemented in the ProcessModel tool[2] based on Moose technology and the Mondrian tool. Visualization aids in data analysis and problem detection (Lee *et al.*, 2003). Using different concerns about process models in a graphical manner, the human visual capacity is used for interpreting the process in context and to evaluate its coherence and suitability (Larkin & Simon, 1987). The approach has been applied to the process model specification of DTS, a real world medium size company that develops software for retail. Process model blueprints have been able to identify several problems in the process and the company's stakeholders agreed that they were actual problems. The process has been improved based on the problems found. This subsection presents the software process blueprints using as example the process of DTS.

[1] Analysis and VIsualization in Software Process Assessment

[2] Freely available at http://www.moosetechnology.org/tools/ProcessModel.

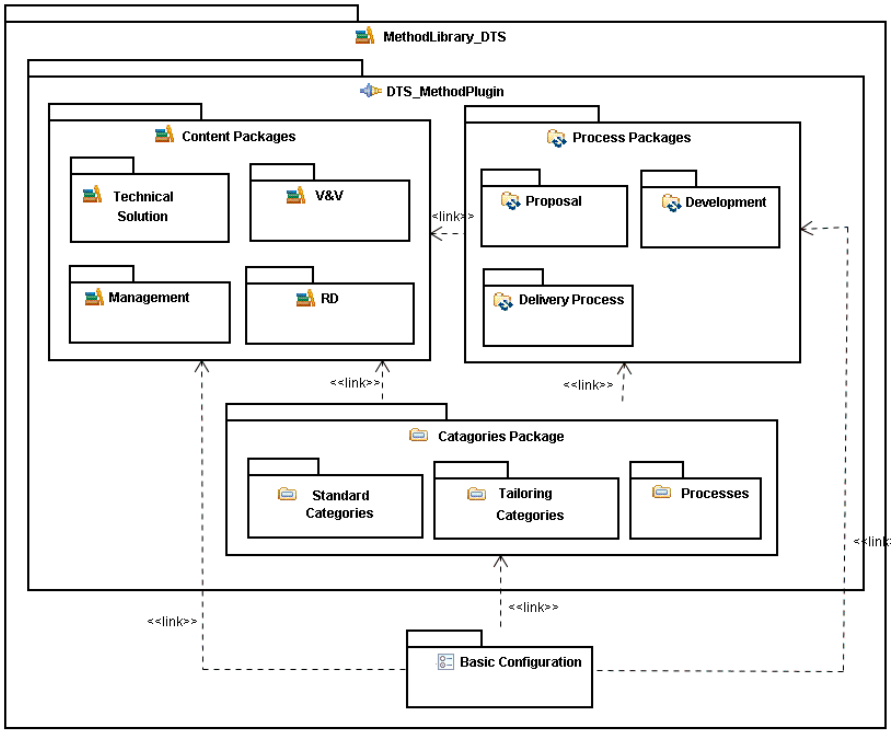

Figure 5.1: DTS Organizational Process

5.2.1 Example Process: DTS Process

System and Technology Development - DTS is a company that started business around 1990 from a joint venture between a Chilean Aeronautic company, EANER and ELTA Electronics Industries. DTS works in solutions for military and civil technology. It has around 250 employees, including engineers, certified technicians, operation workers and managers. DTS started to define its software process model in 2008, using the Unified Process as a reference process. In DTS there is no specific software process improvement project; its effort has been oriented toward recovering the software process actually applied in the organization, in order to formalize it, analyze it, and eventually improve it.

Figure 5.1 shows the organizational process of DTS specified with SPEM 2.0. The depicted organizational software process defines:

• A method library called MethodLibrary_DTS that defines the organizational knowledge and the mechanisms for applying the software process.

• A method plug in called DTS_MethodPlugin that facilitates the management of reusable and configurable software processes and method content.

• Four Content Packages that encapsulate knowledge about development methods, techniques, and concrete realizations of best practices on areas relevant for the organization: technical solution, validation and verification, management, and requirements

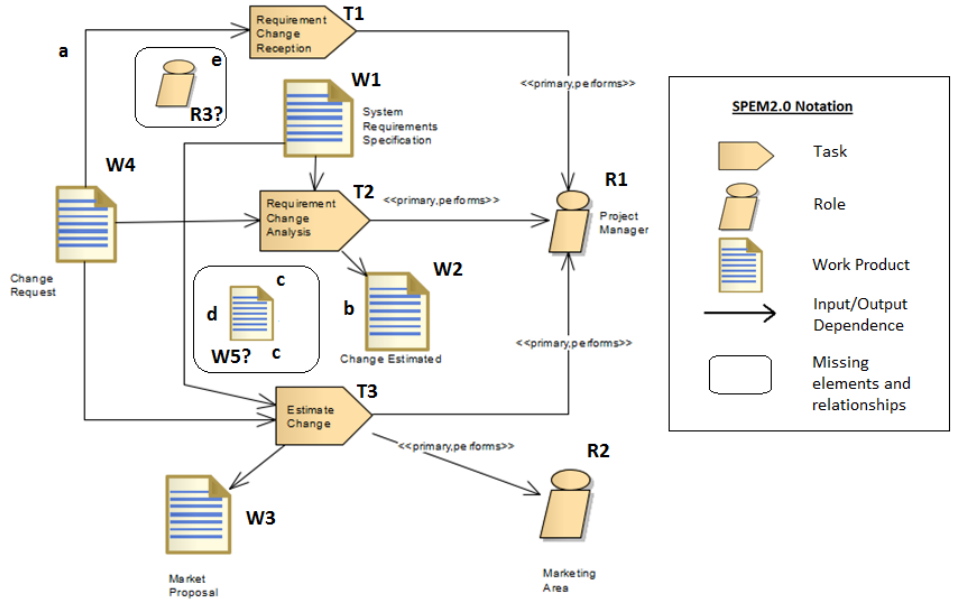

Figure 5.2: SPEM Model Fragment for Requirements Change Management

development.

• Three Process Packages that define the main process structures. Proposal and Development define the process patterns for pre-development breakdown structure (proposal) and development breakdown structure, respectively.

• Three Category Packages that define the main categories used for visual deployment of the process (standard categories and processes) and for grouping elements sensible to a context attribute or a tailoring rule (Tailoring Categories).

• A Method Configuration that defines a basic configuration for the most typical projects in the organization. This basic configuration is used for defining new configurations tailored to specific contexts.

5.2.2 Problems in Software Process Model Analysis

Defining a software development process is an effort to systematize and improve software development. However, defining a software process does not necessarily imply that the process is complete, sound and/or well defined.

Enterprise Architect (EA)[1] and Eclipse Process Framework (EPF)[2] are two popular integrated development environments (IDE) used to describe and visualize SPEM processes. Hurtado *et al.* (2010b) argue that the representations provided by these tools are not enough for easily assessing model quality.

[1] http://www.sparxsystems.com.au
[2] http://www.eclipse.org/epf

The software process model used by DTS is composed of 57 tasks, 66 work products and 9 roles. The complete process has been considered, including project management, testing, requirement management and technical development. Figure 5.2 is a screenshot of a piece of the DTS process model. This model excerpt focuses on some tasks of the requirements change management area. The figure represents the main dependencies between tasks and their related roles and work products.

The process model adopted at DTS received a careful attention at every step of its conception by dedicated engineers. However, a number of problems were recently detected to assist but not easily localized.

Incorrect relationships (a,b): W4 (*Change Request*) should be an output of T1 (*Requirement Change Reception*) instead of being an input, and W2 (*Change Estimated*) should be an output of T3 (*Estimate Change*) instead of T2 (*Requirement Change Analysis*). The change request is received by task *Requirement Change Reception* and is used by all other tasks. The *Change Estimated* work product is the result of the *Estimate Change* task.

Missing elements (d,e): W5 (*Requirement Change Analysis*) is required as output of T2 (*Requirement Change Analysis*) and input of T3 (*Estimate Change*). A new role R3 (*Client*) is required because it is involved in T1 and T2 (*Requirements Change Reception and Analysis*). W5 and R3 are missing because the process has been inadequately designed.

Missing relationships (c): control links between work product W5 (*Requirement Change Analysis*) to task T2 (*Requirement Change Analysis*) as output and to task T3 (*Estimate Change*) as input are missing. Without W5 between T2 and T3, the information flow remains underspecified.

These anomalies in the DTS process were not easily identified mainly due to scalability, complexity and availability issues. Figure 5.2 depicts only a small portion of the complete process and Figure 5.1 depicts the general structure. The complete picture is multi screen which makes it difficult to analyze. Many different sources of information are gathered in the picture obtained from EA. As a consequence, checking the validity of the model is often perceived as a tedious activity if possible at all. A global picture can be obtained through a simple composition within the environment, but this composition is not automatically built in a global fashion by SPEM modeling tools.

Although the analysis has been directed on EA and EPF, the same actions could have been made in other tools (including SPEM as a UML Profile). To tackle these issues, a number of extra visualizations could be used during the process designer's activities. Simulation is not a practical option in this specific case because SPEM 2.0 neither provides concepts nor formalisms for executing process models (Bendraou *et al.*, 2009).

5.2.3 Multiple Software Process Model Blueprints

There is no unique perfect view to visually render a software process model (Jacobs & Marlin, 1996). As for most engineering activities, defining a robust, efficient and multi-view software process is a non-trivial activity that requires flexible and expressive tools. As a complement to software process design tools, a visual approach has been proposed to evaluate some quality criteria. Therefore, a software process model blueprint is a partial but focused graphical representation of a software process model.

5.2.3.1 Process Model Blueprints in a Nutshell

Process model blueprints are graphical representations meant to help software process designers to (i) assess the quality of software process models and (ii) identify anomalies in a model and provide hints on how to fix them. The essence of these blueprints is to facilitate the comparison between elements for a given selected domain using a graph metaphor, composed of nodes and edges.

The size of a node or an edge tells us about their relative importance. In the case that a node is "much larger" than other nodes, this should draw the attention of the process designer to this particular node. Its size may reveal an anomaly or a misconception.

The visualizations proposed are based on *polymetric views* (Lanza & Ducasse, 2003). A *polymetric view* is a lightweight software visualization technique enriched with software metrics information. It has been successfully used to provide "software maps" intended to help comprehension and visualization. Two-dimensional nodes represent entities. A map including up to 5 metrics can be calculated on the node characteristics: position properties X and Y, height property, width property and color property. A software process blueprint follows the intuitive notion that the wider and the higher the node is, the bigger the measurements its size is telling. The color interval between white and black may render another measurement. The convention that is usually adopted by Gîrba & Lanza (2004) is that the higher the measurement the darker the node is. Thus light gray represents a smaller metric measurement than dark gray. An edge between two nodes n and m may be directed representing asymmetry. Direction is usually graphically represented with an arrowed line. Software process blueprints do not make use of other edge measurements.

5.2.3.2 Role Blueprint

Description and Specification: this blueprint shows a role-oriented perspective of a process model. A role (called *Role Definition* in SPEM 2.0) defines a set of skills, competences and responsibilities required for performing it. A role is responsible of some

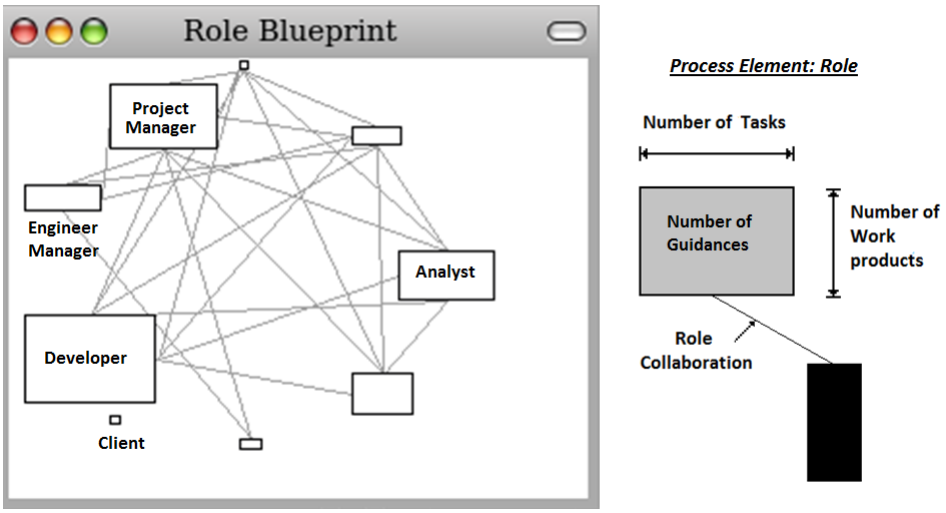

Figure 5.3: Role Blueprint of the DTS Process Model.

tasks and could participate in others; also a role is responsible of some work products in the software process. In general, the more associated to tasks, work products and guidances a role is, the heavier the load the role has. In this blueprint a role is represented as a rectangular node. The width represents the quantity of tasks and the height the quantity of work products this role is in charge of. An edge between two roles represents the collaboration between them, i.e., they work on the same task. Collaboration is not explicit in a SPEM 2.0 model, but it can be obtained from the role associated tasks. The quantity of guidance determines the color of a role(the node is darker to major guidance).

Example: the ROLE BLUEPRINT of DTS is presented in Figure 5.3. It shows collaboration between roles, so it can be seen that a role–the *Client*–appears isolated. This situation suggests that this role is either not involved in the process (and it is a possible conceptual problem), or that the process has been badly specified. On the other hand, the *Analyst* role should have been related to *Engineer Manager* and it is not, so this is an underspecification of the process model. Moreover, in this blueprint two roles are much larger than the others (*Project Manager and Developer*): they have several work products assigned and they participate in a big number of tasks. Furthermore, the role *Client* is small, and this issue suggests that it is not an active participant in the software process.

Interpretation: this blueprint allows the process engineer to evaluate whether the assigned responsibility is suitable to the process requirements. It allows him to discover overloaded roles when one would expected a lightweight one, a lightweight role when a heavy load role is required, a role that is isolated when it requires collaborations, and roles without training material when it would be necessary for a good performance. So, as a response to inconsistencies, a complex role could be decomposed, simple roles could

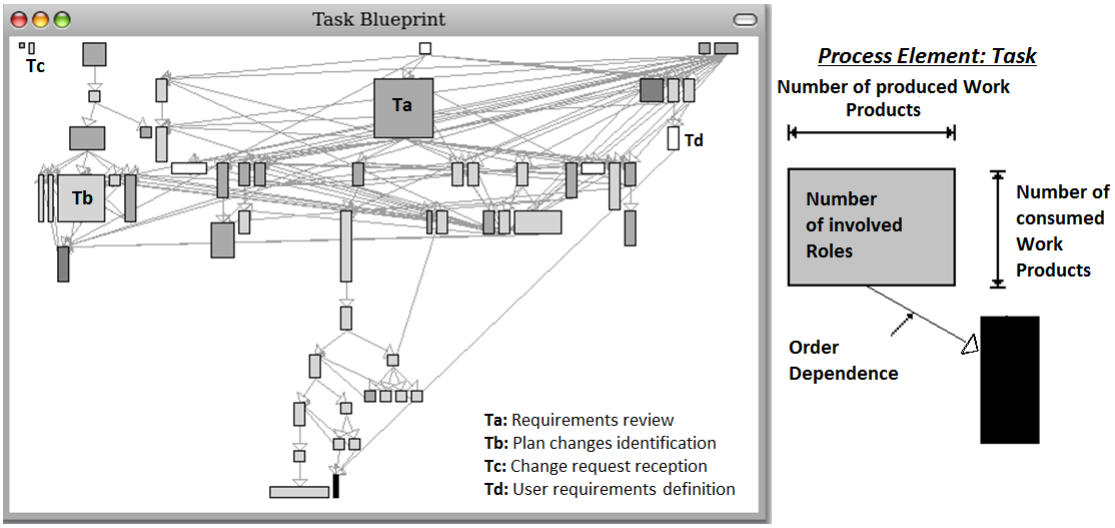

Figure 5.4: Task Blueprint of the DTS Process Model.

be integrated, training material could be added, collaborative techniques and enactment constraints could be added.

5.2.3.3 Task Blueprint

Description and Specification: this blueprint shows a task-oriented perspective of the process model. A task (*Task Definition* in SPEM 2.0) is the basic element for defining a process. It defines a work unit assignable to roles. The granularity of a task ranges from hours to several days. A task is associated to input and output work products. A task has a clear purpose: it provides a complete explanation step by step of the associated work. In this blueprint a node represents a task and it is represented as a box. The width and height of a task represent the quantity of work products that are required and produced, respectively, and the color represents the number of roles involved in the task. A directed edge between two tasks states an execution order dependency between them: T1 depends on T2 if T1 uses work products produced by T2 and therefore then will exist an edge from T2 to T1. The organization of the graph uses a tree order layout, so they are placed in a top-down fashion to reflect the order of dependency. As a natural and intuitive good practice, each task must be connected to a previous and a next task, except for the start and final work products.

Example: the TASK BLUEPRINT of DTS is presented in Fig. 5.4. The size of the tasks is related to their complexity. In the example, the *Requirements Review* task is more complex than all others, so more decomposition would be probably necessary. For example, *Requirements Review* task could be decomposed into *Software Requirements*

Review task and *User Requirements Review* task. The color shows the participant roles, so the *User Requirements Definition* task does not have associated roles. Unless this task is automatic, this situation represents a specification error. Other problems may be identified in the model such as the existence of many initial and final tasks; some of them are intermediate tasks without a possible next or previous task. Another problem arises due to the multiple links between tasks belonging to different process areas. This indicates a high coupling between the main components of the process. The tasks on the upper left of the picture belong to the management process area, those on the upper right correspond to the engineering process area and those on the bottom belong to testing. These process areas, interpreted as process components, should be connected between them with as few dependencies as possible using process ports as SPEM2.0 suggests; they can be used with composed work products and reports generated from other work products between main or facade tasks inside each process component. Some sets of related small tasks can be redefined as steps of a larger task.

Interpretation: this blueprint allows the process engineer to evaluate if task granularity is appropriate for process requirements. It enables the discovery of a complex task, or a disconnected task or task sub graph. A complex task could be decomposed and simple tasks could be integrated. Similarly, this blueprint could be used to find SPI opportunities and misspecifications such as process components with high coupling, final or initial tasks with next or previous tasks, and tasks without assigned roles.

5.2.3.4 WorkProduct Blueprint

Description and Specification: this blueprint shows a work product-oriented perspective of the process model. A work product (*Work Product Definition* in SPEM 2.0) is an element used, modified or produced by tasks. Roles use work products to perform tasks and produce other work products. Roles are responsible for work products, so they aid in identifying skills required for producing them. In this blueprint each node is a work product and it is represented as a box. The width and height of the box represent the quantity of tasks where the work product is an input or an output, respectively. The color refers to the quantity of associated guidances. An edge between two work products designates a production dependency: W1 depends on W2 if W1 is an output of a task where W2 is input. Normally, any work product must be connected to previous and next work products except for the first and the last.

Example: the Work Product Blueprint of DTS is presented in Fig. 5.5. The work product view suggests a process with small work products except for a few of them. *System Requirements Specification* work product is big and it can probably be divided into simpler

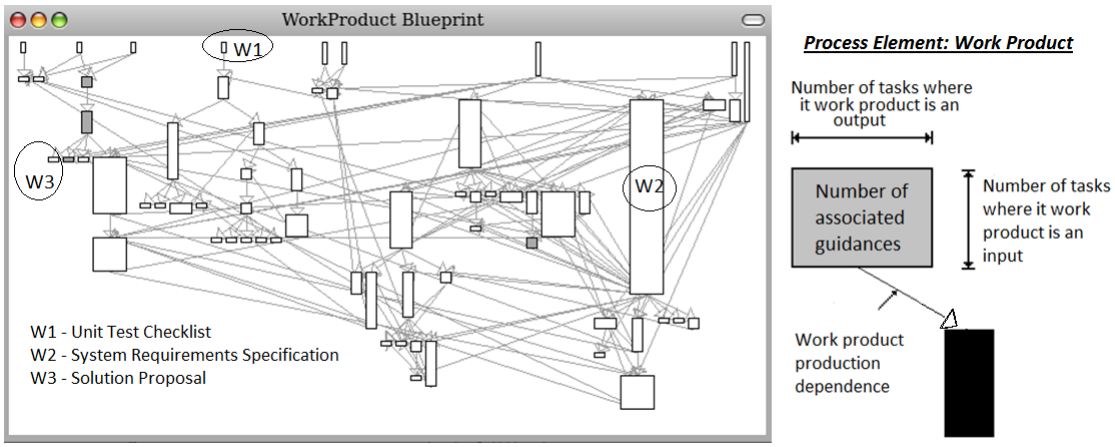

Figure 5.5: Work Product Blueprint of the DTS Process Model.

work products. Very few work products appear colored, so this process has not defined a homogeneous guidance for executing the defined process. This suggests that the process may not be repeatable because each performer may choose a different representation and define himself a specific format. The work product *Unit Test Checklist* work product is produced during the development, but it is not represented in the model, so it is incorrectly modeled as an initial work product. The same situation occurs with *Solution Proposal* work product because it is not a final product but no task consumes it. It is either a waste work product, so process grease can be eliminated (improvement of the process), or it was wrongly specified (improvement of process model specification).

Interpretation: this blueprint allows to process engineer to evaluate if a work product has a considerable bottleneck risk associated. For example, it enables to discover a disconnected work product, or to identify isolated descriptions in the process model. Similarly, this blueprint can be used to find SPI opportunities such as complex, bottleneck and waste work products. Guidances are very important in this blueprint because the more formal a process model is, the more reference material it requires, so the darker the boxes will appear. For example, UP defines templates for each work product, whereas XP does not. Similar to tasks, work products must be separated by process areas; connections between them are first candidates to be component ports.

Table 5.1 summarizes the meaning of boxes, edges and dimensions for each of the blueprints just presented.

	Role Blueprint	***Task Blueprint***	***Work Product Blueprint***
Layout	Circle layout	Tree ordered layout	Tree ordered layout
Node	Role	Task	Work product
Edge	Role collaboration	Task order dependence	Work product production dependence
Scope	Full Process Model	Full Process Model	Full Process Model
Node color	Associated guidances	Associated roles	Associated guidances
Node height	Required and produced work products	Required work products	Tasks where it is an input
Node width	Tasks where it participates	Produced work products	Tasks where the it is an output

Table 5.1: Blueprints Details

5.3 AVISPA

In a previous work, the process blueprints proposed enabled the identification of *exceptional entities* (Demeyer *et al.*, 2002). Blueprints have been successfully used to identify a number of flaws in an industrial process model, but a lot of experience from the process engineer is required for identifying these flaws. Afterwards and based on these experience, several industrial process models have been assessed, and a set of *recurrent patterns* ranging from suboptimal modeling to misconceptions and misspecifications has been discovered.

This section is about presenting, formalizing and implementing these recurrent error patterns as a tool. Recurrent errors appearing in software process models have been defined, and their potential consequences have been explained. This subsection also shows how each of these error patterns can be identified within a software process blueprint.

AVISPA (Analysis and VIsualization for Software Process Assessment)[1] has been built as part of this thesis. This tool displays the blueprints and highlights error patterns. Counting on this tool, the process engineer only needs to analyze highlighted elements, demanding less experience and also less previous knowledge for effective process model analysis, and adding usability as well.

Using visualization to identify error patterns is not new. A large body of research use visual patterns to identify positive or negative properties of software systems (Lanza & Ducasse, 2003; Perin *et al.*, 2010). However, none of the related work presented and discussed in this thesis has been based on visual patterns for identifying problems in software process models. However Knab *et al.* (2010) specify generic visual process patterns that can be found in issue tracking data. With these patterns they analyze information about effort estimation, and the length, and sequence of problem resolution activities. In this

[1] `http://www.moosetechnology.org/tools/ProcessModel` AVISPA is freely available under the MIT license.

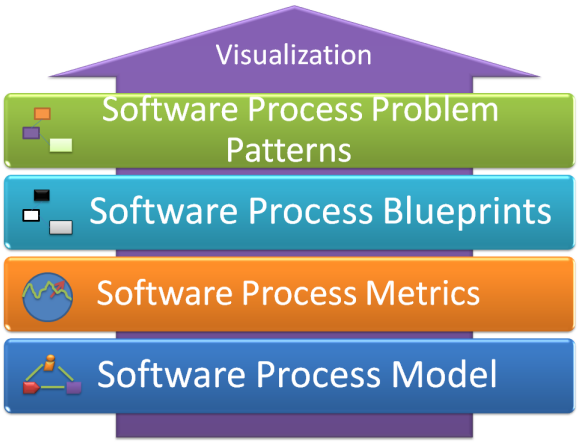

Figure 5.6: AVISPA in localizing software process model improvement opportunities

case the information is about the software process execution trace but not the software process model itself.

Figure 5.6 depicts the pile of technologies involved in building AVISPA. Software process models specified in SPEM 2.0 are assumed to exist. SPEM 2.0 was considerer in this work since it is the standard of OMG, and it has also been promoted within the Chilean software industry by the Tutelkán project. On top of them series of software process metrics are defined; these metrics will be used for identifying errors and improvement opportunities. Software process blueprints are built using these metrics. Error patterns are identified as those elements or constructs within blueprints whose values for certain metrics satisfy some constraints. The identified elements are visually highlighted using AVISPA.

AVISPA has been applied for analyzing the Scrum process model[1] published in the Eclipse Process Framework community where it has been defined as a SPEM 2.0 model OMG (2008) using Eclipse Process Composer[2] to illustrate the approach and the tool (Hurtado *et al.*, 2011a). Process engineers have been able to find several of the defined error patterns, and most of them resulted in actual errors, giving support to our intuition that a formal tool that helps the process engineer is useful. AVISPA has been highly welcomed in all companies where it was presented, these experiences are reported in more detail in Chapter 6.

[1]Scrum:http://www.eclipse.org/epf/downloads/scrum/scrum_downloads.php
[2]EPF: http://www.eclipse.org/epf/

5.3.1 Example Process: Scrum

Scrum is an agile software process frequently used to rapidly develop software. It has been defined by Jeff Sutherland and more formally elaborated by Schwaber (1995). Scrum stresses management values and practices, and it does not include practices for technical parts (requirements, design, and implementation); this is why it is usually used in combination with another agile method such as XP (Beck & Andres, 2004). The application of Scrum enforces a few simple rules that make a team self-organize into a process that can achieve 5 to 10 times the productivity of a waterfall based process. However, most Scrum teams never achieve this goal (Sutherland *et al.*, 2009). According to Sutherland, teams face difficulties to organize work in order to deliver working software at the end of each sprint. Moreover, they also experience trouble working with a Product Owner to get the backlog in a ready state before bringing it into a sprint. Also, organizing into a hyper-productive state during a sprint remains a challenging issue. A hypothesis is that one of the reasons for this situation is an improper definition and implementation of Scrum.

5.3.1.1 Scrum: a rule-based process framework

Scrum is an agile method that works under the idea that software processes are incompletely defined. So, the Scrum approach assumes that the analysis, design, and development processes are unpredictable. A control mechanism is used to manage the unpredictability and control the risk for improving the process flexibility, responsiveness, and reliability (Schwaber, 1995). Scrum is not a process or a technique for building products; rather, it is a rule based framework where various processes and techniques can be used. The role of Scrum is to achieve the most effectiveness possible of a series of development practices while providing a framework where complex products can be developed (Schwaber & Sutherland, 2010). The Scrum framework is a set Scrum teams, time-boxes, artifacts and their rules:

- Scrum teams are designed to maximize flexibility and productivity; Scrum teams are self-organizing, cross-functional, and work in iterations. Each Scrum team has three roles: the *Scrum Master*, who is responsible for ensuring that the process is understood and followed; the *Product Owner*, who is responsible for maximizing the value of the work that the Scrum team does; and the *Scrum Team*, which does the work. The Team consists of developers with all the skills to transform the Product Owner's requirements into a potentially releasable piece of the product by the end of the sprint. Scrum employs time boxes to create regularity.

- The time-boxed elements are the release *Planning Meeting*, the *Sprint Planning Meeting*, the *Sprint*, the *Daily Scrum*, the *Sprint Review*, and the *Sprint Retrospective*. The focus of Scrum is a sprint, which is an iteration of one month or less that is of consistent length throughout a development effort. All sprints use the same Scrum framework, and all sprints deliver an increment of the final product that is potentially releasable.

- Scrum employs four main artifacts. The *Product Backlog* is a prioritized list of features required for the product. The *Sprint Backlog* is a list of tasks to be performed in a sprint, producing an increment of a potentially shippable product from the Product Backlog. A burn down is a measure of remaining Backlog over time. A *Release Burn down Chart* measures remaining Product Backlog in the context of a release plan. A *Sprint Burn down Chart* measures remaining sprint backlog in the context of a sprint.

Scrum life cycle is defined by the *sprint* and by three groups of phases: *pre-game, game and post-game*. In the *pre-game*, the planning and architecture phases are performed. In the *planning phase* a new release is defined according to the current product backlog, including an estimative of its schedule and cost. In the *architecture phase* a design (architectural and high level design) is generated to determine how the backlog items will be implemented. In the *game phase*, the sprints are performed. There are multiple, iterative development sprints that are used to develop the system. In the *post-game* the closure phase is performed. The release is prepared including final documentation, pre-release staged testing, and release. Figure 5.7 is a screen shot of the Scrum model defined using EPF tool.

5.3.1.2 Scrum Process Model in SPEM 2.0

The Scrum process model presented by the Eclipse Process Framework Community has been defined as a SPEM 2.0 model as shows Figure 5.8 using Eclipse Process Framework. This definition includes a Scrum plug in and a Scrum overview configuration. The plug in includes the Scrum method package and four categories. Roles, work products, tasks, and the guidance, are organized in packages inside the Scrum method package. Also, the Scrum activities, roles, work products and guidance are defined as categories linking to corresponding method elements defined in the method package. Process structure with method has not defined because in essence Scrum is an incomplete process and it is considered as a process framework and not a process itself. As a consequence, EPF community has defined Scrum life cycle as a supporting material element (a specific guidance) where it is graphical and textually described. Although, this definition does not

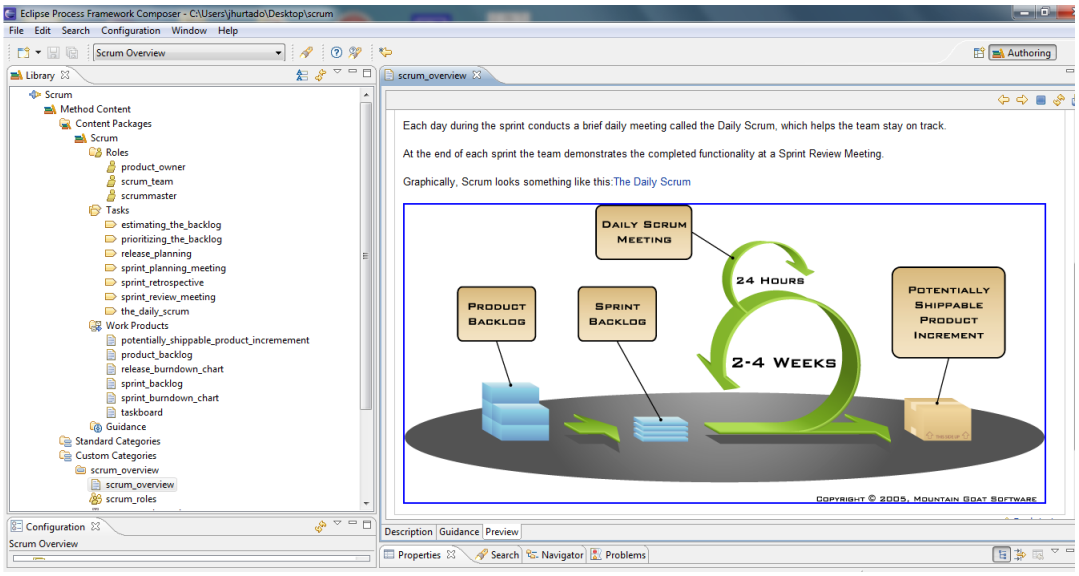

Figure 5.7: Scrum Process Model in EPF

include the phases defined by Schwaber (1995), the Scrum life cycle could be defined and customized in a delivery process by each organization reusing the Scrum method plug-in. The method package elements have been defined and linked according to a Scrum description as was presented above. However, the question is if the method elements will match with this or other adapted life cycle. For example, are the tasks outputs and inputs consistent among the tasks inside a value flow? These are relevant questions mainly when the model is used with academic purposes, for the first time, for comparing or combining it with other process models.

5.3.2 Process Model Error Patterns

For the past five years has been followed applied research in the area of software process models in small software companies in Chile (Hurtado *et al.*, 2010a; Valdés *et al.*, 2010) and Iberoamerica (Pino *et al.*, 2009; Villarroel *et al.*, 2010) as part of the Tutelkán and Competisoft projects. Along this work a number of common errors and problematic situations in software process model specifications have been identified, either due to misconceptions or misspecifications. In this section a series of these patterns are reported, how they may be identified in AVISPA, and mainly how they are automatically highlighted as part of the blueprint where they appear. In this work there is a misspecification in the software process model if the development process is well designed but its specification does not necessarily reflect the actual practice, e.g., there exist some guidance for a role but it is not specified as part of the model. Complementary, there is a misconception

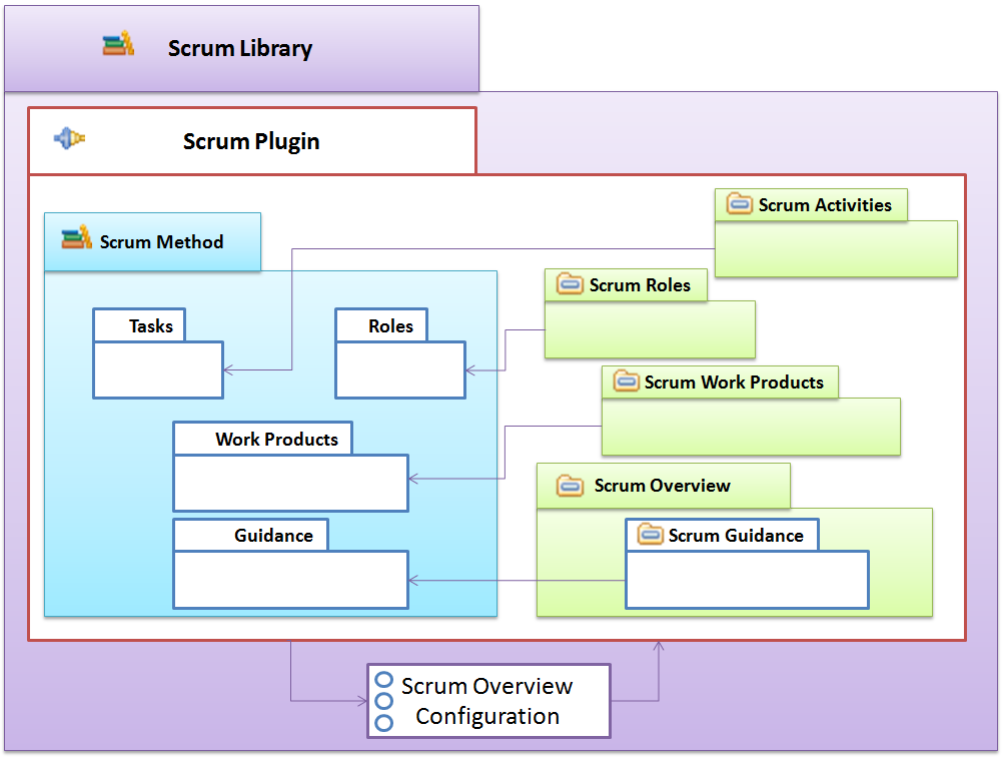

Figure 5.8: Scrum Process Model as SPEM2.0 Model

whenever there is a flaw in the software process design, e.g., a task produces a work product that neither a task nor a role needs.

5.3.3 AVISPA Error Patterns

There is a number of anomalies in software process specifications that are fairly frequent. These anomalies have been structured as *Error Patterns*. This section describes some of them along with their consequences and how they would look in the blueprint where they may be found. Error pattern descriptions provide a tentative quantification for how bad may be considered too bad, so that it serves as a basis for automating their localization.

No guidance associated error pattern. If a role, task or work product has no guidance about how to be executed, there is a big chance that it will not be properly done. This error is generally an underspecification, meaning that there should have been certain guidance associated with each element. In the respective blueprint elements without guidance are highlighted in blue.

Overloaded roles error pattern. If a role is involved in a large number of tasks, it becomes a risk: if it fails, all the associated tasks within the process will fail as well. This is a clear anomaly in the process model conception. Better choices would be either

specializing the role by dividing its responsibilities, or reassigning some tasks to other roles. A role is overloaded if it is more than one standard deviation larger than the average size. This error pattern is computed as part of the Role Blueprint, and the overloaded roles are highlighted in red.

Isolated roles error pattern. There may be certain tasks that a role executes by itself, but it is not frequently right to have a role that never collaborates in any task with other roles. In general, this kind of error pattern shows a misspecification: a role should have been assigned to take part in a certain task but appears to be left apart. This error pattern is also apparent in the Role Blueprint, and the isolated roles are highlighted in green.

Multiple purpose tasks error pattern. A process where tasks have too many output work products may reveal that these tasks are not specified with the appropriate granularity. A task with too many output work products may be too complex since its goal is not unique. This may reflect a misconception in the process model. This pattern is seen in the Task Blueprint where wide nodes (too many output work products) are highlighted red. A task is considered to be too complex if it is more than one standard deviation than the average task width. A better choice could be to divide the task in two or more tasks with more specific purposes.

Demanded work products. Work products required for a high number of tasks may cause serious bottlenecks when they are not available, and thus it could reveal a misconception. This situation is seen in the Work Product Blueprint where wide nodes are highlighted in yellow, nodes whose width (number of tasks that require it as an input) is more than one standard deviation than the average.

Independent sub projects error pattern. In a Task Blueprint and a Work Product Blueprint, tasks and work products are related with edges indicating precedence. Considering that the process model specifies the way to proceed when working on one unique project, it is conceptually odd to have disconnected subgraphs, both in the Task Blueprint and the Work Product Blueprint. In general, these situations arise due to under specifications, when work products have not been specified as input or output work products for certain tasks when they should have been. Each subgraph is presented with a different color in both, the Task Blueprint and the Work Product Blueprint, in order to identify the existence of independent sub projects. So having a graph with more than one color nodes indicates that there are independent sub projects specified.

Waste error pattern. In Work Product Blueprint an arc connecting nodes represents precedence between work products. If there is a WP_a that precedes WP_b in

the graph, that means that there is a task such that WP_a is its input and WP_b is its output. In this way, all leaves in the graph, i.e., nodes with no successor, should represent deliverable work products. This pattern highlights in blue all those leaves that are not defined as deliverables. In SPEM 2.0 deliverables are those work products that need to be delivered to the customer as part of the final product. However, if there are work products that are neither deliverables nor input for any other task within the process, then they are waste.

Table 5.2 summarizes the error patterns that have been identified so far.

Error pattern	**Description**	**Localization**	**Identification**
No guidance associated	An element with no guidance associated.	any blueprint	A white node.
Overloaded role	A role involved in too many tasks.	Role Blueprint	Nodes over one deviation larger than the mean.
Isolated role	A role that does not collaborate.	Role Blueprint	A node that is not connected with an edge.
Multiple purpose tasks tasks	Tasks with too many output work products.	Task Blueprint	Nodes whose more than one deviation wider than the mean.
Demanded Work products	Work products required for too many tasks.	Work Product Blueprint	Nodes more than one deviation higher than the mean.
Independent subprojects	Independent subgraphs.	Task Blueprint or Work Product Blueprint	Subgraphs that are not connected with edges.
Waste	A waste work product	Work Product Blueprint	A work product non deliverable and leave in the graph.

Table 5.2: Error patterns identified by Avispa

5.3.4 Localizing Errors with AVISPA

This section sketches the internals of the implementation of Avispa. The scripts for implementing two of the error patterns are subsequently offered as examples of the way errors are computed. Finally, a description of the tool from the user point of view is provided.

5.3.4.1 Implementation of AVISPA

The SPEM 2.0 error patterns presented in the previous subsection were implemented in Avispa extending the MOOSE platform according to the architecture defined in Figure 5.9. As Figure 5.10 shows, Avispa extends the FAMIX family of meta models of MOOSE[1] by subclassing MooseEntity and MooseGroup. The names of the classes that

[1] `http://www.moosetechnology.org/docs/famix`

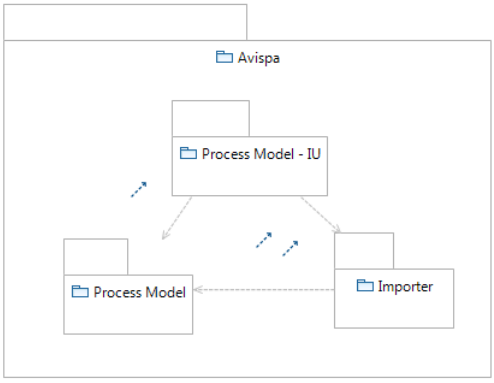

Figure 5.9: The AVISPA logical architecture

belong to AVISPA begin with PM, standing for Process Model. PMObject contains operations and attributes common to all SPEM elements (essentially a particular identifier). PMRole, PMTask and PMArtifact describe elementary components of SPEM 2.0. Each of these classes offers methods for computing metrics and navigating through a model. For example, each task is aware of its following tasks (*i.e.,* tasks that farther need to be completed) and its associated artifacts. A group of roles, tasks and artifacts are expressed as instances of PMRoleGroup, PMTaskGroup and PMArtifactGroup, respectively. The purpose of offering specialized collections is to enable dedicated visualization to be defined on these groups. For example, the method viewTaskBlueprintOn: is defined on PMTaskGroup which defines the enhanced task blueprint describe below.

AVISPA is visualized using the Mondrian visualization engine[1] (Meyer *et al.*, 2006). Mondrian operates on any arbitrary set of values and relations to visually render graphs. As exemplified below, visualizations are specified with the Mondrian domain specific language.

5.3.4.2 Error Pattern Implementation in AVISPA

The implementation is illustrated with two error patterns: independent projects and multiple purpose tasks, i.e.,tasks involving too many output work products. There is a script for each of them, and the rationale in each implementation. The implementation of the other error patterns is conceptually similar to these ones.

Independent sub projects This kind of error is seen, for example, when the TASK BLUEPRINT has disconnected subgraphs. Thus, each independent subgraph is colored differently, and having a TASK BLUEPRINT with more than one color means that there

[1] `http://www.moosetechnology.org/tools/mondrian`

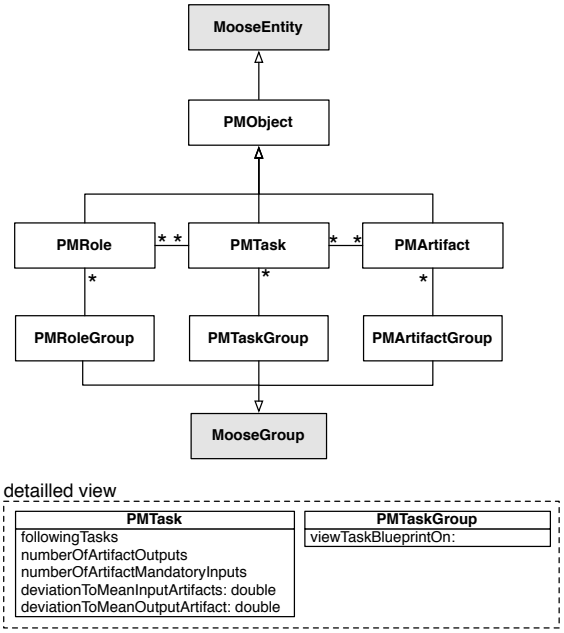

Figure 5.10: The AVISPA metamodel (gray classes belong to FAMIX)

are some missing dependencies. On the other hand, if the TASK BLUEPRINT is all the same color, this will mean that there are no independent sub projects, and therefore the process will be fine with respect to this error pattern. The following script builds a colored TASK BLUEPRINT where independent sub projects are identified. Independent subproject always reveal an error in the process model specification.

```
PMTaskGroup>> viewTaskBlueprintOn: view
  | ds components orderedComponents normalizer cycleColor |

  "Compute disjoint sets"
  ds := MalDisjointSets new.
  ds nodes: self.
  ds edges: self from: #yourself toAll: #followingTasks.
  ds run.

  components := ds components.
  orderedComponents := Dictionary new.
  components doWithIndex: [:roles :index |
        roles do: [:role |
              orderedComponents at: role put: index]].

  "Assign a color to each set"
```

```
normalizer := MONIdentityNormalizer new.
(1 to: (components size * 10) ) do: [:v — normalizer moValue: v].
cycleColor := [:v — normalizer moValue:
        ((orderedComponents at: v) + 10)].


"Display the blueprint"
view shape rectangle
        borderColor: Color black;
        borderWidth: 1;
        fillColor: [:v | cycleColor value:v];
        width: [:each | each numberOutputs * 10];
        height: [:each | each numberInputs * 10].
view nodes: self.
view shape arrowedLine.
view edges: self from: #yourself toAll: #followingTasks.
view treeLayout
```

First a cycle is computed so that edges of connected subgraphs are painted with the same color. Then, individual nodes are built assigning them a size and a color. Tasks are represented as rectangular nodes whose color is that of the subgraph it belongs to. Their width is related to the number of output work products, and the height shows the number of input work products. Arrows between two nodes exist if they are related with the *following* relationship. The whole blueprint is shown as a tree.

Multiple Purpose Tasks Here again the error will be seen in the TASK BLUEPRINT, but now nodes that are wider than one standard deviation from the average number of output work products will be highlighted as potential errors. Highlighted tasks reveal complexity in the task specification, but they are not necessarily errors. One standard deviation in a normal distribution function was the empirical value calibrated from a preliminary analysis. A serie of metrics are precalculated so that the script can be executed. $numberOutputs_i$ is the number of output work products of task i in the process. Then, considering that there are n tasks in the process, the error pattern calculates the mean number of output work products for the whole process as follows:

$$MeanOutWP = \frac{\sum_{i=1}^{n} numberOutputs_i}{n} \tag{5.1}$$

And then, the standard deviation can be calculated as follows:

$$sigmaOut = \sqrt{\frac{\sum_{i=1}^{n}(numberOutputs_i - MeanOutWP)^2}{n}} \tag{5.2}$$

Also, the distance from the mean value to *MeanOutWP* is calculated as follows:

$$distToMeanOutWP_i = numberOutputs_i - MeanOutWP \tag{5.3}$$

These metrics are used as part of the script in order to determine the color of each node in the Task Blueprint.

```
PMTaskGroup>>viewTaskWarningBlueprintOn: view
view shape rectangle
  fillColor: [:each | (each distToMeanOutWP >
                              self myModel sigmaOutWP)
        ifTrue: [Color red]
        ifFalse:[Color white]];
  borderColor: Color black;
  width: [:each | each numberOutputs * 10];
  height: [:each | each numberInputs * 10].view nodes: self.
view shape arrowedLine.
view edges: self from: #yourself toAll: #followingTasks.
view treeLayout.
view root interaction item:
              'inspect group' action: [:v | self inspect]
```

The main part of the script is devoted to determining the color of each node according to its relative size. If the distance from the number of output work products to the mean is larger than one standard deviation, then the node will be red. Otherwise, the node will be white. Edges will be drawn according to the *followingTasks* set that should have been precalculated. The whole blueprint is presented as a tree.

Obtaining a Task Blueprint that is all white means that all tasks have similar complexity with respect to the number of output work products. Several red tasks clearly suggest a poor design because the purpose of the tasks is not always uniquely defined.

5.3.4.3 AVISPA User Interface

Avispa has become a useful tool to import and visualize SPEM 2.0 based process models. It is built on top of Moose and the Pharo programming language[1], and so it benefits

[1] `http://www.pharo-project.org`

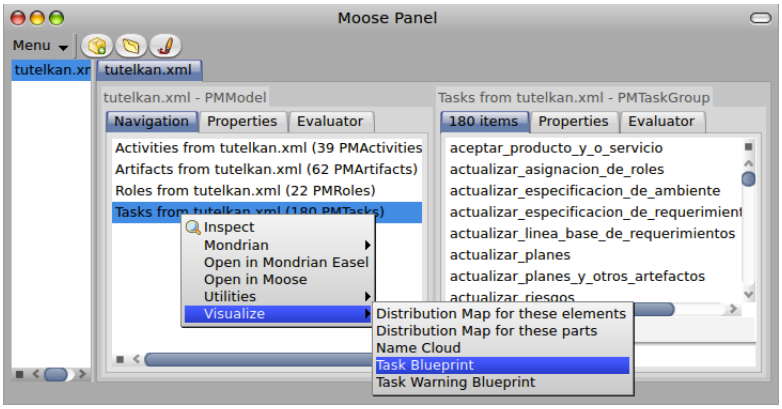

Figure 5.11: The AVISPA main user interface

from a large toolset for navigation and visualization. Figure 5.11 shows the main user interface. The Tutelkan model has been loaded and is ready to be analyzed. The navigation panel shows four entry points to begin an analysis: *activities, artifacts, roles and tasks*. Navigation is realized through the information available in the meta model (see Sect. 5.3.4.1). Although not depicted in the figure, metrics and other specific information (*e.g.,* descriptions and annotations) are also available under the *properties* tab.

5.3.5 Applying AVISPA to the Scrum Process Model

The AVISPA tool was used for analyzing the Scrum process model defined by the EPF process community. It is exported from EPF as an XML file and imported in AVISPA. AVISPA is guided by the kind of error patterns it is able to identify and localize, so the analysis is organized accordingly.

- *No guidance associated.* Process elements leave too much freedom for interpreting the purpose of each element within the process. Scrum provides guidance, but we have found that they are not always associated with the corresponding nodes (see Figure 5.12). In the ROLE BLUEPRINT we found that absolutely no guidance is provided for any role. In the WORK PRODUCT BLUEPRINT, the *Task board* and the *PotentiallyShippableProductIncrement* have no guidance either. This situation is even worse in the TASK BLUEPRINT because the *Sprint Retrospective*, *Sprint Planning Meeting*, *Sprint Review Meeting* and the *Daily Scrum* do not have associated guidance. This situation is particularly serious for Scrum because of its agility: if neither methods nor guidance are provided, it is difficult to achieve the expected results.

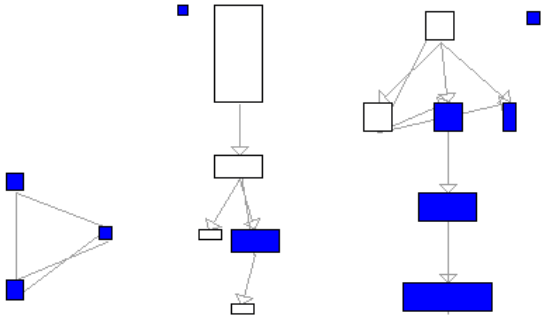

Figure 5.12: Role Blueprint, Work Product Blueprint and Task Blueprint identifying elements without guidelines

- *Overloaded role and isolated role.* Generating the Role Blueprint (see Figure 5.13) there are neither overloaded nor isolated roles. Thus, there are no problems in the specification of Scrum with respect to error patterns referring roles.

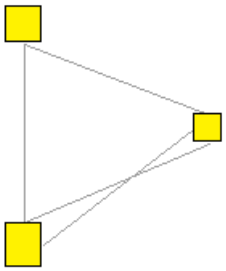

Figure 5.13: Applying Avispa for localizing overloaded and isolated roles in Scrum

- *Multiple purpose tasks.* A task that is too wide in the Task Blueprint will be colored in red in order to call the attention of the process engineer. A task with too many output work products does not have one clear goal, so it may be better to divide it into more specific subtasks. In Figure 5.14, it can be seen that the *Sprint Review Meeting* and *Daily Scrum* tasks are significantly wider than the others, although only one is highlighted(*Sprint Review Meeting*). This is expected in Scrum because these tasks are defined as black boxes hiding the complexity of the software development in Scrum. But, as a software development process, the specification of Scrum is not detailed enough. This is consistent with the literature in the fact that Scrum should be combined with others methods.

- *Demanded work products.* A work product that is required for the execution of too many tasks could become a bottleneck, so a work product that is too demanded

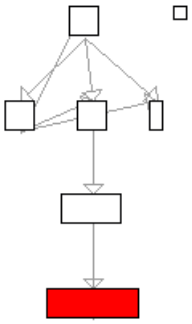

Figure 5.14: Applying AVISPA for localizing multiple purpose tasks

reveals a problem in process conceptualization. A node in the WORK PRODUCT BLUEPRINT that is too high identifies this kind of problem. This is the case of *Product backlog* that can be clearly identified in Figure 5.15. It means that if the *Product backlog* is not defined (or partially defined) when the project is not finished yet, then it could be stopped.

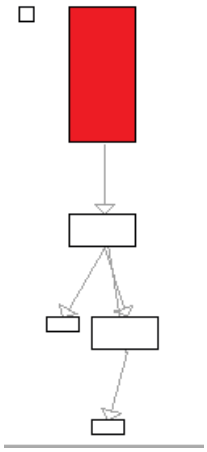

Figure 5.15: Applying AVISPA for localizing demanded work products

- *Independent sub projects.* Having independent sub projects reveals a misspecification in the process model because all tasks and work products should be useful for the project's goals, and as such they should be connected in the TASK BLUEPRINT and the WORK PRODUCT BLUEPRINT, respectively. This error pattern may be seen in either blueprint. Figure 5.16 shows how independent subgraphs have different colors (color is meaningless) in the WORK PRODUCT BLUEPRINT for Scrum. The work product in green, *Potentially Shippable Product Increment*, belongs to a independent graph. This implies that this work product is neither defined as an

input nor output of any task in Scrum. A similar situation occurs in the green task *Sprint Retrospective* that is disconnected from the graph in Figure 5.17. The retrospective is a key task in the Scrum process that consumes and produces changes on work products (and outcomes) of other tasks. The purpose of the retrospective is to inspect the last sprint with respect to people, relationships, process and tools. These include Scrum team structure, meeting arrangements, support management tools, methods of communication, and processes for turning Product Backlog items into value. Clearly the model is imprecisely described with respect to this practice.

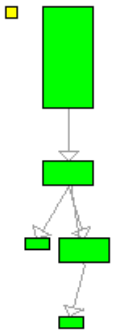

Figure 5.16: Applying AVISPA for localizing independent projects in the WorkProduct Blueprint

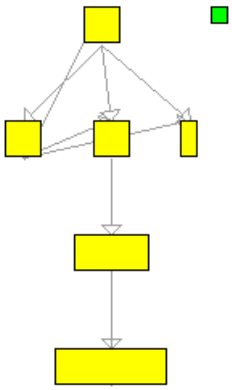

Figure 5.17: Applying AVISPA for localizing independent projects in the Task Blueprint

- Waste. The *Potentially Shippable Product Increment* (A) has been highlighted in Figure 5.18. This work product needs to be an input to the integration task, but the public Scrum process model does not specify this fact, so (A) is an underspecification. The *Release Burn down Chart* (B) and *Sprint Burn down Chart* (C) are clearly necessary for executing the development management tasks, but the model

does not specify these dependencies either. These dependences could be underspecifications, but the work products could be waste too. It depends of the cost/benefit of these work products. Actually, these artifacts do not generate value by themselves, the benefits are required in Scrum method for determining for example as predict the project velocity and the time remaining, so these work products are considered as completely necessary. In this analysis, no false positives are identified: all highlighted elements correspond to errors in the process model.

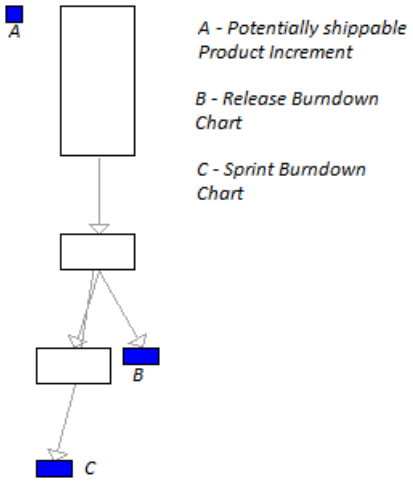

Figure 5.18: Work products that are potential waste in Scrum

5.3.6 Scrum Analysis Results

A software process model definition can be incomplete for agility reasons or self-organization. However, if a relevant principle of the process is not included, it could be applied in an imprecise or inaccurate way. In this subsection the Scrum process model has been analyzed with the AVISPA tool, where the specification widely used by the community has been found to be incompletely defined. In particular, the iterative nature of the process and the incremental nature of the product development was found unclear.

An improved Scrum process model can be completed from its original definition by adding: (i) guidance such as key concepts, examples and guidelines available in the Scrum definition associated to each role; for instance *Product Backlog* Example, *Story Points* Key Concept and *Priorization of the Backlog* guideline should be associated to the Product Owner Role, (ii) some tasks need to be completed associating them with their required and produced work products; for instance, because in the *Spring Retrospective* task *Sprint Burn Down* and the *Task board* Work Products are used and modified, they need to be associated as input and output, respectively, and (iii) the *Task board* Work Product should

be completed with a Task board example; *Potentially Shippable Product Increment* should be defined as an output of the *SprintReviewMeeting* task.

5.4 Synthesis and Discussion

As a complement to process model design, it is also necessary to validate and verify software processes. This chapter presented a mechanism for recovering software process architectural views based on blueprint visualization. This approach proposes to analyze software process models in an early way, based on reviewing the architectural views of a software process model defined as Software Process Blueprints (Hurtado *et al.*, 2010b). There, each blueprint is built following a model-driven strategy where the process model is separated in a set of partial views that may be more illustrative for finding errors than analyzing the whole process directly. Many process model blueprints can be defined, but three essential ones have been described here: Task Blueprint, Role Blueprint and Work Product Blueprint. At the top of the blueprints a set of error patterns has been identified and implemented as part of Avispa, a tool for process model analysis that localizes this set of identified potential errors within a process model specified in SPEM 2.0. These errors may come either from process conceptualization or from misspecifications. This chapter described how each of the error patterns identified are found in the appropriate process blueprint, and how Avispa highlights them. Avispa encapsulates knowledge specialized of an expert software process engineer for identifying improvement opportunities and thus requiring less experienced process engineers for determining a better process.

The quality of the analysis highly depends on the quality of the definition of the error patterns. Even though the error patterns developed in this chapter have shown to be effective in finding improvement opportunities, there is still some room for fine tuning them. For example, in other cases determining that a task is too complex if it is more than one standard deviation from the mean, may not help discriminating really badly specified elements, and maybe two standard deviations is a better measure. The error pattern may also be defined as parametric in the number of standard deviations considered.

The Avispa tool is targeted to those software process models formally specified in SPEM 2.0. This may be one of its main limitations since it is hard and expensive to formally define a complete process. However, if a company decides that it is worth the effort to specify its software process, then Avispa provides an added value to this investment assuring, at least partially, the quality of the specified process.

Chapter 6

CASPER Validation

6.1 Introduction

This chapter reports the empirical application of CASPER. According to the research method defined in Chapter 1, an industrial case study has been used as the validation technique. The study case is holistic because an organizational software process construction was used as a complete research subject (Yin, 1984). According to research theory of Runeson & Höst (2009), this case study is positivist (to test the hypothesis) and explanatory (it is an explanation of a situation where CASPER is applied as a new meta-process).

In this industrial case study, a general requirements engineering process of a medium size Chilean software company was formalized. This enterprise has provided its organizational process as part of the Tutelkan project (Hurtado & Bastarrica, 2010). A rich case aids to reveal more information, activating more actors, more concepts and more basic mechanisms in the situation researched. So, for this case study the requirements engineering process has been taken including its adaptation guidelines. These guidelines indicate that certain artifacts should or should not be included as part of the adapted process, according to certain situations. In this way, there is a set of predefined project types such as large development, small development, maintenance or incident. This case study shows that the approach is able to automatically produce the expected process for these project types. Furthermore, this case study shows how CASPER is also able to produce an appropriate process for an unexpected context. All these results have been analyzed and validated by the company's process engineer. The subsequent sections present the complete industrial case study where the CASPER approach was applied.

6.2 Research Question

CASPER follows a planned software process tailoring strategy to resolve the challenges of traditional tailoring where tailoring involves intensive knowledge generation and deployment (Rolland, 2009) and it is also time consuming (Ocampo *et al.*, 2005). Is it possible to apply CASPER to an industrial case study in a cost-effective way using only project context information?

6.3 Case Study Metrics

To achieve the case study goal, cost-effectiveness in the tailoring process is defined as an indicator ICE - Index of Cost-Effectiveness defined by the equation:

$$ICE = \frac{CR}{TE} \tag{6.1}$$

Where CR is the coverage ratio and TE is the tailoring effort. These metrics are described in Table 6.6.

Table 6.1: Metrics of the Case Study

Metric	**Description**	**Rationale**
CR - Coverage Ratio	The ratio between the available processes and the possible situations	Effectiveness - the larger the number of processes, the more situations will be covered
TE - Tailoring Effort	The effort in hours-persons spent during tailoring stage	Cost - the lower the tailoring effort, the lower the cost of the tailoring

6.4 Case Study Selection

The selection of the requirements process case was driven by several factors: (1-available case) to have an industrial environment, so validation is performed like a real case; (2-extreme case) the requirements process is the more variable part of the complete software process model in the organization, so this part is the richest part of the process model used to validate the approach, and (3- critical case) the requirements process model is known by the research community as the most complex and richest part of the process model in general (Pandey *et al.*, 2010). The complexity in this process is due to several issues: stakeholders (including paying customers, users and developers) may be numerous and distributed; their goals may vary and conflict; their goals may not be explicit or may be difficult to articulate, and, inevitably, satisfaction of these goals may be constrained

by a variety of factors outside their control (Nuseibeh & Easterbrook, 2000). Included data of this case study was obtained from the process line examination and unstructured interviews to their process engineers.

6.5 Case Study Context

KIT is a medium size Chilean software development company, its requirements engineering process, KIT-RE, has been formalized as a software process line. This company has provided its organizational process as part of the Tutelkán project (Hurtado & Bastarrica, 2010; Valdés *et al.*, 2010) and it is publicly available[1]. For this tailoring scenario the requirements engineering process was took, along with its adaptation guidelines.

The current method used by KIT to tailor its software precess is based on guidelines. These guidelines indicate that certain artifacts should or should not be included as part of the adapted process, according to certain context values. In this way, there is a series of predefined project types such as large development, small development, maintenance or incident. However, the guidelines suggest the minimal process to be followed but they do not suggest the most suitable software process. Normally, the project manager meets with the process engineer to select the artifacts to be used in the project. Thus, the tailoring process usually results insufficient using only the guideline, it is dependent on the process engineer and it is repetitive only in some cases. The process engineer incrementally counts on more tailoring rules, but the project managers only count on a partial view of possibles tailoring decisions. Additionally, the company needs to count on a quantitative mechanism for making good tailoring decisions. It considers that formalizing the context and the specific process, it is possible to evaluate int the long term its tailoring decisions. Hence, a systematical software process tailoring allows reusing process engineering knowledge, making in the long term the best decisions, and counting on a more suitable process in each project,achieving the benefits of the tailoring

6.6 Organizational Process Model

The requirements engineering process is part of the KIT Development Process which is based on Rational Unified Process including its four phases:

1. Inception: the project scope is defined, cost and schedule are estimated, risks are identified, the problem analysis is conducted, the project staff is defined and the environment is prepared (including the process tailoring).

[1]Tutelkán: `http://www.tutelkan.org`.

2. Elaboration: requirements are detailed; the architecture is identified, defined and validated, and the environment is updated.

3. Construction: the system is modeled, built and tested. Additionally supporting documentation is developed.

4. Transition: includes system testing, user testing, system deployment and user training.

In the general requirements engineering process we can identify two main components that are executed asynchronously: *Requirements Development* and *Requirements Management*. Figure 6.1 shows the process formalization in the EPF tool.

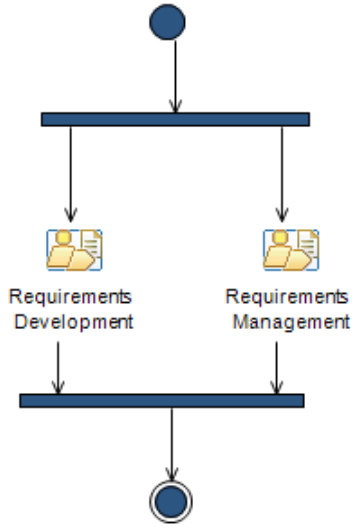

Figure 6.1: Requirements Engineering Process

Requirements Development is depicted in Figure 6.2. Here the process may take two different forms depending on the development stage. In the Inception stage, this process is formed by two parallel and optional activities: *Problem Analysis* and *Environment Specification*. In all other stages, this process is formed by three parallel activities: *Requirements Specification*, *Requirements Analysis and Validation* and *Early Change Management*; only the last one is optional. Also the *Problem Analysis* is formed by the *Preliminary Analysis* and the *Project and Problem Scope Definition*, and this latter one is also optional.

Requirements Management consists of *Requirements Understanding*, *Requirements Commitment*, and then in parallel *Requirements Tracking* and *Requirements Change Management*, as shown in Figure 6.3.

The *Requirements Understanding* process is illustrated in Figure 6.4. It is formed by three tasks: *Identify Requirements Providers*, *Requirements Review* and *Ensuring Common Requirements Understanding*. Notice that the *Identify Requirements Providers* is

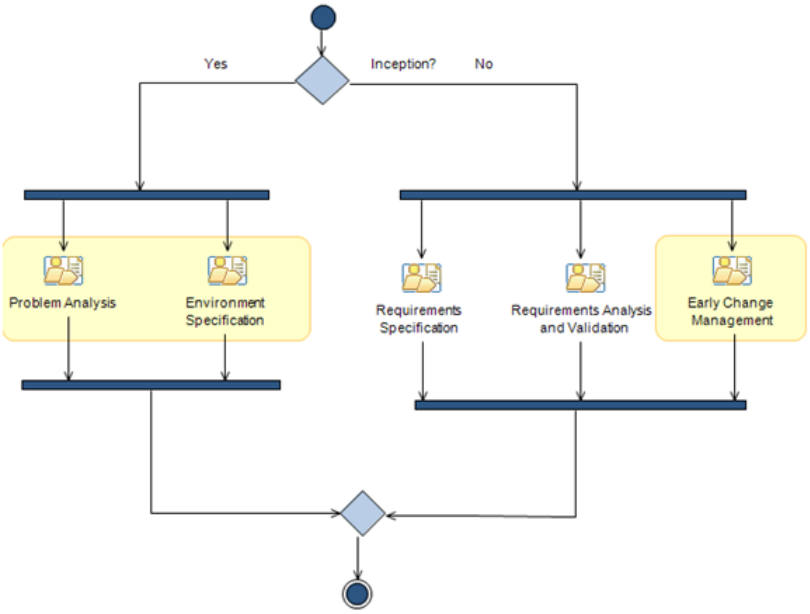

Figure 6.2: Requirements Development

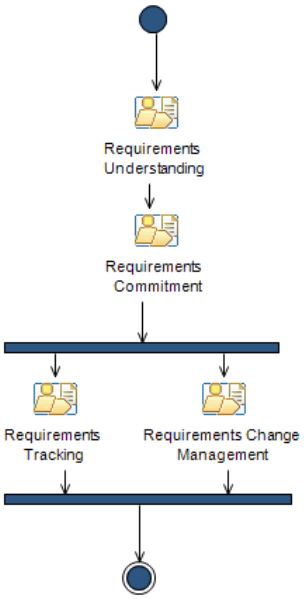

Figure 6.3: Requirements Management

marked as optional. In this case, the task will only be carried out if the project is a new development.

All optionalities in the process can be summarized in a Process Feature Model (Czarnecki & Antkiewicz, 2005) as shown in Figure 6.5.

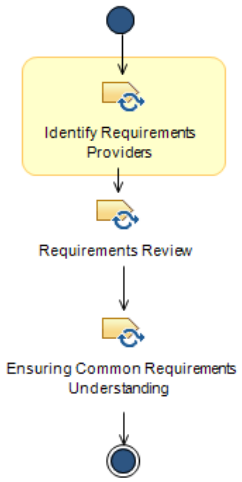

Figure 6.4: Requirements Understanding

6.7 Context Model

The general requirements engineering process model presented in the previous section is applied in different kinds of projects. Several dimensions and attributes have been identified as relevant by the company for characterizing projects. Figure 6.6 shows the context model. The *Domain* dimension has three attributes: *Application Domain*, *Development Environment* and *Source of Documentation*. The first two may be either known or unknown, and the last one may exist, not exist, or there may be an expert who may provide information. Similarly, the *Team* dimension has two attributes: *Team Size* and *Team Expertise*, each one with their corresponding values. The *Management* dimension has five attributes: *Project Type*, *Provider*, *Business*, *Customer Type* and *Project Duration*. In this study case the Team dimension and the Business attribute was not used, however these were defined because it will be used for tailoring other process components besides the requirements process.

The second column in Table 6.2 describes the values of the context variables for a new development within an unknown application domain, whose documentation does not exist, where the development environment and costumer type are unknown, the provider is in house, and the duration is small. In this case the expected tailored process would include all the optional tasks, roles and work products as it is the most complex situation.

On the other hand, the third column in Table 6.2 describes a simple maintenance corrective project, where the application domain, the development environment and the costumer type are known, the documentation exists, the provider is in house and the duration is medium.

In this case a much simpler process is expected to be applied. Figure 6.7 shows both

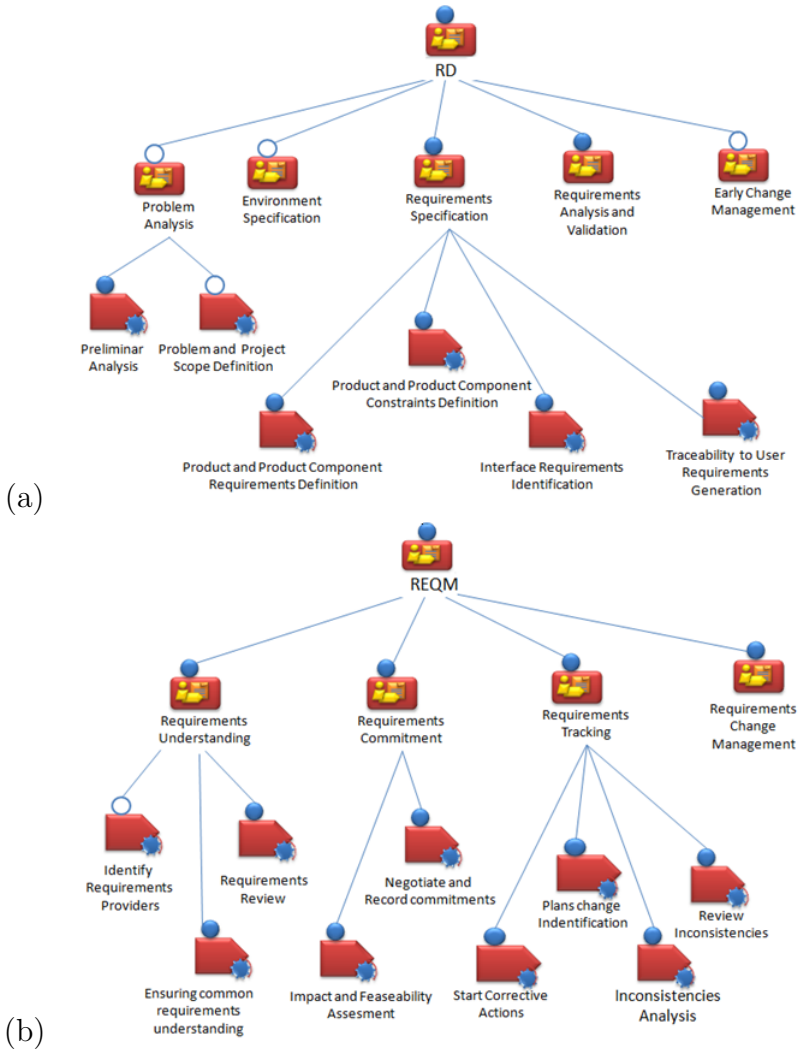

Figure 6.5: (a) Requirements Development and (b) Requirements Management Feature Models

the *Requirements Development* and the *Requirements Understanding* activities where some optional tasks have been removed from the context adapted process.

6.8 Tailoring Transformation

The tailoring transformation takes the general requirements process and a particular context model, and automatically yields a context adapted process. To this end particular rules are provided so that, according to particular values in the context dimensions, decisions could be made about all variation points identified as part of the Feature Model. Table 6.3 shows some of the directions included in the original adaptation guideline that were taken as a starting point for building the transformation rules.

Figure 6.6: Context Model

Table 6.2: Two project contexts

Context attribute	**Novel Development**	**Simple Maintenance**
Project type	New development	Corrective Maintenance
Application domain	Unknown	Known
Documentation	Does not exist	Exist
Provider	In-house	In-house
Development environment	Unknown	Known
Customer type	Unknown	Known
Project duration	Small	Medium

It is clear from the table that most common contexts are described and there is no ambiguity about the expected adapted process. For example, for *Maintenance-Correction* project type, the *Early Change Management* activity is never required. However, there are certain combinations of attribute values that are not defined. For example, for the case of providing *in house development*, the *Problem and Project Scope Definition* task could be required or not depending on the values of other attributes, but it is not clearly established. There are still other situations, like that happening when the *Source of Documentation* exists, where there is no clear action to be taken.

Moreover, there are situations (not shown in the table) where the action to be taken does not only depend just on the value of one attribute, and if there are two or more

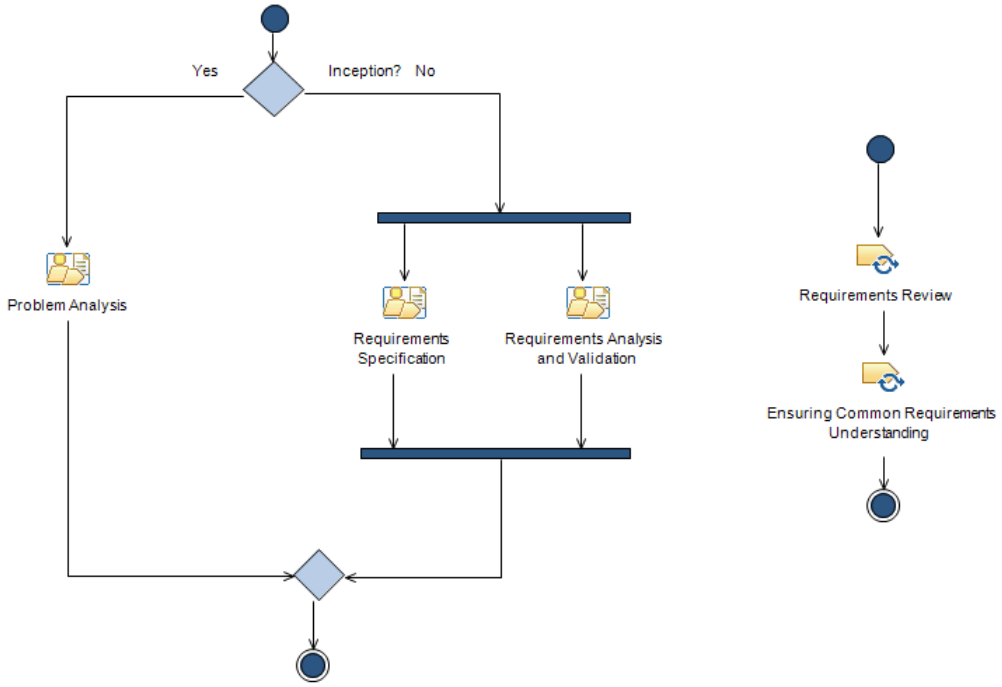

Figure 6.7: Requirements Development and Requirements Understanding for a simple Maintenance project

Table 6.3: Adaptation guidelines

Context attribute	**Value**	**Action**
Project type	Maintenance Enhancement	Problem and Project Scope Definition Task is required
Project type	Maintenance Correction	Early Change Management Activity is not required
Provider	In house	Problem and Project Scope Definition Task could be required
Provider	Outsource	Problem and Project Scope Definition Task is required
Source of Documentation	Does not exist	Environment Specification could be required
Source of Documentation	Exist	no action is suggested

attribute values that yield contradictory actions, priorities should be established. In general it is apparent the usefulness of counting on a tree based tool that enables us to compose complex rules by combining simple rules referring to different attributes and values in the context. Rule evolvability and scalability would be improved because the recursivity of the structure. Figure 6.8 shows an abstract tree of conditions on attribute values for determining the inclusion of the *Environment Specification* activity. In this case

the main rule could be used as an operand to compose a new and more complex rule. The following code shows the ATL implementation of the rule.

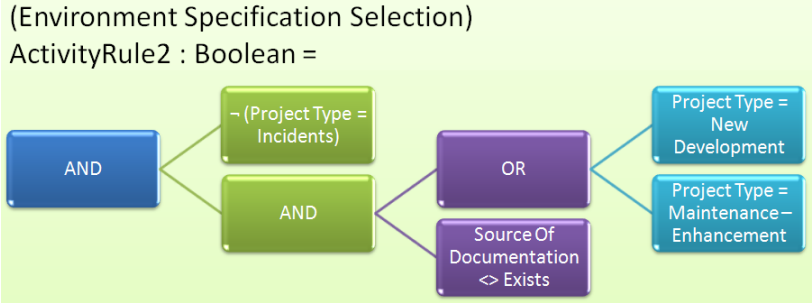

Figure 6.8: Attribute values for selecting the Environment Specification activity

```
–Rule 2 - Environment Specification Activity selection
helper def:activityRule2(elementName:String) : Boolean =
  if (elementName = 'Environment Specification') then
        if (thisModule.getValue('Project Type') = 'Incidents') then
                        false
        else
              if ((thisModule.getValue('Project Type') = 'New Development') or
                    (thisModule.getValue('Project Type') = 'Maintenance-Enhancement') then true
              else
                    if (thisModule.getValue('Source of Documentation') <> 'Exist') then true
                    else false
                    endif
              endif
        endif
  else true
  endif
```

Let us now consider the case where a project context is similar to that in the corrective maintenance (third column in Table 6.2), but now considering that the project does not have documentation available. Clearly this is a different case and there is no definition within the adaptation Table 6.3 that indicates the decisions to be made. In this case we define the project context as shown in Table 6.4, and we apply the rules, in particular Rule 2 just presented.

The obtained process will include the *Environment Specification* and *Identify Requirements Provider Task* that were not previously included, provided that the rule indicates that these process elements must be included whenever the documentation is not available

Table 6.4: Maintenance without documentation

Context attribute	Attribute value
Project type	Corrective Maintenance
Application domain	Known
Documentation	Does not exist
Provider	In-house
Development environment	Known
Customer type	Known
Project duration	Medium

(see Figure 6.9). According to the process engineer of the company, this is the expected result even though it was not explicitly stated in the adaptation guidelines.

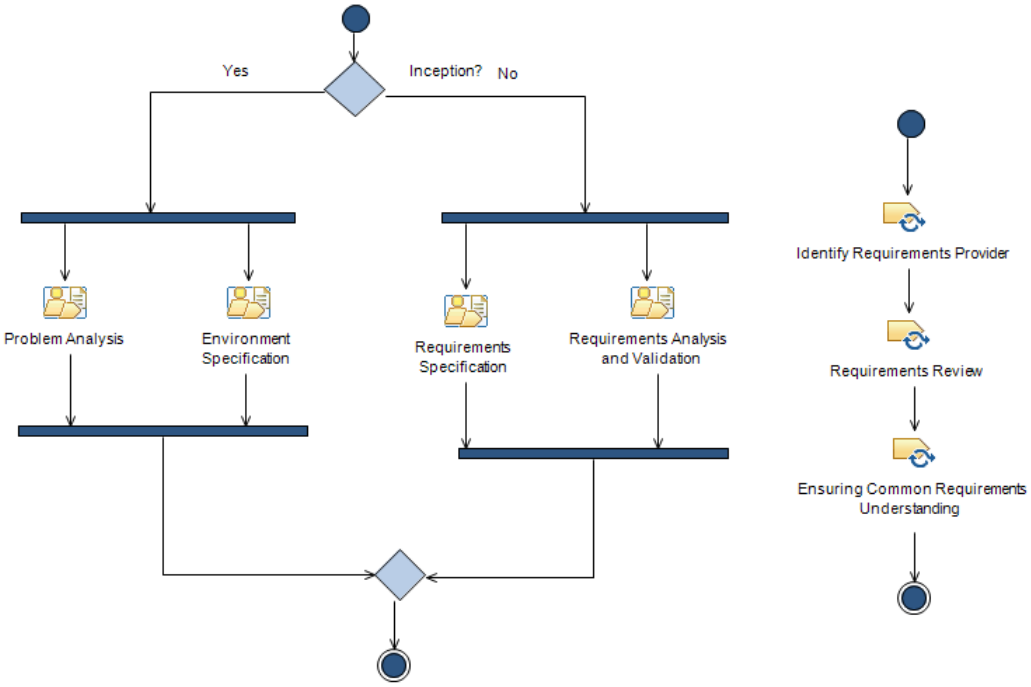

Figure 6.9: *Requirements Development* process in the case of non existent documentation

6.9 Case Study Results

The case study was developed by two external engineers, experts in process models and requirements engineering, according to the process available and the adaptation guidelines. The effort in the definition was 12 persons-hour for modeling the process, 6 persons-hour were spent to model the context and, 15 persons-hour were spent to implement the rules, with a total effort of 32 persons-hour. The process model was composed of 5 optional

elements producing 24 different processes (an optional feature was inside another optional feature giving 3 possibilities instead of 4 for this pair of the features). Furthermore, in the context model 6 context attributes (the attributes of the dimension Team and the Business attribute were not taken into account) were effectively used as part of the rules, resulting in 480 possible contexts. The coverage ratio between the available processes (24) and the possible situations (480) was of 0.05.

The MDE-based strategy was evaluated in a four-hour workshop including business, process and project management people from the company (12 persons-hour to present, evaluate and discuss the approach). In this workshop the technical work and a demo of the solution were presented including solutions of past projects and new possible project characterizations. Every possible adapted process was efficiently generated (5 minutes defining the specific context, less than 5 seconds generating the new process model - 0.1 person-hour) and collectively evaluated with the process engineer of the host company. The results indicated that the generated processes were correct and suitable for each particular project context (without tailoring errors). The organizational process was assumed to be already formalized, as well as the adaptation guidelines. However the tailoring effort was calculated using the adaptability effort used to define the context model and the tailoring rules distributed among 24 processes (plus the effective effort of tailoring - 0.1 person-hour).

Table 6.5: Metrics of KI-SPrL

Metrics	**Value**
CR - Coverage Ratio	0.05
TE - Tailoring Effort	0.98 Hours-Person
CE - Index of Cost-Effectiveness	0.051

To determine the cost-effectiveness of Casper in the case study respect to other approaches, a comparative study is required. For instance comparing with a single process model the coverage ratio is of 0.002. For a template approach, similar to the previous mechanisms in the case, considering four processes, the coverage ratio is of 0.008. Thus, the ratio between the available processes and the possible situations in the process line case was approximately 25 times the ratio for a unique process model approach and 6 times the ratio for a template approach (considering four process templates). In the case of a process framework (considering a full-flexible framework), the coverage ratio is 1.0 because each process model could be defined for each context. However, to the best of our knowledge the literature does not report quantifiable data about tailoring effort or other metrics, thus, establishing some comparison results difficult. Pedreira *et al.* (2007) analyze

the informality of both, approaches and studies. A qualitative comparison is established to compare two extreme approaches selected for this comparison. The approaches and their rationale are presented as follows:

- Template approach: selected because tailoring in a SPrL consists just in select a process model. It is less error prone error and faster. A typical case is Crystal Methodology (Cockburn, 2000) where 4 processes are included. The effort to generate 3 additional processes is distributed among the four processes.
- Framework approach: the tailoring in a SPrL requires to build processes from a defined infrastructure; it is more error prone and requires more effort than the template approach. However the coverage ratio is very High. A typical case is the Unified Process (Jacobson *et al.*, 1999).

The metrics take as reference the following scale: Very Low if the variable is not satisfied, Low if the variable is acceptably satisfied, High if the variable is satisfied to a great extent and Very High if the variable is well satisfied.

Table 6.6: Process Template, Process Framework and SPrL Comparison

Metric	CASPER	**Template**	**Framework**
CR - Coverage Ratio	High	Low	Very High
TE - Tailoring Effort	Very Low	High	Very High
Index of Cost-Effectiveness	High	Low	Very Low

The effort involved in generating the formalized organizational process with variabilities was low since it consisted in identifying the process elements affected by the adaptation. Writing the rules was more time consuming mainly because of the inherent ambiguity in the adaptation guidelines. Defining the context model took some time for analyzing the project specific situation, but defining a particular context only takes time to understand the specific project (this is the minimal effort to understand the context in any approach) and this step did not require the process expert's participation. These findings determine the suitability of the approach to make an adaptable software process and its adaptations. The return of investment will become clearer as more projects are tailored. So, this industrial case study shows that CASPER supported its process definition in a cost-effective way (High in the results) using only project specific information. Taking into account that the process model only included the requirements process and the context model was near completely defined, this ratio will improve significantly (at least 2 times for each new independent variation point) when the process model is completely defined (estimated at

least four major optional artifacts, it adds a minimum of four variation points to achieve a coverage ratio of 0.8). The requirements process line is effectively ready to be adapted to a significative number of situations.

6.9.1 Qualitative Results

Previously, during and after the workshop, some findings were extracted out of the participant's comments.

- The complexity for programming the tailoring rules was seen as problematic, because many rules used multiple context attributes and multiple process elements. The transformation developed in this case left a basic and reusable transformation infrastructure, so, the complexity could decrease in new cases. However a language such as ATL imposes unnecessary complexity to the process engineer. Even, define in abstract the tailoring rules is itself problematic.

- The environment to configure contexts did not result usable to the workshop participants, because the user interface used was provided by EMF, where a set of properties must be fixed.

- During the workshop, the process engineers found valuable the approach, it responded to their expectations, however they recommended adding a feedback loop. Therefore, the performance of the adapted process in the projects could be used to refine and evolve the software process line.

- The most experimented software engineer argued that the requirements process is the most problematic part of the process and that the approach could help to configurate many options, considering new and relevant context attributes than those considered by its original tailoring guide.

- The process engineers argued the benefits of the independence of the tailoring process. Tailoring in Casper only requires to characterize the project. So, knowledge about the process (and its variabilities and variants) is not necessary at tailoring time. Then, tailoring process does not depend on the experimented people.

- The process engineers advice on some unclear aspects of Casper. First, with respect to software process line evolution, many assets could become obsolete including the tailoring rules. The research group showed the Casper metaprocess, describing its iterative nature, thus, assets are dynamic elements. During application engineering

a jump to domain engineering has been established. Second, about when the adaptation task must be performed into the development project. Normally, the context is established when the requirements are understood, thus, the requirements process and tailoring process are interdependent activities. The research group argues that although CASPER follows a static approach, due to its simplicity and quickness, it could be applied many times until the project context is well understood. However, the process engineering group and the research group agreed that CASPER requires a better definition and implementation to consider this kind of scenarios.

6.9.2 Influence of the size of the process family in the Cost-Effectiveness Index

An observation of the case study is that the more variability have the process, a major CE is obtained. It is important validate this observation using new applications of CASPER approach. Therefore, this section extends the case study with a complete set of applications of CASPER available in the CASPER's web site [1]. The Table 6.7 shows the main results of this application.

- CC51A-RE: the course Software Engineering II (CC51A) process model at the University of Chile was used as a small initial but complete case to validate the suitability of the whole approach.
- Amisoft-TS: the process model of the application of CASPER approach to the technical solution process of Amisoft, a small Chilean software company. This company defined its organizational process in 2009, and it is currently implementing the ISO9001:2008 standard, as well as certifying CMMI Level 2. This case study is part of a paper submitted to the Journal of Systems and software.
- ISPW-6: the process model of the canonical example problem of the ICSSP conference (before ISPW) including the activity develop a change and the unit test. This case study corresponds to a paper accepted in the International Conference on Software and System Processes 2012.
- ID-UP: the unified process model, particularly, the implementation discipline model was used to define a small software process line. This case study was submitted to the Colombian Conference in Computer Science 2012.

[1] `https://sites.google.com/site/softwaremetaprocess/home/3-examples`

The results suggest that the size of the process family (derived of the feature process model) improve significantly the CE Index, therefore it implies that CASPER is more effective when the family grows.

Table 6.7: Index of Cost-Effectiveness For SPrL of Different Sizes

Metric	CC51A-RE	Amisoft-TS	ISPW-6	ID-UP	KI-RE
CR - Coverage Ratio	0.22	0.81	0.75	0.1	0.05
TE - Tailoring Effort	4.30	0.94	1.01	3.73	0.98
Index of Cost-Effectiveness	0.05	0.86	0.74	0.03	0.05
Process Line Size	6	52	24	8	12

6.10 Case Study Validity

This section analyzes the threats to the validity of the CASPER case study including construct validity, internal validity, external validity and reliability. This analysis was realized according to (Runeson & Höst, 2009) framework.

- Construct Validity: the capacity of the adapted software processes to fit many specific contexts (determined by the scope) is managed by the coverage ratio between the available processes and the possible situations. This ratio allows objectively to compare the approach with other strategies (the ideal case has N processes to N specific situations, ratio = 1, whereas the worst case has 1 process to N specific situations ratio = 1/N). The effectiveness of the rules is determined by the effort measured in person-hour. Although the measurement used really matched the research question, the comparison with other approaches is a threat to validity, because the other approaches, to the best our knowledge, do not count on quantitative data. Some data was approximated by analogy, however the comparison looses reliability. To partially improve this, a more discrete comparison was established to show that the approach has a better trade-of between the two extreme alternatives (respect to a good performance in some on the established metrics). For instance, tailoring is a simpler but limited template approach, hence, it requires less tailoring effort but its scope is small. On the other hand the framework approach is complex (tailoring requires more effort) but its scope is wider at the best cases. The study was extended using other applications in order to determine the influence of the size of the process family.

- Internal Validity: There are no intermediate collection of data processes or subjects that can introduce new threat factors to the internal validity. However, the coverage ratio is defined in a general way and it does not take into account the weight of each situation, so, a few processes that cover the most relevant or common situations (situation weight) could increase (or decrease) the coverage ratio. It applies to the template and framework approaches too, hence, this approach could improve the coverage ratio similar to the CASPER approach. New controlled experiments are required to establish a comparison more detailed including weights to each situation. The qualitative data obtained from different processes and the qualitative results extracted of the workshop, are data input directly obtained for analyzing the approach.

- External Validity: related work compares the CASPER approach with respect to others. Previous approaches are preliminary or complementary ways to adapt processes. The transformational strategy is not contrary to common sense in the literature (Armbrust *et al.*, 2009). The requirements process of a CMMI-Dev Level 2 medium size company was selected as study case because it has been the most complex part of the software process according to industrial experience in the Tutelkan project (Hurtado & Bastarrica, 2010). Furthermore, according to what is reported by the literature (Pandey *et al.*, 2010), requirements engineering for software development processes is a complex activity that considers a big number of viewpoints, roles, responsibilities, and objectives. Thus, if CASPER results cost-effective when it is applied to the most complex part of a process, will CASPER be cost-effective on the whole software process model as well? this question remains open. Additionally, if CASPER is cost-effective in a case with a medium size organization where the process has a typical size, will CASPER be cost-effective in large settings where the process will count with more definitions, tailoring guides and experiences? The case of a small setting was developed in the CC51A Requirements Process case presented in Chapter 4 showing its feasibility in this case. With the aim to generalize, new applications must be realized, using other contexts, sizes and organizations to establish a more general conclusion. However, according to the results applying CASPER to four new processes, allows us to establish a stronger conclusion about the CASPER approach. A threat of external validity is the absence of data to establish a direct comparison with the related work.

- Reliability: The research question stated, for this study case, is if CASPER is a cost-effective technique. Both, comparison and workshop coincide with each other in this study case. Process engineers found effective the approach because it does not

requires expert knowledge and it is less error prone. The comparison estimates the best situations for the other approaches (four process templates is a high number and a flexible process framework to produce any derivated process model), so the comparison was developed using ideal competitors to Casper approach. However, more controlled experiments must be conduced to achieve a better quantitative and qualitative comparison, allowing to establish how good Casper really is with respect to various approaches. The information obtained using the workshop is not replicable. Other study cases could be realized to generalize the conclusions obtained using this technique. The technical part of the study case following the Casper approach could be considered as repeatable; the environment and process models are available, they are concrete and replicable. The adaptation was executed according to what was expected; models, meta models and the ATL rules are the main concrete elements to repeat the experience.

6.11 Synthesis and Discussion

In this chapter an MDE approach to define software process models from a software process line has been proved in an industrial environment (Hurtado *et al.*, 2011c). A context model including dimensions and context attributes has been defined using a Software Process Context Meta model following the ideas in Hurtado & Bastarrica (2009), and Kajko-Mattsson (2010). The process model variability has been represented using a process feature model similar to software features models (Kang *et al.*, 1998) and the process models have been implemented using SPEM 2.0. The MDE strategy was effectively implemented to achieve a separation between the process modeling stakeholders and process enactment stakeholders (project stakeholders) (Bai *et al.*, 2010) and it hides the complexity by intensively reusing tailoring knowledge (Hurtado & Bastarrica, 2009). Furthermore, for the case study presented, the MDE tailoring strategy provides a way to cost-effectively instantiate a general process model into a project-specific process model where the project manager should only provide a definition of a specific context.

This approach has the potential to enable improvement with respect to project productivity and quality of the resulting software products. Provided that the adapted process will include all process elements that are required for the particular project context, no extra work will be needed and only the essentially required effort and resources will be spent. In addition, high quality work products can be expected, because the process is adjusted with this goal in each particular project context. Since this tailoring process is automatic, and it eventually uses already validated transformations, it is expected to

achieve a reduction of the tuning time and cost, and also the number of adaptation errors (Hurtado *et al.*, 2011c).

Chapter 7

AVISPA Validation

7.1 Introduction

This chapter reports the empirical application of Avispa. According to the defined research method, a case study (Runeson & Höst, 2009) was used as validation technique covering a wide spectrum of process types and scenarios. Avispa has been applied for analyzing the process models defined in three different Chilean software companies. Avispa was applied to find several of the defined error patterns, and most of them resulted in actual errors, giving support to the hypothesis that a visual tool that helps the process engineer is effective to analyze software process models. Avispa has been highly welcomed in all companies we have worked with, and process engineers also pointed out that it is relevant for them to count on Avispa for maintaining their software process model, an application that was not initially envisioned. In this chapter this experiences are reported.

7.2 Preliminary Validation

Before exposing the proposed tool and approach to the industrial cases, a set of preliminary validation issues were conduced to confirm its consistency, feasibility, and applicability. The preliminary validation included:

1. Avispa tool prototype described in chapter 5
2. Avispa initial cases with EPF community software process models
3. Initial industrial case with Software Process Blueprints

The following subsections provide details on the particular forms of validation and they present the obtained results.

7.2.1 AVISPA initial cases with EPF community software process models

AVISPA was incrementally implemented and used with small process models. First a test process model was created, but to evaluate the approach, known and available process models were required. So, EPF process models available in the EPF web site[1] were used, particularly XP, Open/UP and Scrum process models were used to validate the blueprints, the metrics, the views and the patterns. XP is an incomplete specification of the process where no tasks are specified (because of its agility), so it was not very interesting to analyze it with AVISPA. Scrum was a relevant work to refine the problem patterns; this case was shown in the Chapter 5 where it is used as an example to illustrate the identification of problematic patterns with AVISPA. The experience with Scrum conduced to the validation of the metrics, visualizations and problem patterns in an incremental way. So, the tool was iteratively improved where the Scrum process model was used as the simple but complete case to evaluate model consistence and usability (easy interpretation of the visualizations) (Hurtado *et al.*, 2010a).

7.2.2 Initial Industrial Case using Software Process Blueprints

In Chapter 5, DTS's development process was presented as an illustrative case. As part of its SPI project DTS wants to assess the quality of its process, so they used the Process Model Blueprints (the initial version of AVISPA) (Hurtado *et al.*, 2010b). The problems found were used as input for the following improvement cycle in DTS and to introduce in the AVISPA tool the problem patterns. Even though some evaluations such as CMMI appraisals were included, there was still no information about the quality of the process model itself or the adequacy to the company's goals. The evaluation process was carried out by two process engineers, one from DTS and another one from the University of Chile, and two software engineers users of the process definition. The steps followed were:

(1) Visualization: the process engineers imported the model from EPF and they realized a short review. Some errors in the tool were identified and some aspects were tuned up. A first visualization of the blueprints was generated.

(2) Potential problems identification: for each blueprint problems were identified. Each problem was recorded for further analysis. However a trivial analysis was realized in this step. Table 7.1 presents the set of problems found in DTS's process model using blueprints; false positives have been discarded.

[1] CASPER tool website http://www.eclipse.org/epf/

Blueprint	*Identified Problems*	*Quantity*	*Improvements*
WorkProduct Blueprint	Few guidelines, some are very long, some are disconnected, many initial and final artifacts	31	23
Task Blueprint	Tasks without roles, disconnected tasks, many initial tasks, many final tasks, many relationships big tasks, deformed tasks	22	19
Role Blueprint	Roles with light load, roles with weight load, roles isolated, there are few guidelines	7	5
Total		60	47

Table 7.1: Process Model Problems Identified with PMBlueprints

(3) Problem analysis: each blueprint was also used to find a symptom of a problem and the related elements. For example, the analysis did not have relationships to *Client* and *Engineer Chief* roles, so it was possible that some tasks where these roles participated were not adequately specified. Some problems required more analysis and discussion with users of the software process.

(4) Data collection: the potential and actual problems were collected, and also some solutions found were reported.

(5) Data analysis: the data collected in this study case was analyzed and the main results are presented below.

Table 7.1 summarizes the main problems found during the evaluation of DTS using Software Process Blueprints. For a total of 132 process elements, 47 actual problems were found. So, the process model error rate was of 0.356 errors per model element. In general, the review determined that there were few guidances associated to model elements. This situation suggests low formality in software process specification. Disconnected elements caused modeling problems: some connections were omitted at modeling time, but others caused serious conceptual problems in the software process. For example, the isolated *Client* role was a symptom of a problem, and actually a big problem: the client was not involved in many tasks where his/her participation is required, specially in requirements tasks. Two kinds of improvements was obtained:

Improvements to the Software Process: the main suggested improvements are: increase guidance for roles (guide of work) and for work products (templates, examples and concepts); task re-design for getting tasks of medium size (neither too large nor too small), such as *Requirements Review*; big work products decomposition such as *System Requirements Specification*; decrease coupling between process areas with fewer relationships between work products and tasks; increase the *Client* participation and improve the collaboration required between the *Analyst* and *Engineer Manager* roles. The bottleneck risks identified on the *System Requirement Specification*, *Alpha Component* and *Alpha*

Blueprint	*Improvements to Specification*	*Improvements to Process*
WorkProduct Blueprint	19	4
Task Blueprint	16	3
Role Blueprint	2	3
Total	37	10

Table 7.2: Improvements suggested with PMBlueprints

System Cycle n work products are normal bottlenecks because of the incremental nature of the software process, so these work products are completed and refined cycle by cycle during the software process. However the *Architecture* work product and its associated tasks are not adequately contextualized in this approach (it is related to the requirements area but it is isolated with respect to incremental development tasks and work products). So, the architecture knowledge, including its associated tasks need a strong analysis for supporting a better incremental development.

Improvement to the Software Process Model Specification: the main improvements to software model specification are about missing nodes, edges and guidances. Several problems were identified directly, others required more analysis as collaboration between specific roles, and others were generally due to conceptual problems of the software process. These types of problems were the most frequent ones.

7.3 AVISPA Case Study

According to the research method defined in Chapter 1, a positivist and explanatory industrial case study has been used as the technique for validating Avispa. It is positivist because it seeks to test the hypothesis and it is explanatory because it seeks an explanation of a situation where Avispa is applied as both a new analysis approach and a tool. This Avispa case study is embedded (Runeson & Höst, 2009) because it includes the analysis of three complete process models of three small size Chilean software companies: Amisoft, BBR Engineering and DTS (Hurtado *et al.*, 2011a). In order to be able to compare results we chose to analyze the three processes according to the same error patterns: disconnected subgraphs in the Task Blueprint and tasks with too many output work products. Also, disconnected subgraphs could have been analyzed in the Work Product Blueprint. The process models used in this research were developed in the last two years; these models were obtained from the respective libraries using the exporter feature of EPF. The process models were analyzed in the MaTE (Model and Transformation Engineering Group) Laboratory and then the results were discussed with the respective process engineers.

7.3.1 Research Question

Is it possible for a non expert process engineer to early analyze the quality of the process model? The term *early* refers to analyze a process model before it is enacted and executed on a specific project. AVISPA has demonstrated its power to analyze software process models. However this study case tries to validate if AVISPA enables the process engineer to analyze the software process model quality in an easy way before this model is enacted.

7.3.2 Case Study Selection

The research object is the software process model. In this case, real, complete and formalized software process models have been used to assess the AVISPA effectiveness for identifying some specification and conceptualization problems. A set of three software companies were selected to conduct this embedded case study. The companies' contexts match the research objective.

7.3.3 Case Study Context

AVISPA was applied on three Chilean software companies: *Amisoft*, *BBR Engineering* and *DTS*. The characteristics of each company are described as follows.

Amisoft is around ten years old and it is formed by thirteen qualified employees. Its main goal has been to deliver specialized and quality services. Its development areas are: client/server architecture solutions, enterprise applications based on J2EE and Systems integration using TCP/IP and MQ Series. Amisoft has started its software process improvement project in 2009, and it is currently implementing the ISO9001:2008 standard and the CMMI model. Its software process model has been inspired by OpenUP.

BBR Engineering is one of the main software factories of BBR, an international consulting company since 1994. It is formed by twenty four employees specialized in different roles including architects, project managers, developers, quality assurance specialists and analysts. BBR Engineering has developed solutions mainly in the area of retail; specifically, its main areas are: points of sale, payment systems, communications and interfaces, e-business, and integration. The company has started its software process improvement project in 2009 using the Tutelkán Reference Process as a reference for its implementation.

DTS enterprise was described in the Chapter 5 for introducing the blueprints.

7.3.4 AVISPA Case Study Results

Amisoft. In this software company, AVISPA helped to identify 5 instances of the pattern *independent sub projects* corresponding to the nodes with a color different than

blue in Figure 7.1. These nodes represent the tasks: *Configuration Items Update*, *Non-Compliant Communications*, *Delivery Document Generation and Sending*, *Getting Configuration Items* and *Execute Unitary Test to Interfaces and Communications*. These disconnected subgraphs (in this case disconnected tasks) represent a high risk because the configuration management process could be chaotic (everybody needs to know how to get and put configuration items) and the testing of interfaces and communications could be forgotten just when it is required the most. Looking for independent sub projects in the WORK PRODUCT BLUEPRINT is a dual case: whenever there are errors in one, normally, there are also errors in the other. This pattern facilitates to find isolated work products too: *Directory Structure, Case Test Template, Client Satisfaction Survey and Glossary*. These work products corresponding to the nodes with a color different than yellow in Figure 7.2. These work products are not adequately linked with the rest of the process elements being this ambiguous for the process users.

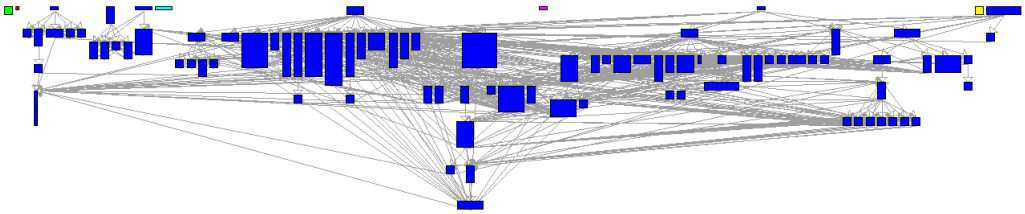

Figure 7.1: TASK BLUEPRINT for localizing disconnected subgraphs in Amisoft.

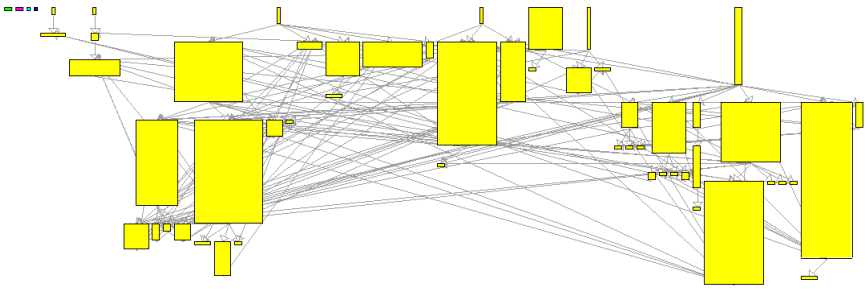

Figure 7.2: WORK PRODUCT BLUEPRINT for localizing disconnected subgraphs in Amisoft.

When the *multiple purpose task* pattern was applied on *Amisoft Process Model* (red nodes in Figure 7.3), the result was 9 potential errors of multiple purpose tasks out of 93 tasks in total (9.7%). However, reviewing these tasks, many of them refer to management tasks where different inputs are required to evaluate the project advance or to make some decisions, and as a result these tasks modify many work products. So, the

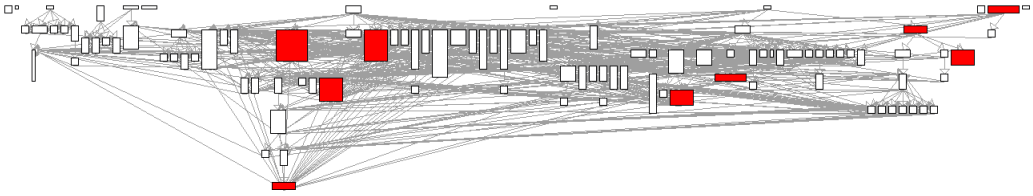

Figure 7.3: Task Blueprint for localizing multiple purpose tasks in Amisoft.

granularity cannot be finer, but more guidance could be added. However, the task *Document Requirements* could have been better decomposed into two different tasks separating abstraction levels (concerns covered or users of the requirements). On the other hand the task *Measure Data Collection* is shown as a *multipurpose* task and it really is, because the measurement results are not available directly in a unique work product; instead of this, the data is available in many work products according to the metrics established in the measurement plan.

BBR Engineering. Similarly, Avispa was applied for finding disconnected graphs and multiple purpose tasks to the process model of *BBR Engineering*, and the results are shown in Figure 7.4, Figure 7.5 and Figure 7.6, respectively. Many tasks were found disconnected, 29 out of 79 (36.7%), and this situation shows that the process presents many underspecifications, increasing the risk of not applying it as intended. Most of the problems are related to project management and configuration management. The configuration management issues reveal the process immaturity in BBR Engineering contrary to project management. However, both process components must be specified with more precision. Nevertheless, there are some tasks underspecified which could be problematic when the process is instanciated: *User Needs Understanding, Requirements Priorization, Measure Data Collection, Unit Data Base Testing, Unit Component Testing* and *Interfaces and Communication Testing.*

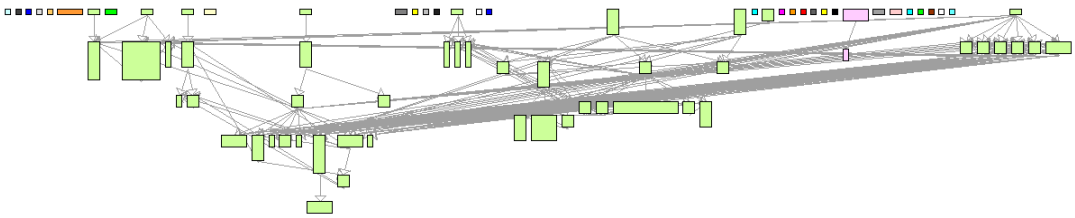

Figure 7.4: Task Blueprint for localizing disconnected subgraphs in BBR Engineering.

When the independent projects pattern was also applied on the Work Product Blueprint using Avispa, similar problems arose on project management, but excluding

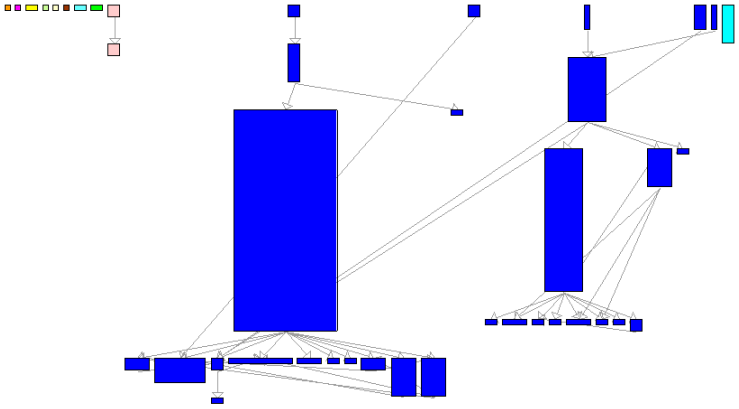

Figure 7.5: Work Product Blueprint for localizing disconnected subgraphs in BBR Engineering.

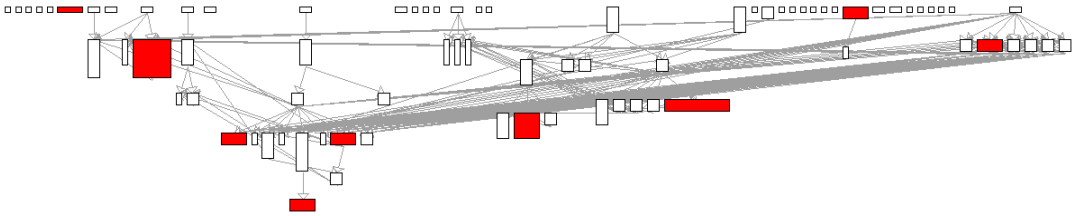

Figure 7.6: Task Blueprint for localizing tasks involved with too many work products in BBR Engineering.

these general problem, the problematic work products were: *User Interface Model, Integration Plan* and *Design Document*. This process would inject many technical problems at instantiation time because it includes many imprecisions in tasks and work products of requirements, technical solution, testing and project management areas. The process in BBR includes 79 tasks, and 9 of them are identified as problematic (11.4%) according to the multiple purpose task pattern (red nodes in Figure 7.6). Similarly to the previous case study, the project management task defines many outputs and for the same reasons cannot be changed. However, the tasks *Data Base Design* and *Component Design* could be decomposed to reach a homogeneous process model definition.

DTS. Avispa was also applied to the process of DTS for identifying and localizing both kinds of error patterns (see Figure 7.7, Figure 7.8 and Figure 7.9). Only two tasks were found disconnected showing a careful specification job: *Identify Requirements Provider* and *Change Requirements Reception*. These tasks are critical to manage requirements change, so this part of the process would not be instantiated adequately at projects. This analysis is consistent with the obtained previously in (Hurtado *et al.*, 2010b). The

process in DTS includes 57 tasks, and 6 of them were found to be multi purpose tasks (10.5%) painted in red in Figure 7.9. Similar to the previous cases, most of these tasks are part of the project management and this characteristic cannot be changed. But, the *Generating Implementation Document* and *Requirements Review* tasks could be decomposed, whereas *Help Diagram Development* could be specialized for each specific help diagram to be designed.

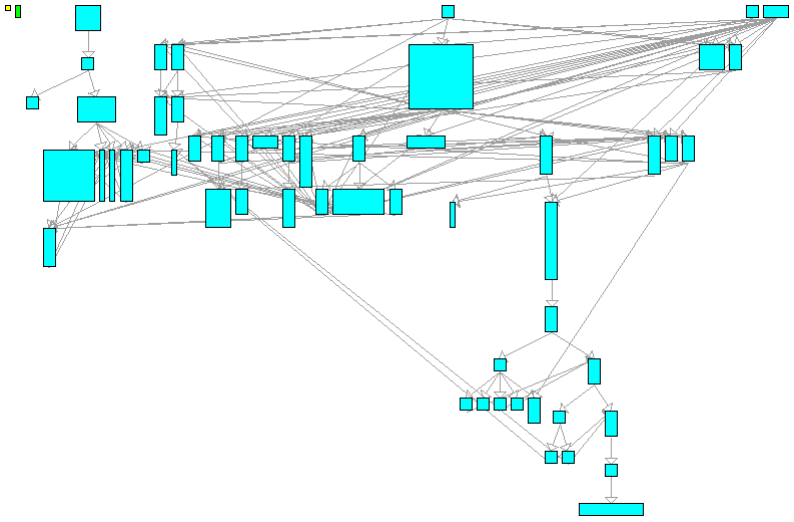

Figure 7.7: Task Blueprint for localizing disconnected subgraphs in DTS.

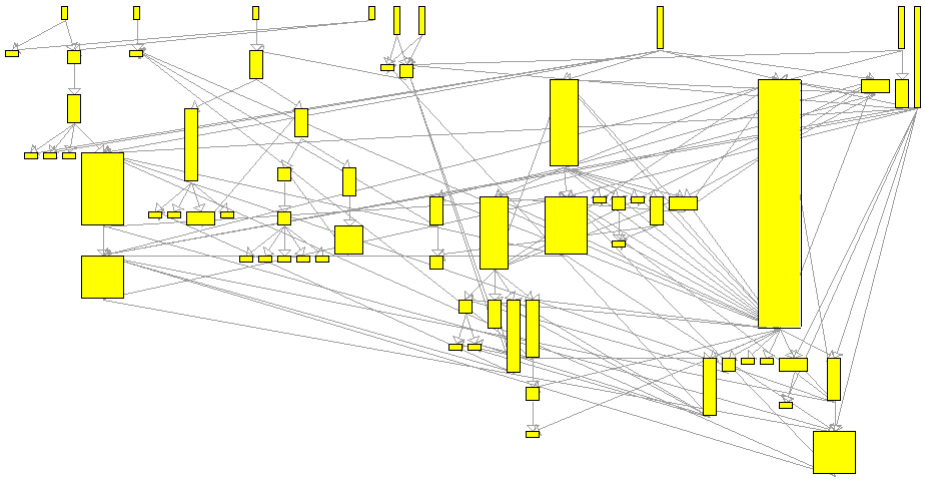

Figure 7.8: Work Product Blueprint for localizing disconnected subgraphs in DTS.

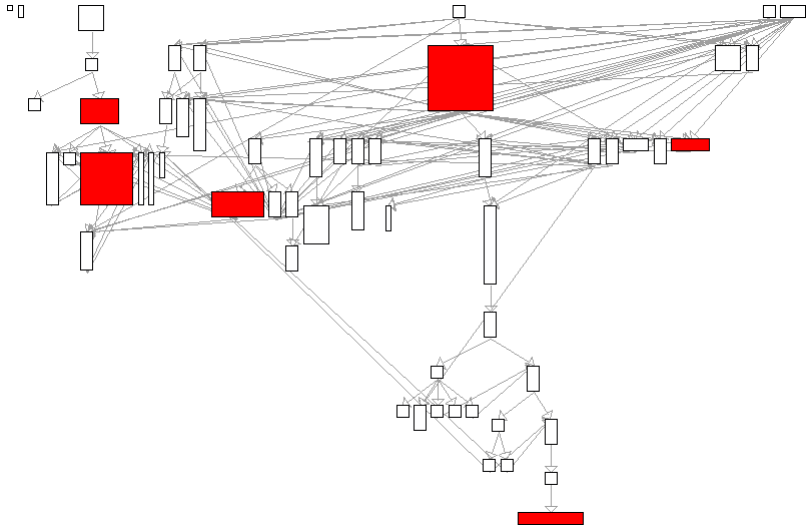

Figure 7.9: Task Blueprint for localizing tasks involved with too many work products in DTS.

7.3.5 AVISPA Case Study Results Analysis

This section analyzes the results in order of establish valid conclusions on the Avispa approach. In general, Avispa is able to identify and localize problematic elements in three blueprints. However, it is also important to evaluate the data to delimit Avispa capabilities. First the false positives are analyzed and then, the tunning of the patterns according to the F-Measure is analyzed.

7.3.5.1 False Positives Analysis

The *No guidance associated*, *Isolated roles* and *Waste* error patterns always identify specification errors (also highlighting conceptual errors), so, false positives are not analyzed for these cases. *Overloaded roles* error pattern does not define specifically when a role is overload, it only defines an overload level, the actual overload is subjectively determined by the process engineer. For the rest of the error patterns a false positive analysis has been included. The Tables 7.3 and 7.4 show the relationships between the identified problems and the actual problems in order to determine the effectiveness of the Avispa approach. According to these results the *Independent Subproject* was very effective in the study case with only 5.4% of false positives. On the other hand the *multiple purpose tasks error* pattern had a lower effectiveness with 54.1% of false positives. The *Demanded work products* pattern was not validated in this case study because this pattern was not implemented when the case study was realized. This pattern still requires a validation similar to *multipurpose task error pattern*.

Table 7.3: False Positives in the Independent Subprojects Error Pattern

	Identified Problems	Actual Problems	Percentage of False Positives
BBR	28	28	0
Amisoft	6	5	16
DTS	3	2	33
Global	37	35	5.4

Table 7.4: False Positives in the Multiple Purpose Error Pattern

	Identified Problems	Actual Problems	Percentage of False Positives
BBR	9	3	66.6
Amisoft	9	3	66.6
DTS	6	5	16.6
Global	24	11	54.1

In order to validate the *Waste* error pattern, it was applied independently to the DTS process. In order to evaluate the relevance of the waste work products, the blueprint's results were evaluated with the DTS process engineer. From the 22 highlighted work products found, 3 have been confirmed to be non specified deliverables, 16 underspecified task inputs, and 3 are indeed waste work products. As a summary, the practical effectiveness of the tool has been confirmed with DTS by identifying 22 problematic work products: 13.6% of deliverables that had not been identified, 31.9% underspecified tasks (where the work product should be an input), 13.6% waste and 40.9% could be improved (refactoring and integration). Hence, in well-specified software process model for DTS, only the actual wast would be detected.

7.3.5.2 Pattern Tuning

In these case studies, AVISPA has been applied to validate its effectiveness to visually identify problematic elements in software process models. The F-Measure (Van Rijsbergen, 1979) has been used to evaluate the effectiveness of the patterns because it recovers problematic elements in software process models but not all were actual errors. The F-Measure is obtained by identifying two types of sets: recovered potential problems (retrievedElements) and the actual relevant problems (relevantElements). The recall (R - fraction of the process elements that are relevant to the query that are successfully retrieved) and precision (P - the fraction of retrieved process elements that are relevant to the search) metrics are calculated according to the formulas:

Table 7.5: F-Measure including all the process models

	1-sigma	2-sigma	3-sigma
BBR	0.50	1.00	1.00
Amisoft	0.50	0.17	0.33
DTS	0.91	0.44	0.29
Average	0.64	0.54	0.54

$$R = \frac{\|\{relevantElements\} \cap \{retrievedElements\}\|}{\|\{relevantElements\}\|} \tag{7.1}$$

$$P = \frac{\|\{relevantElements\} \cap \{retrievedElements\}\|}{\|\{retrievedElements\}\|} \tag{7.2}$$

$$FMeasure = 2 * \frac{P * R}{P + R} \tag{7.3}$$

The analysis is focused on two patterns: independent projects and multipurpose tasks. The first pattern resulted very effective because most of the identified problems were actual problems: its F-Measure is of 0.96. The second pattern is not as effective as the first one. It was tuned to achieve a larger benefit using as reference unit a standard deviation (sigma) and three cases were analyzed: using 1-sigma, 2-sigma and 3-sigma as problem identification filters. Tables 7.5 and 7.6 show the F-Measure applied to the process models.

Table 7.5 includes all processes included in the case studies. The BBR process had a particularity: only one problem was identified and the problem was identified in the three cases; as a result, the average F-Measure is similar in the three cases although it was slightly lower for 1-sigma. Table 7.6 excludes the BBR process because a unique identified problem makes it less reliable, thus, using the two processes with major error reporting (DTS and Amisoft) the F-Measure was considerably higher for the case of 1-sigma case (two times the others). So, the multipurpose task pattern was tuned to 1-sigma: each task positively deviated 1-sigma from the average number of outputs will be identified as a potential error of multipurpose task.

7.3.6 Qualitative Results

Previously, during and after the work meeting with the process engineers, some observations, comments and results came out:

Table 7.6: F-Measure including the most relevant process models

	1-sigma	2-sigma	3-sigma
Amisoft	0.50	0.17	0.33
DTS	0.91	0.44	0.29
Average	0.70	0.31	0.31

- Previous to the evaluation meeting, when a demonstration of AVISPA was performed. The all process engineers requested more time to debug the software process model.

- Each potential error was identified by the research team with the process engineer. The process engineer evaluates each potential error localizing the problematic process element in Eclipse Process Composer. Then errors identified and classified by the process engineer were analyzed by the research team. This interaction added reliability to the analysis made by the processes engineers. Amisoft and DTS reported more evaluations with AVISPA after of the case study realization.

- During the process analysis meeting, AVISPA was highly welcomed. It responded to the expectations. The process engineers found the tool practical, intuitive and effective. In Amisoft, where three process engineers participated, the tool was incorporated as part of its software process improvement infrastructure.

- The process engineer recommended to include a top-down structure to analyze processes, because they considered that the follow-up analysis could be focused on a specific process area because with a big process the relationships make the AVISPA blueprints less clear.

7.3.7 Case Study Validity

This section analyzes the threats to the validity of the AVISPA case study including construct validity, internal validity, external validity and reliability. This analysis was realized according to Runeson & Höst (2009) framework.

- Construct Validity: the used metrics count elements related to the measured element and considered statistical issues for deriving some object information. These metrics are presented in a visual way to aid the process engineering to analyze the process model. The patterns have been derived for the last five years while implementing processes in small software companies in Chile. Many alternatives for the

visualization of scripts and for statistical parameters were evaluated in the preliminary cases to calibrate the AVISPA tool to facilitate a consistent data collection and visualization. The *No Guidance Associated*, *Isolated Roles* and *Waste* error patterns always identify specification errors (and also conceptual errors) whereas the *Independent Subproject* presents a 5.4% of false positives. The *multiple purpose tasks error* pattern had a low effectiveness with 54.1% of false positives. *The Waste and Demanded Work Products* error patterns require more validation.

- Internal Validity: the process analysis with AVISPA is still dependent on the process engineer experience, intuition and subjectivity. The results, in particular the actual problems in the processes were mainly determined by the process engineers. However, the problem analysis, resolution and determination of false positive was carried out working together the process engineer with the research team. Thus, errors, improvements suggested and false positives were identified and validated.

- External Validity: process model analysis is normally realized through model checking, metrics or simulation. This approach is comparable to other approaches and its applicability is different from them. Any process model defined in EPF and exported as a XML file can be analyzed with AVISPA in a similar way as the analysis presented in this case. However, some errors detected could not be considered as actual errors by a process engineer according to her/his experience and the organization context. So, AVISPA could only assist to an process engineer to visually analyze her/his software process model.

- Reliability: False positives were identified in order to determine a reliability level. Because patterns compute and highlight recurrent problems, this case study can be replicated with the same process models and the results of the blueprints will be exactly the same. However the actual errors reported will change according to the process context, the process engineer participation and the specific pattern. Some error patterns are more reliable than others, according to the results of this study case. However it is not possible to generalize the metrics and patterns; more experimentation is required for achieving an objective generalization.

7.4 Synthesis and Discussion

A set of error patterns has been implemented as part of AVISPA, a tool for process model analysis that localizes a set of identified potential errors within a process model specified in SPEM 2.0. These errors may come either from process conceptualization or from

misspecifications. Process models of some of the Chilean companies we were working with for the last years have been analyzed using AVISPA to validate their applicability. Some errors were found, as well as some improvement opportunities, showing the effectiveness of the patterns and the AVISPA tool. These errors were not foreseen by process engineers, but they agreed they were real improvement opportunities. The quality of the analysis highly depends on the quality of the definition of the error patterns. Even though the error patterns presented in this chapter have shown to be effective in finding improvement opportunities, there is some room for fine tuning them. For example, determining that a task is too complex if it is more than one standard deviation from the mean, may not help discriminating really badly specified elements, and maybe two standard deviations is a better measure. We may also define the error pattern as parametric in the number of standard deviations considered. As part of the practical experience, some typical recurrent errors in software processes have been identified, some of them due to conceptual errors in the process design and other errors introduced during process specification. But none of them are easily identified, let alone localized, because of the enormous amount of process elements involved, multiple views, and informal notations that may sometimes introduce ambiguity.

Chapter 8

Conclusions, Contributions and Limitations

This chapter presents the main contributions of this thesis, the goals are reviewed, the main results are analyzed, some limitations are identified and further work is delineated.

8.1 Goals review

At the beginning of this thesis, some goals were defined. The subsequent items explain how these goals were achieved:

1. In order to build a cost-effective mechanism for software process tailoring, an MDE approach was proposed as a SPrL production strategy. The SPCM meta-model was defined to specify project specific contexts, a subset of SPEM 2.0 meta-model was used to specify software process models and the ATL language was used to implement the tailoring decisions. In each study case this infrastructure was used and completed with specific case study information. Chapters 4 and 6 show some scenarios where a cost-effective tailoring mechanism is possible using the proposed techniques. The technique shows its effectiveness, although more case studies will be necessary to conclusively demonstrate that the cost (effort) has been decreased and to express with certainty how much it was. So, an MDE technique is a feasible and practical approach to support software process tailoring under the SPrL approach.

2. To build a practical way to verify a software process model. This thesis proposed as another hypothesis that a visual approach could be used to analyze software process models in a practical way. The Avispa tool was developed and applied to many processes; among them there are three real software process models presented as a case study in Chapter 7. With this case study, the hypothesis was demonstrated and

the objective was achieved. The visual approach resulted useful to analyze software process models provided that a set of the error patterns were used to show that the exhibited errors effectively were misspecifications or conceptual errors.

8.2 Main Contributions

The main contributions of this thesis are:

1. According to the literature review performed in this thesis, to the best of our knowledge, there are no other reports presenting practical results where MDE is used as a tailoring strategy. In this thesis an MDE strategy of production in a SPrL has been applied to real cases showing its effectiveness. It may guide the research community to introduce more sophisticated mechanisms to produce SPrLs in a more practical way. Particularly, models and tailoring rules should be presented in a usable and practical way to achieve a massive application in the software industry and to get more empirical evidence about this approach and its consequences. In the Chilean case the ADAPTE[1] project funded by Fondef - Chile is heavily based on this thesis as a starting point to create and apply novel mechanisms to support context-adaptable software process models.

2. This thesis contributes to the process research community with Casper, a meta-process to build software process lines. The MDE based tailoring strategy results useful when the general software process model includes variabilities and these have been mapped according to specific context values. Therefore, the tailoring rules are possible because the reuse and adaptation of the process model has been defined in a planned way. For example, a process element can be deleted at tailoring time for the complete process model to remain consistent in the tailored process model. The meta-process Casper adds a concrete methodology to the software process lines literature. So, new methodologies could be created complying with technical and managerial requirements or new meta-process assets (meta-models and rules) could be created or evolved to strengthen this approach. As a result, tailoring rules design is addressed as a process engineering activity whereas tailoring execution as a software engineering activity.

3. It is the first case where the context is formally defined and used as an independent and formalized process asset. To express contexts, a simple meta-model has been

[1]Adaptable Domain and Process Technology Engineering http://www.adapte.cl/

defined enabling the specification of relevant context dimensions and their context attributes and context attribute values providing flexibility to the organizations for defining their own tailoring drivers. This context meta-model contributes to the literature with the idea that the context can be formally specified, so an interesting line could be oriented to asses if this meta-model in different application cases will support context definitions. Additionally, new proposals of context models could be developed according to this assessment and even a standard meta-model could be proposed and defined. So this thesis has contributed to consider the context as a first order citizen in software process models.

4. The software process verification task was exhibited as a practical activity using the power of visualization. Software visualization has shown to be practical to analyze software models. Based on the idea that software processes are software too, visualization was applied to process models showing its power too. It opens a great area where software process models and particularly complex software process models could be analyzed from different points of view. This thesis has proposed some of these views as blueprints; however, other blueprints could be defined. For instance, performance and, modifiability could require measurement and visualization of the process from other perspectives. Additionally, Avispa's analysis is limited by the process model language; so, improvements to process meta-models should be suggested if more assessment views are required.

5. The error patterns found were implemented in Avispa for supporting the findings that the process analysis could be assisted in a practical way by a visual tool indicating what could be wrong in the model. Error patterns are a starting point to establish practical paths about how a process model could be assessed. The term *practical* refers to assessing the quality of the process model according to organizational, project and team needs, beyond a syntactic, semantic or standards-based assessment.

8.3 Conclusions

As process technology is largely based on models, model-driven is naturally a suitable approach (Lonchamp, 1993). Thus, the paradigms and schemes of model-driven engineering could be applied to process engineering. This thesis has addressed in general the hypothesis that the application of MDE in the definition and validation of software process models is suitable. The definition uses the MDE approach as a context-oriented

adaptation strategy, while validation uses an MDE strategy to retrieve a process model to a visual-oriented analysis environment. Both approaches have proved feasible and practical, allowing the process engineer to increase the value and the reliability of the process models. Process models explicitly express how the organizations and teams work, so they these are key knowledge assets. However, the value of these assets depends on their usefulness and quality. Process model quality attributes addressed in this thesis have been the suitability and correctness, showing that it is feasible to address them using an MDE approach. Several studies presented in Chapter 2 show different approaches dealing with relevant properties related to tailoring or analysis of the software process models, however there is no quality framework for process models which allows to establish the relevance of referential measures and approaches to objectively compare the results as presented in this thesis with others works. In addition to determine the properties of process models, it is necessary to replicate the experiences reported in this thesis to extend and generalize the findings obtained in its study cases. The potential of the approaches together was not addressed to maintain independence to validate the hypotheses, but in general a traditional tailoring approach or the method engineering approach could be benefited with the approach of Avispa. Avispa is useful in the analysis of process models with variability, variabilities could be shown in Avispa as errors suggested in the same patterns, so, they must be refined to exclude these cases. Processes evaluated using Avispa are processes that lack variability, so this has not interfered with the evaluation results of the process models. Finally, there is a technological gap in the proposals of this thesis, Casper is a proposal including process model properties (context and tailoring rules) not supported by existing tools, thus in order to validate this thesis we have developed a minimal infrastructure. Therefore, there are no actual process models previously specified under this approach. A different scenario occurs with Avispa where available real processes are analyzed using the XML scheme of EPF tool processes. Thus, although both, Avispa and Casper need to mature, Avispa is better prepared for the replication of the case studies and therefore closer to the extension and generalization of the error patterns presented in this thesis. Avispa has been highly welcomed in all companies where it was presented, and process engineers also pointed out that it is relevant for them to count on Avispa for maintaining their software process model, an application not envisioned for the tool at the beginning.

8.4 Limitations

The case studies and the discussions have also evidenced some limitations of the proposed approaches:

1. The context model has been considered as a key asset in software process tailoring. However the technique to define a context model still requires more experimental support. The question is how an organization could define the relevant context attributes and the suitable scale. Thus, a suitable definition of the context model requires evidence-based techniques (Kitchenham, 2006).

2. A major empirical experimentation with Casper meta-process application is required and not only on the MDE strategy. Particularly in defining and implementing software process features to enable the planned tailoring.

3. ATL tailoring rules are not usable. A higher abstraction level is required, a specialized meta-model could be defined to specify rules, for example a decision tree as was showed in Chapter 4 is a promising structure to represent tailoring rules in an abstract way.

4. The Avispa tool does not take advantage of the complete specification of SPEM 2.0. Avispa only uses some key elements and their relationships in order to analyze software processes. A complete analysis tool would significantly increase its power of analysis.

5. Avispa tool metrics are not evidence based. Incrementally metrics were defined and applied to software process models as a calibration mechanism. However empirical studies on the relationships among metrics and process model quality is required.

6. The case studies reveal some unclear aspects of Casper. First, with respect to the software process line evolution, it is to say that many assets could become obsolete including the tailoring rules. Casper metaprocess describes that process are iterative, thus, assets are dynamic elements. During application engineering a jump from application engineering to domain engineering has been established. However this scenario has not been tested. Second, with respect to software process adaptation and the software development itself. Normally, the context is established when the project progresses, so the process execution and the tailoring process are interdepent activities. Although Casper follows a static approach, due to its simplicity and quickness, it could be applied many times until the project context is well understood. However, Casper should be improved to considerer this kind of scenarios.

7. AVISPA and CASPER have been considered powerful tools to process engineers and project managers, respectively. Process engineers were initially afraid for being in charge of analyzing the software process models, but after a while, they were enthusiastic about AVISPA's capacity to find real problems in process specifications. Project managers found the CASPER approach very relevant to get more suitable software processes, however they considered that many tailoring decisions could be planned whereas other only could be taken in to account when new context attributes appear at tailoring time. Because a fine and manual tailoring is necessary in these cases, CASPER included manual tailoring as an additional activity. Although CASPER and AVISPA conceptually could work together, there is a technological limitation. CASPER maturity level is low, so its languages are not known by the industry whereas AVISPA works with industrial processes defined by a known community tool.

8.5 Further Work

Several further work can be considered in the short and medium terms. They include:

1. More experimental work will be developed within the ADAPTE project. It includes work with 6 software companies intending to have adaptable software process models using the techniques proposed in this thesis. For usability issues, the research prototypical ideas should be implemented on practical tools, as Eclipse Process Framework, where context and rules could be defined and executed. The ADAPTE project will offer more empirical evidence on the application of AVISPA too.

2. A transformation language as ATL could result too complex for a process engineer, so it is important to increase the abstraction level, making it easier for the process engineer to define tailoring rules.

3. Some of the models defined could have a more relevant role. It is the case of the process feature model and the scope model. These models could be integrated in the transformation chain for facilitating the adaptation. Specifically, tailoring rules could be automatically programmed from a match model of context attributes with process features.

4. SPEM 2.0 supports the definition of many variation points. However the representation could result unclear to the process modeler. For example, a variation point and a non variation point could not be appropriately differentiated, so an

error could be produced when a variation point is not resolved at rule programming time. So, extensions could be used as a SPEM 2.0 profile to express these issues. However, similarly to what happens in software product lines, the implementation language provides various mechanisms to express variability, making it simple for many purposes, and instead artifacts of a higher abstraction level are used to define and evolve the software product line. So, SPEM 2.0 could keep simple the process model definition and implementation and meta-models could be defined to express a process feature model and the derived process feature configuration models. So, a process feature configuration could be obtained using the context model independently of SPEM 2.0. Consequently, more portable and revolvable rules than SPEM 2.0-based rules could be defined. This configuration could be used to implement directly, with the respective variation points and their variants, a context- adapted software process model.

5. AVISPA will be complemented with a method based on a set of steps guiding the analysis of software process models for specification problems as was introduced in (Bastarrica *et al.*, 2011). Then, conceptual problems could be analyzed with the process users to improve the software process model and the software process itself.

6. The AVISPA tool and its underlying conceptual infrastructure will be extended for supporting more evidence-based metrics and a more complete set of errors from the perspective of a SPEM 2.0 process model. So, more blueprints and problematic patterns could be identified and implemented in AVISPA improving its analysis capability. For example currently AVISPA is being extended to measure the coupling and cohesion levels of a SPEM 2.0 software process model, and implementing the resolution of variability definitions of a SPEM 2.0 model. With modular views a top-down analysis approach could be realized in a more organized way in order to avoid that in a big process the relationships darken the AVISPA blueprints visual information.

7. CASPER meets AVISPA. Although CASPER and AVISPA are complementary approaches, these are not been validate together. As further work, with an SPEM based version of AVISPA or an EPF based version of CASPER support tool, the two approaches could be validated together.

References

Acuña, S.T. & Ferré, X. (2001). Software Process Modelling. In *World Multiconference on Systemics, Cybernetics and Informatics, ISAS-SCIs 2001, July 22-25, 2001, Orlando, Florida, USA, Proceedings, Volume I: Information Systems Development*, 237–242. 14, 16

Aharoni, A. & Reinhartz-Berger, I. (2008). A Domain Engineering Approach for Situational Method Engineering. In *Proceedings of the International Conference on Software Engineering Advances, ICSEA 2008*, 455–468. 21, 47

Antkiewicz, M. & Czarnecki, K. (2004). FeaturePlugin: Feature Modeling Plug-in for Eclipse. In *Proceedings of the Workshop on Eclipse Technology eXchange*, ETX '04, 67–72, ACM, New York, NY, USA. 51

Armbrust, O. & Rombach, H.D. (2011). The Right Process for Each Context: Objective Evidence Needed. In *International Conference on Software and Systems Process, ICSSP 2011, Honolulu, HI, USA, May 21-22, 2011, Proceedings*, 237–241. 49

Armbrust, O., Katahira, M., Miyamoto, Y., Münch, J., Nakao, H. & Ocampo, A. (2008). Scoping Software Process Models: Initial Concepts and Experience from Defining Space Standards. In *ICSP'08: Proceedings of the Software process, 2008 international conference on Making globally distributed software development a success story*, 160–172, Springer-Verlag, Berlin, Heidelberg. 34, 39, 40, 44, 55

Armbrust, O., Katahira, M., Miyamoto, Y., Münch, J., Nakao, H. & Ocampo, A. (2009). Scoping Software Process Lines. *Software Process: Improvement and Practice*, **14**, 181–197. 4, 27, 33, 35, 122

Atkinson, D. & Noll, J. (2003a). Automated Validation and Verification of Process Models. In *Proceedings of the 7th International Conference on Software Engineering and Applications, IASTED 2003*, 587–592. 29

Atkinson, D.C. & Noll, J. (2003b). Automated Checking of Software Process Models. Tech. rep., Santa Clara University. 30

Bai, X., Huang, L. & Zhang, H. (2010). On Scoping Stakeholders and Artifacts in Software Process. In *New Modeling Concepts for Today's Software Processes, International Conference on Software Process, ICSP 2010, Paderborn, Germany, July 8-9, 2010. Proceedings*, 39–51. 4, 27, 34, 123

Bastarrica, C., Hurtado, J.A. & Bergel, A. (2011). Toward Lean Development in Formally Specified Software Processes. In *European System and Software Process Improvement and Innovation, EuroSPI 2011*. 11, 146

Bayer, J., Flege, O., Knauber, P., Laqua, R., Muthig, D., Schmid, K., Widen, T. & DeBaud, J.M. (1999). PuLSE: A Methodology to Develop Software Product Lines. In *Proceedings of the symposium on Software Reusability*, SSR 1999, 122–131, ACM, New York, NY, USA. 56

Beck, K. & Andres, C. (2004). *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional, 2nd edn. 1, 21, 90

Belkhatir, N. & Estublier, J. (1996). Supporting Reuse and Configuration for Large Scale Software Process Models. In *Proceedings of the 10th International Software Process Workshop, ISPW 1996*, 35–39. 23, 24

Benbasat, I., Goldstein, D.K. & Mead, M. (1987). The Case Research Strategy in Studies of Information Systems. *MIS Quarterly Journal*, **11**, 369–386. 8

Bendraou, R., Jezéquél, J.M. & Fleurey, F. (2009). Combining Aspect and Model-Driven Engineering Approaches for Software Process Modeling and Execution. In *International Conference on Software Process, ICSP 2009*, 148–160, Springer-Verlag, Berlin, Heidelberg. 82

Bézivin, J. & Bretón, E. (2004). Applying the Basic Principles of Model Engineering to the Field of Process Engineering. *SPT Software Process Technology - Upgrade*, **V**, 27–33. 26, 35, 41

Boehm, B., Clark, B., Horowitz, E., Westland, C., Madachy, R. & Selby, R. (1995). Cost Models for Future Software Life Cycle Processes: COCOMO 2.0. **1**. 34, 39, 47, 50

Boehm, B.W. (2010). A Risk-driven Decision Table for Software Process Selection. In *Proceedings of the international conference on software processes*, ICSP 2010, 1–1, Springer-Verlag, Berlin, Heidelberg. 56

Bretón, E. & Bézivin, J. (2001). M odel Driven Process Engineering. In *Computer Software and Applications Conference, COMPSAC 2001. 25th Annual International*, 225–230, IEEE Computer Society. 2, 26, 27, 35, 41

Brinkkemper, S., Saeki, M. & Harmsen, F. (1998). Assembly Techniques for Method Engineering. In *Proceedings of the 10th International Conference on Advanced Information Systems Engineering, CAISE 1998*, 381–400, Springer-Verlag, London, UK. 14

Bucher, T., Klesse, M. & Winter, R. (2006). Contextual Method Engineering. Working paper, University of Saint Gallen. 47, 50

Budinsky, F., Brodsky, S.A. & Merks, E. (2003). *Eclipse Modeling Framework*. Pearson Education. 69

Bustard, D.W. & Keenan, F. (2005). Strategies for Systems Analysis: Groundwork for Process Tailoring. In *Proceedings of the 12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, ECBS'05*, 357–362, IEEE Computer Society, Washington, DC, USA. 22

Cánfora, G., García, F., Piattini, M., Ruiz, F. & Visaggio, C.A. (2005). A Family of Experiments to Validate Metrics for Software Process Models. *Journal of Systems and Software*, **77**, 113–129. 5, 27, 28

Clements, P. & Northrop, L. (2001). *Software Product Lines: Practices and Patterns*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. 23, 31, 55, 59

Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Nord, R. & Stafford, J. (2002). *Documenting Software Architectures*. Addison-Wesley. 79

Cockburn, A. (2000). Selecting a Project's Methodology. *IEEE Software*, **17**, 64–71. 21, 118

Conradi, R., Fernström, C., Fuggetta, A. & Snowdon, R.A. (1992). Towards a Reference Framework for Process Concepts. In *Proceedings of the Second European*

Workshop on Software Process Technology, EWSPT 1992:, 3–17, Springer-Verlag, London, UK. 15

Conradi, R., Fernström, C. & Fuggetta, A. (1993). A Conceptual Framework for Evolving Software Processes. *Software Engineering Notes, SIGSOFT*, **18**, 26–35. 13

Cook, J.E. & Wolf, A.L. (1999). Software Process Validation: Quantitatively Measuring the Correspondence of a Process to a Model. *ACM Transactions on Software Engineering Methodology*, **8**, 147–176. 27, 28

Curtis, B., Kellner, M.I. & Over, J. (1992). Process Modeling. *Communications of ACM*, **35**, 75–90. 16

Cusumano, M.A., MacCormack, A., Kemerer, C.F. & Crandall, W.B. (2009). Critical Decisions in Software Development: Updating the State of the Practice. *IEEE Software*, **26**, 84–87. 20

Czarnecki, K. & Antkiewicz, M. (2005). Mapping features to models: a Template Approach Based on Superimposed Variants. In *Proceedings of the 4th international conference on Generative Programming and Component Engineering*, GPCE 2005, 422–437, Springer-Verlag, Berlin, Heidelberg. 52, 110

Czarnecki, K. & Helsen, S. (2006). Feature-based Survey of Model Transformation Approaches. *IBM Systems Journal*, **45**, 621–645. 72

Dai, F. & Li, T. (2007). Tailoring Software Evolution Process. In *International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, ACIS 2007*, vol. 2, 782–787. 21

Dami, S., Estublier, J. & Amiour, M. (1998). APEL: A Graphical Yet Executable Formalism for Process Modeling. *Automated Software Engineering*, **5**, 61–96. 17

Demeyer, S., Ducasse, S. & Nierstrasz, O. (2002). *Object-Oriented Reengineering Patterns*. Morgan Kaufmann. 78, 88

Dörr, J., Adam, S., Eisenbarth, M. & Ehresmann, M. (2008). Implementing Requirements Engineering Processes: Using Cooperative Self-Assessment and Improvement. *IEEE Software*, **25**, 71–77. 20

Eisenhardt, K.M. (1989). Building Theories from Case Study Research. *The Academy of Management Review*, **14**. 8

Feiler, P.H. & Humphrey, W.S. (1993). Software Process Development and Enactment: Concepts and Definitions. In *International Conference of Software Process, ICSP 1993*, 28–40, IEEE Computer Society, Berlin, Germany. 13, 15

Firesmith, D. (2004). Creating a Project-Specific Requirements Engineering Process. *Journal of Object Technology*, **3**, 31–44. 20

Flyvbjerg, B. (2006). Five Misunderstandings About Case-Study Research. *Qualitative Inquiry*, **12**, 219. 8

France, R. & Rumpe, B. (2007). Model-driven Development of Complex Software: A Research Roadmap. *Future of Software Engineering, FOSE 2007*, 37–54. 26

Franch, X. & Ribo, J.M. (1999). Using UML for Modelling the Static Part of a Software Process. In *Proceedings of UML 1999, Forth Collins CO (USA). Lecture Notes in Computer Science*, 292–307, Springer-Verlag. 17

Fuggetta, A. (2000). Software Process: A Roadmap. In *International Conference in Software Engineering - Future of Software Engineering Track*, 25–34. 13, 17

Ge, J., Hu, H., Gu, Q. & Lu, J. (2006). Modeling Multi-View Software Process with Object Petri Nets. In *Proceedings of the International Conference on Software Engineering Advances, ICSEA 2006*, 41, IEEE Computer Society, Washington, DC, USA. 27, 29

Gîrba, T. & Lanza, M. (2004). Visualizing and Characterizing the Evolution of Class Hierarchies. In *Inproceedings of International Workshop on Object-Oriented Reengineering, WOOR 2004*. 83

Gruhn, V. (1991). Validation and verification of software process models. In *Proc. of the Software development environments and CASE technology*, 271–286. 27, 28

Hanssen, G.K., Westerheim, H. & Bjørnson, F.O. (2005). Tailoring RUP to a Defined Project Type: A Case Study. In F. Bomarius & S. Komi-Sirviö, eds., *Proceedings of International Conference on Product Focused Software Process Improvement, PROFES 2005*, vol. 3547 of *Lecture Notes in Computer Science*, 314–327, Springer. 25

Henninger, S. & Baumgarten, K. (2001). A Case-based Approach to Tailoring Software Processes. In *Proceedings of the 4th International Conference on Case-Based Reasoning, ICCBR 2001*, 249–262, Springer-Verlag, London, UK. 22

Humphrey, W.S. (1989). *Managing the Software Process*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. 13, 14, 15, 27

Hurtado, J.A. & Bastarrica, C. (2009). Process Model Tailoring as a Mean for Process Knowledge Reuse. In *2nd Workshop on Knowledge Reuse, KREUSE 2009*, Falls Church, Virginia, USA. 10, 27, 123

Hurtado, J.A. & Bastarrica, M.C. (2010). Tutelkan Implementation Process: Adapting a Reusable Reference Software Process in the Chilean Software Industry. Tech. rep., Computer Science Department, Universidad de Chile. 47, 106, 108, 122

Hurtado, J.A. & Bastarrica, M.C. (2012). Building Software Process Lines with CASPER. In *Proceedings of International Conference on Software and Systems Process, ICSSP 2012*, 170–179, IEEE. 11

Hurtado, J.A., Bastarrica, M.C. & Bergel, A. (2010a). Analyzing the Scrum Process Model with AVISPA. In *International Conference of the Chilean Computer Science Society, SCCC 2010*, 60 –65. 11, 92, 126

Hurtado, J.A., Lagos, A., Bergel, A. & Bastarrica, M.C. (2010b). Software Process Model Blueprints. In *Proceedings of International Conference on Software Process, ICSP 2010*, 285–296. 11, 78, 81, 105, 126, 132

Hurtado, J.A., Bastarrica, M.C. & Bergel, A. (2011a). Analyzing Software Process Models with AVISPA. In *Proceedings of International Conference on Software and Systems Process, ICSSP 2011*, 23–32, ACM. 11, 44, 89, 128

Hurtado, J.A., Bastarrica, M.C. & Bergel, A. (2011b). Is It Safe to Adopt the Scrum Process Model? *CLEI Electronic Journal*, **14**. 11

Hurtado, J.A., Bastarrica, M.C., Quispe, A. & Ochoa, S.F. (2011c). An MDE Approach to Software Process Tailoring. In *Proceedings of International Conference on Software and Systems Process, ICSSP 2011*, 43–52, ACM. 11, 48, 123, 124

ISO (2011). Software Engineering - Lifecycle Profiles for Very Small Entities (VSEs) - Part 2: Framework and Taxonomy. 21

ISO/IEC (1998). ISO/IEC 15504 : Information Technology - Software Process Assessment and Improvement. Tech. rep., International Organization for Standardization. 27, 28

ISO/IEC (2008). ISO/IEC 12207:2008 Systems and Software Engineering – Software life cycle processes. Tech. rep., International Organization for Standarization ISO. 1

Jaccheri, M.L., Picco, G.P. & Lago, P. (1998). Eliciting Software Process Models with the E3 Language. *ACM Transactions on Software Engineering Methodology*, **7**, 368–410. 17

Jacobs, D. & Marlin, C. (1996). Multiple View software Process Support Using the MultiView Architecture. In *Proceedings of the Second International Software Architecture Workshop and International Workshop on Multiple Perspectives in Software Development, ISAW 1996, Viewpoints 1996*, 217–221, ACM, New York, NY, USA. 5, 13, 14, 83

Jacobs, D. & Marlin, C.D. (1995). Software Process Representation to Support Multiple Views. *International Journal of Software Engineering and Knowledge Engineering*, **5**, 585–597. 79

Jacobson, I., Booch, G. & Rumbaugh, J. (1999). *The Unified Software Development Process*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. 21, 41, 118

Johnson, E.W. & Brockman, J.B. (1998). Measurement and Analysis of Sequential Design Processes. *ACM Transactions Design Automatization Electronic Systems*, **3**, 1–20. 29

Jouault, F., Allilaire, F., Bézivin, J. & Kurtev, I. (2008). ATL: A Model Transformation Tool. *Sciences of Computer Programming*, **72**, 31–39. 70

Kajko-Mattsson, M. (2010). Maturity is Also About the Capability to Conform the Process to the Right Context! In *Proceedings of the workshop on Future of software engineering research, FSE/SDP 2010*, FoSER '10, 181–186, ACM, New York, NY, USA. 27, 123

Kang, K.C., Kim, S., Lee, J., Kim, K., Shin, E. & Huh, M. (1998). FORM: A Feature-oriented Reuse Method with Domain-specific Reference Architectures. *Annals of Software Engineering*, **5**, 143–168. 40, 51, 123

Killisperger, P., Stumptner, M., Peters, G., Grossmann, G. & Stückl, T. (2009). Meta Model Based Architecture for Software Process Instantiation. In *International Conference on Software Process, ICSP 2009*, 63–74. 23, 26, 35, 41

Kitchenham, B. (2006). Evidence-Based Software Engineering and Systematic Literature Reviews. In J. Münch & M. Vierimaa, eds., *Product-Focused Software Process Improvement, 7th International Conference, PROFES 2006, Amsterdam, The Netherlands*, vol. 4034 of *Lecture Notes in Computer Science*, 3, Springer. 144

Klein, H.K. & Myers, M.D. (1999). A Set of Principles for Conducting and Evaluating Interpretive Field Studies in Information Systems. *MIS Quarterly*, **23**, 67–93. 8

Knab, P., Pinzger, M. & Gall, H.C. (2010). Visual Patterns in Issue Tracking Data. In *Proceedings of the International Conference on Software Processes*, ICSP 2010, 222–233, Springer-Verlag, Berlin, Heidelberg. 88

Kontio, J. (1998). A Software Process Engineering Framework. *Elsevier*, **46**, 35 – 108. 2

Koolmanojwong, S. & Boehm, B. (2010a). The Incremental Commitment Model Process Patterns for Rapid-fielding Projects. In *Proceedings of the International Conference on Software Processes*, ICSP 2010, 150–162, Springer-Verlag, Berlin, Heidelberg. 56

Koolmanojwong, S. & Boehm, B.W. (2010b). The Incremental Commitment Model Process Patterns for Rapid-Fielding Projects. In *Proceedings International Conference on Software Process, ICSP 2010*, 150–162. 39, 56

Kruchten, P. (2003). *The Rational Unified Process: An Introduction*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. 1, 22, 41

Lange, C.F.J., Wijns, M.A.M. & Chaudron, M.R.V. (2007). Supporting Task-oriented Modeling Using Interactive UML Views. vol. 18, 399–419, Academic Press, Inc. 31

Lanza, M. & Ducasse, S. (2003). Polymetric Views-A Lightweight Visual Approach to Reverse Engineering. *Transactions on Software Engineering*, **29**, 782–795. 30, 83, 88

Larkin, J.H. & Simon, H.A. (1987). Why a Diagram is (Sometimes) Worth Ten Thousand Words. *Cognitive Science*, **11**, 65–100. 29, 44, 79

Lee, K. & Kang, K.C. (2010). Usage Context as Key Driver for Feature Selection. In *Proceedings of the International Conference on Software Product Lines: Going Beyond*, SPLC 2010, 32–46, Springer-Verlag, Berlin, Heidelberg. 56

Lee, M.D., Reilly, R.E. & Butavicius, M.E. (2003). An Empirical Evaluation of Chernoff Faces, Star Glyphs, and Spatial Visualizations for Binary Data. In *Proceedings of the Asia-Pacific Symposium on Information Visualisation, APVis 2003*, 1–10, Australian Computer Society, Inc., Darlinghurst, Australia, Australia. 29, 44, 79

Lonchamp, J. (1993). A Structured Conceptual and Terminological Framework for Software Process Engineering. *Second International Conference on the Continuous Software Process Improvement*, 41–53. 13, 14, 15, 27, 142

Madhavji, N.H. (1991). The Prism Model of Changes. In *Proceedings of the 13th International Conference on Software Engineering, ICSE 1991*, 166–177, IEEE Computer Society Press, Los Alamitos, CA, USA. 15

Martinho, R., Varajao, J. & Domingos, D. (2008). A Two-Step Approach for Modelling Flexibility in Software Processes. In *Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering*, ASE 2008, 427–430, IEEE Computer Society, Washington, DC, USA. 22

Meyer, M., Gîrba, T. & Lungu, M. (2006). Mondrian: An Agile Information Visualization Framework. In *Proceedings of the ACM Symposium on Software Visualization, SOFTVIS 2006*, 135–144, ACM. 96

Mirbel, I. & Ralyé, J. (2005). Situational Method Engineering: Combining Assembly-based and Roadmap-driven Approaches. vol. 11, 58–78, Springer-Verlag New York, Inc. 1, 4, 21, 47

Mishali, O. & Katz, S. (2006). Using Aspects to Support the Software Process: XP Over Eclipse. In *Proceedings of International Conference on Aspect-Oriented Software Development, AOSD 2006*, 169–179, ACM, New York, NY, USA. 57

Neumuller, C. & Grunbacher, P. (2006). Automating Software Traceability in Very Small Companies: A Case Study and Lessons Learned. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering, ASE 2006*, 145–156, IEEE Computer Society, Washington, DC, USA. 47

Nuseibeh, B. & Easterbrook, S. (2000). Requirements Engineering: A Roadmap. In *Proceedings of the Conference on Software Engineering*, ICSE 2000, 35–46, ACM, New York, NY, USA. 108

Ocampo, A., Bella, F. & Münch, J. (2005). Software Process Commonality Analysis. In *Software Process: Improvement and Practice. Special issue on ProSim 2004, The 5th International Workshop on Software Process Simulation and Modeling, Edinburgh, Scotland*, vol. 10, 273–285. 4, 26, 35, 40, 53, 107

O'Connor, R.V. & Laporte, C.Y. (????). Towards the Provision of Assistance for Very Small Entities in Deploying Software Lifecycle Standards. 22

OMG (2008). Software Process Engineering Metamodel SPEM 2.0 OMG Specification. Tech. Rep. ptc/07-11-01, Object Management Group. vii, 5, 14, 17, 18, 20, 60, 61, 64, 78, 89

Osterweil, L.J. (1987). Software Processes are Software Too. In *Proceedings of the 9th International Conference on Software Engineering, ICSE 1887*, 2–13, IEEE Computer Society Press, Los Alamitos, CA, USA. 5, 15, 23, 26, 27, 29, 35, 79

Osterweil, L.J. (1998). JIL and little-JIL Process Programming Languages. In *Proceedings of the 6th European Workshop on Software Process Technology, EWSPT 1998*, 152, Springer-Verlag, London, UK. 17, 30

Osterweil, L.J. & Wise, A.E. (2010). Using Process Definitions to Support Reasoning about Satisfaction of Process Requirements. In J. Münch, Y. Yang & W. Schäfer, eds., *ICSP*, vol. 6195 of *LNCS*, 2–13, Springer. 30, 31

Pandey, D., Suman, U. & Ramani, A. (2010). An Effective Requirement Engineering Process Model for Software Development and Requirements Management. In *Proceedings of International Conference on Advances in Recent Technologies in Communication and Computing ARTCom, 2010*, 287 –291. 107, 122

Park, S. & Bae, D.H. (2011). An Approach to Analyzing the Software Process Change Impact Using Process Slicing and Simulation. *Journal on Systems and Software*, **84**, 528–543. 28

Park, S., Na, H. & Sugumaran, V. (2006). A Semi-automated Filtering Technique for Software Process Tailoring using Neural Network. *Expert Systems with Applications*, **30**, 179–189. 22

Pedreira, O., Piattini, M., Luaces, M.R. & Brisaboa, N.R. (2007). A Systematic Review of Software Process Tailoring. *Software Engineering Notes SIGSOFT*, **32**, 1–6. 3, 117

Pérez, G., El Emam, K. & Madhavji, N. (1996). Evaluating the Congruence of a Software Process Model in a Given Environment. 49–62. 28, 34, 39, 40, 47, 50

Perin, F., Gîrba, T. & Nierstrasz, O. (2010). Recovery and Analysis of Transaction Scope from Scattered Information in Java Enterprise Applications. In *Proceedings of International Conference on Software Maintenance, ICSM 2010*. 30, 88

Pino, F.J., Hurtado, J.A., Vidal, J.C., García, F. & Piattini, M. (2009). ProceedingsA Process for Driving Process Improvement in VSEs, International conference on software process, icsp 2009. In *ICSP*, vol. 5543 of *Lecture Notes in Computer Science*, 342–353, Springer. 10, 92

Raffo, D.M. & Kellner, M.I. (2000). Empirical Analysis in Software Process Simulation Modeling. vol. 53, 31–41, Elsevier Science Inc., New York, NY, USA. 28

Ralyté, J. & Rolland, C. (2001). An Assembly Process Model for Method Engineering. In *Proceedings of the 13th International Conference on Advanced Information Systems Engineering, CAISE 2001*, 267–283, Springer-Verlag, London, UK. 14

Ralyté, J., Deneckére, R. & Roll, C. (2003). Towards a Generic Model for Situational Method Engineering. In *Advanced Information Systems Engineering International Conference 2003, LNCS 2681*, 95–110, Springer-Verlag. 14, 21

Rolland, C. (2009). Method Engineering: State-of-the-Art Survey and Research Proposal. In *Proceeding of Conference on New Trends in Software Methodologies, Tools and Techniques, 2009*, 3–21, IOS Press, Amsterdam, The Netherlands, The Netherlands. 4, 21, 35, 107

Rombach, H.D. (2005). Integrated Software Process and Product Lines. In *International Software Process Workshop, SPW 2005, Beijing, China*, 83–90. 1, 25, 33, 40

Rosch, E. (1978). *Principles of Categorization*, 27–48. Lawrence Erlbaum Associates, Hillsdale (NJ), USA. 8

Rowley, J. (2002). Using Case Studies in Research. *Management Research News*, **25**, 16–27. 8

Royce, W. (1998). *Software Project Management: A Unified Framework*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. 47, 50

Runeson, P. & Höst, M. (2009). Guidelines for Conducting and Reporting Case Study Research in Software engineering. *Empirical Software Engineering*, **14**, 131–164. 8, 106, 121, 125, 128, 137

Sadiq, W. & Orlowska, M.E. (2000). Analyzing Process Models Using Graph Reduction Techniques. *Information Systems*, **25**, 117 – 134. 31

Sauer, C., Jeffery, D., Land, L. & Yetton, P. (2000). The Effectiveness of Software Development Technical Reviews: a Behaviorally Motivated Program of Research. *IEEE Transactions on Software Engineering*, **26**, 1–14. 29

Scacchi, W. (2000). Understanding Software Process Redesign Using Modeling, Analysis and Simulation. 29

Schwaber, K. (1995). SCRUM Development Process. In *Proceedings of the Annual ACM Conference on Object Oriented Programming Systems, Languages, and Applications, OOPSLA 1995*, 117–134. 1, 41, 90, 92

Schwaber, K. & Sutherland, J. (2010). Scrum. Available in: http://www.scrum.org/storage/scrumguides/Scrumof 2012. 90

SEI (2006). CMMI® for Development, Version 1.2. Tech. Rep. CMU/SEI-2006-TR-008, Software Engineering Institute. 1, 27, 28, 41

Shull, F., Basili, V., Carver, J., Maldonado, J.C., Travassos, G.H., Mendonça, M. & Fabbri, S. (2002). Replicating Software Engineering Experiments: Addressing the Tacit Knowledge Problem. In *Proceedings of International Symposium on Empirical Software Engineering, ISES 2002*, 7–, IEEE Computer Society, Washington, DC, USA. 8

Simidchieva, B.I., Clarke, L.A. & Osterweil, L.J. (2007). Representing Process Variation with a Process Family. In *International Conference on Software Process, ICSP 2007*, Lecture Notes in Computer Science, 109–120, Springer. 4, 24, 33

Soto, M., Ocampo, A. & Münch, J. (2009). Analyzing a Software Process Model Repository for Understanding Model Evolution. In *Proceedings of the International Conference on Software Process, ICSP 2009*, ICSP '09, 377–388, Springer-Verlag, Berlin, Heidelberg. 30

Sutherland, J., Downey, S. & Granvik, B. (2009). Shock Therapy: A Bootstrap for Hyper-Productive Scrum. In Y. Dubinsky, T. Dybå, S. Adolph & A.S. Sidky, eds., *AGILE*, 69–73, IEEE Computer Society. 90

Sutton, S.M. & Osterweil, L.J. (1996a). PDP: Programming a Programmable Design Process. In *Proceedings of the 8th International Workshop on Software Specification and Design*, 186–190. 23, 25

Sutton, S.M. & Osterweil, L.J. (1996b). Product Families and Process Families. In *Proceedings of the 10th International Software Process Workshop, ISPW 1996*, 109, IEEE Computer Society, Washington, DC, USA. 23, 33

Valdés, G., Astudillo, H., Visconti, M. & López, C. (2010). The Tutelkán SPI Framework for Small Settings: A Methodology Transfer Vehicle. In *Proceedings of the 17th European System & Software Process Improvement and Innovation, EuroSPI 2010*, Grenoble, France. 92, 108

Van Rijsbergen, C.J. (1979). *Information Retrieval*. Butterworths, London, 2nd edn. 135

Vergara, A. (2008). *Automatic Generation of Requirements-based Metrics in Software Projects*. Master's thesis, Computer Science Department. Universidad de Chile. 66

Villarroel, R., Fajardo, R. & Rodríguez, O. (2010). Implementation of an Improvement Cycle using the Competisoft Methodological Framework and the Tutelkán Platform. *CLEI Electronic Journal*, **13**. 92

Von Wangenheim, C.G., Anacleto, A. & Salviano, C.F. (2006). Helping Small Companies Assess Software Processes. *IEEE Software*, **23**, 91–98. 47

Washizaki, H. (2006). Building Software Process Line Architectures from Bottom up. In J. Münch & M. Vierimaa, eds., *Product-Focused Software Process Improvement*, PROFES 2006, 415–421, Springer. 3, 4, 25, 26, 33, 41, 62, 75

Xu, P. (2005). Knowledge Support in Software Process Tailoring. *Proceedings of the Annual Hawaii International Conference on System Sciences, HICSS 2005*. 22

Yin, R.K. (1984). *Case Study Research: Design and Methods*. Applied social research methods series, Sage Publications, Beverly Hills, CA. 7, 106

Yoon, I.C., Min, S.Y. & Bae, D.H. (2001). Tailoring and Verifying Software Process. In *Asia-Pacific Software Engineering Conference, APSEC 2001*, 202–209. 21

Zamli, K.Z. (2004). A Survey and Analysis of Process Modeling Languages. *Malaysian Journal of Computer Science*, **17**, 68–89. 2, 17

Zave, P. (1997). Classification of Research Efforts in Requirements Engineering. *ACM Computer Survey*, **29**, 315–321. 49

Zhu, L., Jeffery, D.R., Staples, M., Huo, M. & Tran, T.T. (2007). Effects of Architecture and Technical Development Process on Micro-process. In *International Conference on Software Process, ICSP 2007*, LNCS, 49–60, Springer. 26