The Journal of Systems and Software 86 (2013) 1153–1171

Contents lists available at SciVerse ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

MDE software process lines in small companies

Julio Ariel Hurtado[a,b,*], María Cecilia Bastarrica[a], Sergio F. Ochoa[a], Jocelyn Simmonds[c]

[a] Computer Science Department, Universidad de Chile, Chile
[b] IDIS Group, Systems Department, Universidad del Cauca, Colombia
[c] Informatics Department, Universidad Técnica Federico Santa María, Chile

ARTICLE INFO

Article history:
Received 28 November 2011
Received in revised form
20 September 2012
Accepted 25 September 2012
Available online 20 November 2012

Keywords:
Software process lines
Model-driven engineering
Process asset reuse

ABSTRACT

Software organizations specify their software processes so that process knowledge can be systematically reused across projects. However, different projects may require different processes. Defining a separate process for each potential project context is expensive and error-prone, since these processes must simultaneously evolve in a consistent manner. Moreover, an organization cannot envision all possible project contexts in advance because several variables may be involved, and these may also be combined in different ways. This problem is even worse in small companies since they usually cannot afford to define more than one process. Software process lines are a specific type of software product lines, in the software process domain. A benefit of software process lines is that they allow software process customization with respect to a context. In this article we propose a model-driven approach for software process lines specification and configuration. The article also presents two industrial case studies carried out at two small Chilean software development companies. Both companies have benefited from applying our approach to their processes: new projects are now developed using custom processes, process knowledge is systematically reused, and the total time required to customize a process is much shorter than before.

1. Introduction

Software development life cycles, from traditional models like Waterfall, to more modern ones like RUP, Scrum and XP, suggest specific activities that need to be carried out as part of the development process, as well as the order of these activities. Moreover, if a company wants to certify or evaluate its software development process, this process must be rigorously defined as prescribed by the most popular models and standards (e.g., CMMI, 2006; ISO/IEC 12207, 2008). Specifying an organizational process requires an enormous amount of effort, and the resulting process must still be adapted in order to satisfy the specific characteristics of different project settings (Mirbel and Ralyté, 2006).

There is no optimal software process since appropriateness depends on various organizational, project and product characteristics and, even worse, these characteristics evolve continuously. Therefore, 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 (Laplante and Neill, 2004). The process of selecting a set of practices and then integrating them into a coherent process must also be aligned with the business context (Cusumano et al., 2009). Based on the findings presented in Dörr et al. (2008), we support the idea that a project's context must be taken into account when deciding which process variant best fits the project. Additionally, the specific process applied should not vary dramatically from project to project, so that the process knowledge acquired by the development team can be reused.

The process through which a general software process is configured in order to adapt it to a project's particular setting is known as tailoring (Pedreira et al., 2007). In other areas, like Situational Method Engineering, this process is also known as configuration. In this work, we consider configuration and tailoring as synonyms. Empirical studies show that process tailoring is difficult because it involves intensive knowledge generation and deployment (Rolland, 2009), and it is also time-consuming (Ocampo et al., 2005). Moreover, the expertise required to produce a good process tailoring may be lost from one project to the next. Therefore, the tailoring process itself is hard to replicate and does not scale if done manually.

We apply two complementary approaches to process tailoring: *model-driven engineering* and *software process lines*. Model-driven engineering (MDE) is a software development approach that advocates the creation of abstract models, which are then systematically transformed into more concrete models, and eventually into source

* Corresponding author at: IDIS Group, Systems Department, Universidad del Cauca, Colombia.
E-mail addresses: jhurtado@dcc.uchile.cl (J.A. Hurtado), cecilia@dcc.uchile.cl (M.C. Bastarrica), sochoa@dcc.uchile.cl (S.F. Ochoa), jsimmond@inf.utfsm.cl (J. Simmonds).

0164-1212/$ – see front matter © 2012 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jss.2012.09.033

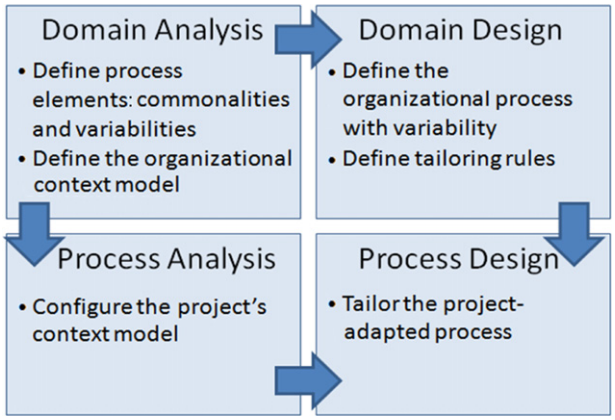

Fig. 1. The product line engineering framework (Pohl et al., 2005), adapted to allow software process line development.

code (Schmidt, 2006). This approach promotes reuse through a generative strategy. MDE can also be used in the software process engineering domain (Breton and Bézivin, 2001), where transformations are used as instantiation strategies (Killisperger et al., 2009).

On the other hand, a software product line (SPL) is a set of software-intensive systems that share a common and managed set of assets satisfying the specific needs of a particular market segment or mission (Clements and Northrop, 2001). Team productivity and product quality increase as a consequence of the resulting massive reuse of assets. Analogously, software process lines (SPrL) is a set of software processes, built from a series of shared process assets in order to reuse process knowledge across projects with different needs. According to Rombach, a SPrL is a systematic mechanism for managing a process and its variants (Rombach, 2005).

The FAST (Family-Oriented Abstraction, Specification, and Translation) process (Weiss and Lai, 1999), which was particularly proposed to define products families, separates product-line engineering process into two main stages: *domain engineering* and *product engineering*. In domain engineering, common and variable elements are identified, as well as how these may be combined in order to generate particular products. Each of these elements is implemented as a reusable asset. During the product engineering stage, members of the product family are built using the previously defined assets.

Each stage of the SPL definition is further subdivided into other phases: analysis, design, implementation and verification (Pohl et al., 2005). In Fig. 1, we show how the analysis and design phases of domain and product engineering were adapted to allow the development of SPrLs. The activities involved in these SPrL phases are described in the rest of the article. We also show their application in two industrial case studies.

Thus, the main contribution of this article is an MDE-based approach to software process line analysis and design, focusing on process tailoring with respect to a context. There are multiple advantages to combining MDE and SPrL: by explicitly specifying implicit tailoring knowledge as tailoring transformations (as proposed by MDE), and by creating a general library of reusable process assets (as proposed by SPrL), we can systematically tailor general, reusable processes with respect to a specific project context.

Small Software Organizations (SSO) has particular characteristics that affect their software processes and make them different from medium-size and large organizations. Only some of these companies have a rigorous organizational structure and a formally defined development process (von Wangenheim et al., 2006a,b). They usually work on small projects, involving small groups (Batista and De Figueiredo, 2000; Harris et al., 2007), and basing their competitiveness on being specialized (Aranda et al., 2007). Therefore, a successful SSO is usually linked to a particular niche (Aranda et al., 2007; Jantunen, 2010) which makes the development context rather stable and thus reduces the required variability of the organizational software process. These companies typically consider a few project context variables (Hofer, 2002), but different configurations of these variables require different development processes (Hurtado et al., 2011). We have validated our approach in two small companies: KI Teknology and Amisoft. These organizations are considered small according to European Commission (2005) because they both have around 50 employees, and their annual turnover does not exceed 10 million Euros. The selected companies develop software for niche markets (Aranda et al., 2007; von Wangenheim et al., 2006b), so the domain and technology do not change dramatically from project to project. KI Teknology develops web-based applications and Amisoft focuses on judiciary information systems. The rest of the paper is structured as follows. Section 2 presents the related work. In Section 3, we describe the tailoring process, as well as the models and transformations required by our approach, illustrating these with a running example. In Sections 4 and 5, we describe two case studies where we applied the proposed tailoring approach. Finally, Section 6 presents the conclusions and future work.

2. Related work

In this section, we first give a brief overview of software product lines and software process lines. We then discuss several approaches to software process tailoring, and finally, we review various proposals for representing and using context information.

2.1. Software product lines and software process lines

Given that software processes are software too (Osterweil, 1987), a software process line (SPrL) can be considered as a special software product line (SPL) in the software process engineering domain. Processes in a SPrL share common features and exhibit variability (Stanley and Osterweil, 1996). Consequently, a SPrL is an ideal way to define, tailor and evolve a set of related processes, an opinion that is supported by the work on process variability representation (Simidchieva et al., 2007), process families (Rolland and Nurcan, 2010), SPrL architectures (Washizaki, 2006), process domain analysis (Ocampo et al., 2005), and SPrL scoping (Armbrust et al., 2009). A classic tailoring approach reactively integrates unanticipated variability in the process model (Armbrust et al., 2009), while a SPrL approach facilitates planned reuse. Nevertheless, as stated in Deelstra et al. (2005), deriving products (processes) in the context of a SPL (SPrL) is a challenging task.

SPEM 2.0 (OMG, 2007) is the OMG standard for modeling software and systems development processes and their components. Various small Chilean software development companies, as part of the Tutelkán project (Valdés et al., 2010), formalized their processes using SPEM 2.0. This standard defines four primitives for specifying variability, which must be specified between two process elements of the same type:

1. Contributes: a source variability element *contributes* its properties to the target variability element without directly altering any of the target element's properties. The target element takes on any extra attributes and associations defined by the source element, except for those already defined by the target element.
2. Replaces: a source variability element *replaces* its target variability element. In this case, only the incoming associations of the target element are preserved, both the target's attributes and outgoing associations are replaced by the source element's. The

target of multiple *replaces* relations can only be replaced by one source element in a configuration.
3. Extends: a source variability element *extends* the definition of its target variability element, possibly overriding the target's attributes and associations.
4. Extends-Replaces: in this relationship, a source variability element first *extends* its target variability element, and then *replaces* it.

Instances of these relations may override each other, so variability relations must be resolved in the presented order. The process tailoring step is successful if the process engineer can resolve away all variability in a non-conflicting manner. A priori, it is hard to predict how variability relations interact with each other, which is why the SPEM 2.0 variability mechanisms are rarely used in practice (Martínez-Ruiz et al., 2011).

2.2. Tailoring approaches

There are several approaches to process tailoring, which mainly differ in terms of expected process formality, size of the company, and available tool support (Pedreira et al., 2007). For example, the *assemble* approach presented in Dai and Li (2007) provides a reduced set of process modification actions (add, delete, split and merge operation), which are used to "assemble" a new process from a previous version. The underlying formalism is Petri nets, and as such, process models only consider process activities, ignoring roles and work products, and there are no variable elements. Also, the project context is not formally used to guide the adaptation process, as tailoring is done in an ad hoc manner by the process engineer, who selects what action to apply and when.

The *situational method engineering* (SME) approach focuses on project-specific process construction (Ralyté et al., 2003), where pre-existing pieces of methods are selected and combined in an attempt to produce the most appropriate process for an organization or a project. This is an active research area, see (Henderson-Sellers and Ralyté, 2010; Tolvanen et al., 1996) for surveys on these techniques. SME techniques can be used not only to customize a software development process to a particular project context, but also to perform a continuous improvement process (Bajec et al., 2007a,b). Nevertheless, in most cases the effort for tailoring the process is still huge, especially when an assemble approach is adopted (Mirbel and Ralyté, 2006). This is a big problem because process tailoring is normally the responsibility of the project manager, but it requires the experience and knowledge of a software process engineer, so it is difficult to achieve a reasonable separation of their tasks (Bai et al., 2012). The two most significant challenges for the SME community are the rate of industry adoption and how to automate the method construction process (Henderson-Sellers and Ralyté, 2010). Provided that SME follows a bottom-up approach, it is a creative task and requires the knowledge and experience of a method engineer; thus, automation in SME is very complex if possible at all. Instead, our approach uses a top-down strategy, where a process structure including variation points and their variants are defined. In our approach, the flexibility is controlled and only some process could be defined according to the variation points. However, this limited variation allows us to implement a practical transformational strategy for automatically generating each process. Hence, our approach follows a trade-off criterion between flexibility and practicality.

Some processes like the Unified Process (Jacobson et al., 1999) use an *adjustment guide* approach, where tailoring rules are specified as general guidelines about how to adapt phases, iterations and disciplines according to project-specific situations. This was the approach originally followed by KI, the company subject of our first case study (see Section 4). The successful application of this approach highly depends on the process engineer, who must be careful to be consistent with how these guidelines are applied to different projects (each one with a potentially different context).

Agile methods such as XP (Beck and Andres, 2004) use an *auto-adaptable* approach, where the project- and team-adapted process emerges from the set of principles, values and practices that define the agile methodology. Other processes such as Crystal Methodology (Cockburn, 2000) follow a *template-based* approach, where a family of methodology templates are defined (Clear, Yellow, Orange and Red), and each template is more detailed than the last. Commercial processes, such as RUP (Kruchten, 2003), use a *framework-based* approach (Belkhatir and Estublier, 1996) (also known as a configuration approach), where a general process is defined and a specific configuration is created for each specific project. Processes developed using this approach tend to be large and complex, and a high level of process engineering knowledge is required to produce valid configurations, whereas the difficulty of the template strategy is that it is hard to define an adequate set of templates that satisfy all the possible project scenarios (Bustard and Keenan, 2005).

Killisperger et al. (2009) propose an *instantiation-based* approach; it consists of defining a general process that is instantiated for each project up to the enactment level. This approach requires an enormous amount of process formalization in order to obtain all of the expected benefits. In our work, we tailor the general process according to the characteristics of the project context, and the resulting SPEM process serves as a process guide. However, we do not reach the enactment level as this is not supported by SPEM. In this article, we represent process model variability using a process feature model, similar to a software product feature models (Kang et al., 1990), and the organizational software process is modeled using eSPEM, a metamodel including the core SPEM concepts and relations required by our proposed tailoring mechanism. The MDE strategy we apply helps achieve a separation between the process modeling stakeholders and process enactment (project) stakeholders (Bai et al., 2012), and it hides the complexity by intensively reusing tailoring knowledge.

2.3. Context modeling

Context representation in software development has gained relevance lately, particularly because of its use in software process tailoring. However, its definition and scope differ in almost every proposal. The literature reports the use of the context to address several challenges, e.g., estimating project costs, selecting the software process to be used in a particular project, or characterizing a SPrL. Most context representations involve specifications in text (Kettunen and Laanti, 2005; Xu, 2005), tables (Killisperger et al., 2009; Koolmanojwong and Boehm, 2012) or some kinds of checklists (Ma and Wang, 2006; Park et al., 2006), which are not always in easy-to-process formats.

The work of Pérez et al. (1995) presents a set of context characteristics related with processes. Such context characteristics support the customization of software process models. Armbrust et al. (2009) define three dimensions (or context variables) to define the context characteristics in the SPrL scope definition: product, project and process. The COCOMO II model (Boehm et al., 1995) defines a set of attributes and dimensions that help estimate project metrics. Such context variables are also useful for representing context models. The Incremental Commitment Model (ICM) process (Koolmanojwong and Boehm, 2012) defines four process patterns for rapid process deployment using contextualized information, processes for new projects are derived from these patterns. Zowghi et al. (2005) propose a strategy to define an appropriate requirements engineering method that is well-suited for the particular system or application development endeavor under consideration.

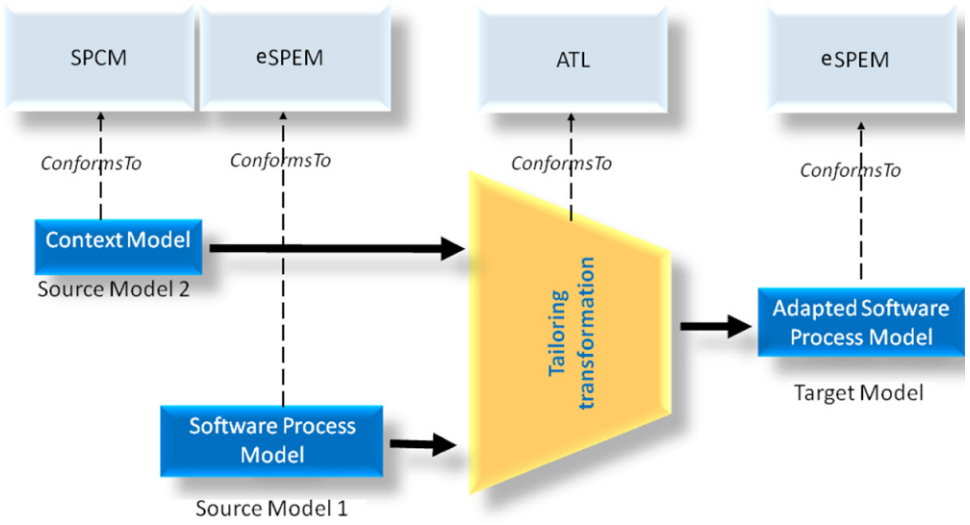

Fig. 2. Generative strategy for process tailoring.

The context information used to perform this activity is embedded in a set of guidelines used as support.

Kornyshova et al. (2010) describe the context dimensions and variables affecting the tailoring process in the information systems domain. They also present a process for contextualizing methods and selecting method components based on the project characteristics. The context model described in Kornyshova et al. (2010) is focused on the contextualization of method components for selecting and reusing method components instead of tailoring the software process to a specific situation. The component selection is carried out using queries on the values of the characteristics of the available method components. If the query does not provide a conclusive result, the method engineer could considerer other methods, modify the required characteristics' values, or rank characteristics by their order of importance. Our approach uses context information to identify the relationship between the process variability and context attributes, and at tailoring time also to define the specific situation (context attribute values) where the process model will be applied. Whereas method components context is oriented to method reuse following a bottom-up approach, project-specific context is oriented to process adaptation following a top-down approach.

Hurtado et al. (2011) present a survey of context representations that can be used for tailoring software development processes, and also a simple language to represent the project contexts. Our proposal uses this language to represent software context models.

In order to help organizations to determine their relevant dimensions and context attributes to perform the tailoring activities, we have defined a Software Process Context Metamodel, based on the ideas presented in Hurtado and Bastarrica (2009). We describe this metamodel in Section 3.2.

3. Tailoring the software process

Defining an organizational software process is necessary if a company wants to improve its development process, and absolutely necessary in order to achieve an evaluation or certification such as CMMI or ISO/IEC 12207. Although defining and documenting a general process requires a significant amount of work, this process is not necessarily appropriate for all projects, even within the same organization. Moreover, an organization that usually develops certain types of projects using a particular process may eventually become involved in a different type of project, and thus the process that worked well before becomes inadequate. Defining a customized process for each project is too expensive due to the amount of resources this effort would consume, resources which could be used by the project instead. Having a set of predefined processes for a series of different contexts implies a high maintenance cost, and it still does not ensure that all possible contexts are taken into account. Therefore, tailoring the organizational process seems to be a good trade-off.

Fig. 2 presents an overview of our proposed tailoring approach, which uses two models as input: the *software process model* and the *context model* for a specific project. Then, using a set of transformation rules (embedded in the *tailoring transformation* component), the software process is tailored according to the project *context model* in order to generate the *adapted software process model*. This tailoring proposal was briefly introduced in Hurtado et al. (2012b), and is illustrated with an example in the rest of this section.

The input *software process model* is an eSPEM model that represents the general organizational process that will be tailored. This model must be customized at the beginning of each new project. eSPEM models consist of task, work product and role definitions, as well as a specification of how these process elements interact in order to accomplish the process's goals. The initial software process model includes the specification of its variability, which is resolved through the tailoring process in order to achieve the *adapted software process model*.

In our tailoring approach, the organization must also define the *context model*, which is used to describe the different types of projects that the company handles. This *context model* specifies relevant project variables (e.g., the project type), their possible values (e.g., development or maintenance), as well as how these variables are classified along different dimensions.

The *tailoring transformation* component, that involves the set of transformation rules expressed in ATL (Jouault et al., 2006), indicates under which conditions the general process model should change, and how. Rule conditions are expressed using the context variables and values, and valid process changes include task, work product and role inclusion/elimination, as well as choosing from a set of possible alternatives. In the case of organizations with mature development processes, these rules can usually be deduced from (semi-)formal general process adaptation guidelines. In other cases, these rules must be formalized from scratch or reverse-engineered from existing projects. This formalization is one of the most difficult steps in the proposed approach, but once the transformation rules are formalized and validated, tailoring becomes simpler, more consistent, and less error-prone.

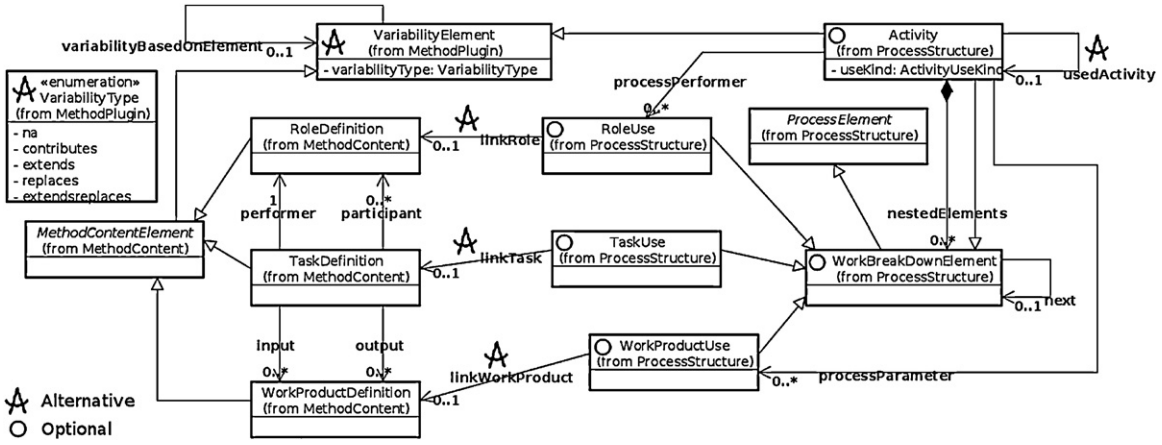

Fig. 3. The experimental SPEM (eSPEM) metamodel, where the process element metaclasses that can vary have been marked.

We have developed a proof-of-concept tool chain to support our empirical work. This tool chain is built on top of the Eclipse Modeling Framework – EMF 3.4[1] and the ATL plug-in 2.0.[2] Metamodels were defined as ecore metamodels in EMF and the transformations were implemented as ATL rules. Models were implemented as instances of defined metamodels and edited using Exeed (Extended EMF Editor), the EMF reflective editor.

In this section, we first define the models and metamodels involved in the proposed tailoring approach, and then the ATL transformations that implement the tailoring process. All steps are illustrated using a simple example.

3.1. Organizational process model

SPEM 2.0 (OMG, 2007) is the OMG standard for modeling software and systems development processes and their components. SPEM 2.0 is defined both as a MOF metamodel and as a UML 2.0 profile, and takes an object-oriented approach to process modeling. SPEM diagrams are used to model two different views of a process: a static view, where process components (tasks, work products and roles) are defined; and a dynamic view, where interaction diagrams (like UML activity diagrams) are used to model how process components interact in order to accomplish the goals of the modeled process.

In order to encourage process maintenance and reuse, SPEM 2.0 makes a difference between the definition of process building blocks and their later use. Process components, like tasks, roles and work products, are defined and stored in a Method Library. An activity is a "big-step" grouping of role, work product and task uses, and activity diagrams are used to model the workflow between tasks of the activity. Roles perform tasks, and work products serve as input/output artifacts for tasks.

In SPEM, a process is a set of activities, where the relationship between these activities is also specified as a workflow. Task, role, work product and activity definitions are recommendations made by a process engineer, so these can be overridden when creating a new process, e.g., by adding/removing the association between a work product and a task.

In our approach, process models are defined using eSPEM (Hurtado et al., 2012b), a SPEM 2.0 subset, which is expressive enough for our experimental purposes. The metamodel for eSPEM is shown in Fig. 3. *Task-*, *Role-* and *WorkProductDefinition* are subclasses of the *MethodContentElement* metaclass, which in turn is a subclass of the *VariabilityElement* metaclass. On the other hand, *Activity*, *Task-*, *Role-* and *WorkProductUse* are subclasses of the *WorkBreakDownElement* metaclass. A *VariabilityElement* is a process element that can be modified or extended by other *VariabilityElements* of the same kind through a variability relationship (defined by the *VariabilityType* enumeration).

We use *VariabilityElements* to implement alternatives (labeled with an alternative symbol similar to that used in feature models). A set of alternatives can be defined from the same *VariabilityElement* (which may be abstract). So, when a *ProcessElement* is linked to the *VariabilityElement*, one of these alternatives could be selected. For example, a *TaskUse* can be linked to one of many available and consistent *TaskDefinitions*. Additionally, each *WorkBreakDownElement* can be considered as optional or not according to the *isOptional* attribute. Optional elements are labeled with an empty circle.

For example, the general requirements engineering process shown in Fig. 4 is a part of the overall development process followed by students taking CC51A, an advanced software development course taught at the University of Chile. This process consists of three activities: *Exploration*, *User Requirements Specification and Validation*, and *Software Requirements Specification and Validation*, carried out in sequence. The main goal of this course is to expose senior students, in groups of 5–6, to a real-life work environment, developing real applications for real clients. The process itself is simple because students only have four months to complete their projects. Fig. 5 shows the eSPEM specification of this process.

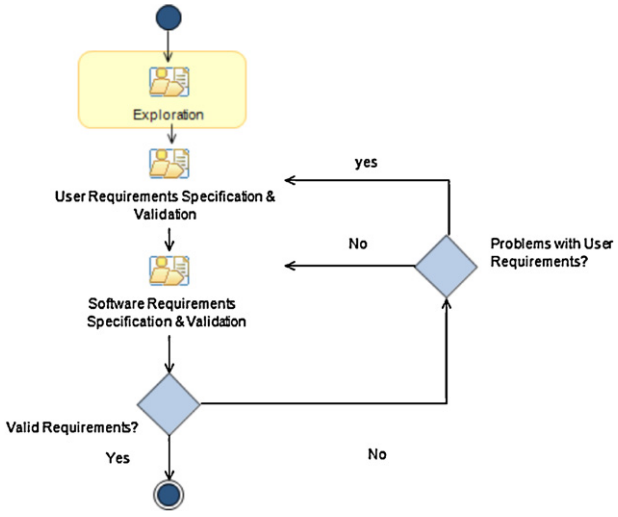

Fig. 4. The CC51A requirements engineering process.

[1] EMF website http://download.eclipse.org/tools/emf.
[2] ATL website http://www.eclipse.org/downloads/.

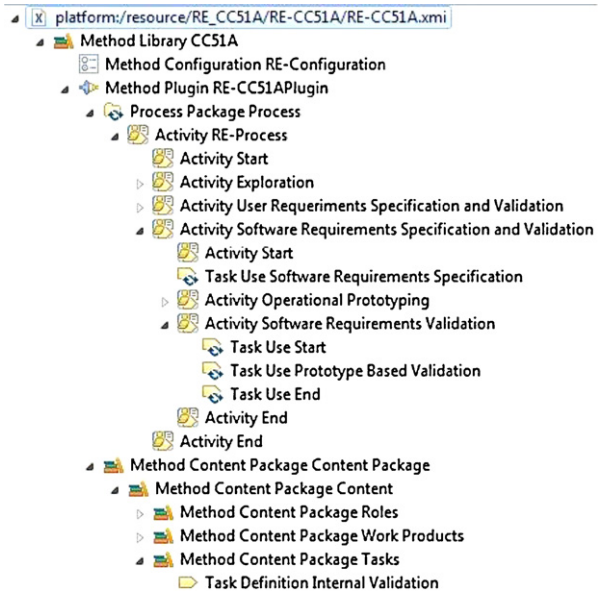

Fig. 5. eSPEM model of the CC51A requirements engineering process.

During the *Exploration* activity, the client, analyst and project manager meet in order to define the project. Occasionally, a client shows up with a clearly defined project and this activity is not necessary, which is why it is marked as optional (enclosed in a yellow box). This notation is ad hoc, since the SPEM standard does not recommend how to visually indicate element optionality. During the *User Requirements Specification and Validation* activity, the team members specify the goals and scope of their project 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. This activity takes as input a project definition, and produces as output a user requirements specification.

During the *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 (which must be validated by the users and clients). If the team members are familiar with the project domain, this validation can be done internally; otherwise, the team must create an operational prototype and validate it with the client.

Following a general approach for specifying variability in Domain Engineering, we use feature models (Kang et al., 1990) to formally specify process variability at a high level of abstraction (Simmonds and Modeling, 2011). Features represent activities, tasks, roles and work products, and we allow cross-tree constraints between features of any kind.

For example, the feature model in Fig. 6 shows both the common and variable elements of the CC51A RE process. This model shows the *Software Requirements Specification and Validation* activity in more detail, breaking it down into three parts: a *Software Requirements Specification* task, an (optional) *Operational Prototyping* activity, and a *Software Requirements Validation* task. The *Software Requirements Validation* is further broken down, representing the two validation alternatives mentioned previously. If the team is familiar with the project domain, they can carry out an *Internal Validation*, otherwise, they must do a *Prototype Based Validation* (but in order to do this, the team *must* have first carried out the *Operational Prototyping* activity).

3.2. Context model

The context of a project may vary according to different project variables along specific dimensions such as: size, duration, complexity, development team size, knowledge about the application domain, or familiarity with the technology involved. Formalizing these characteristics as a model enables us to automatically tailor the organizational process according to them. We have defined SPCM (Software Process Context Metamodel) for specifying the context model for each project. The metamodel for SPCM is shown in Fig. 7.

SPCM is based on three basic concepts: *ContextAttribute*, *Dimension* and *ContextAttributeConfiguration*. Every element in SPCM extends a *ContextElement* that has a name and a description. A

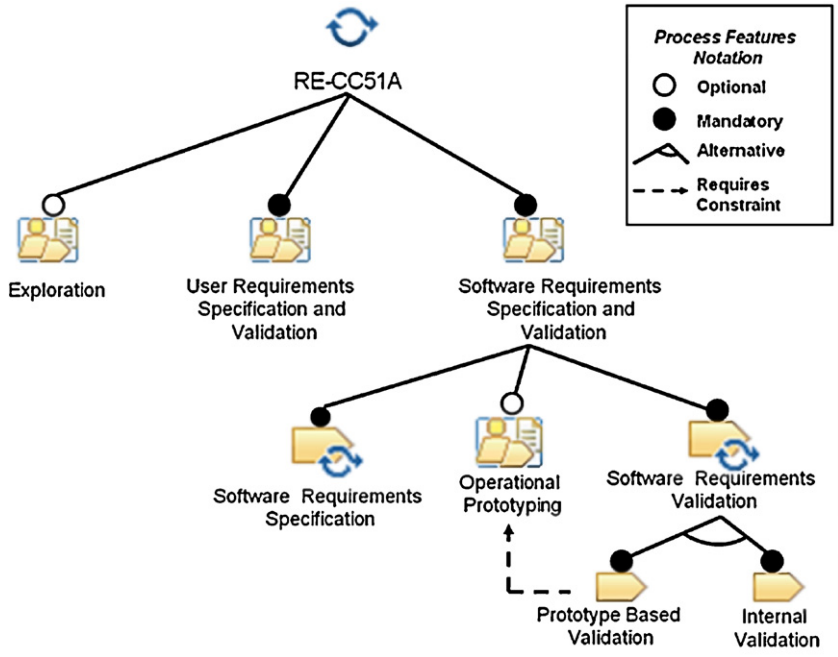

Fig. 6. Feature model corresponding to the CC51A requirements engineering process.

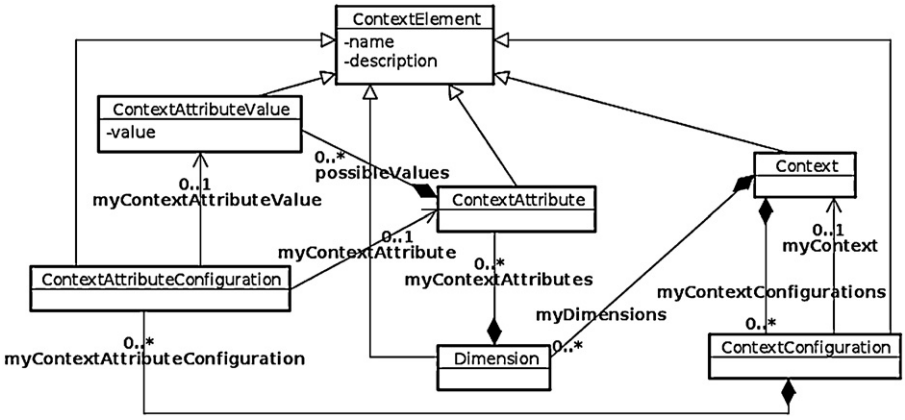

Fig. 7. The Software Process Context Metamodel – SPCM.

ContextAttribute represents a relevant characteristic of the process context required for tailoring. The *ContextAttribute* includes a priority (used when a trade-off among context attributes is required). A *ContextAttribute* can take one of a set of values defined as *ContextAttributeValue*. An example of a *ContextAttribute* is the size of the project (Size). *ContextAttributeValue* represents a type for qualifying a *ContextAttribute*. Examples of *ContextAttributeValues* for the *ContextAttribute* Size are the *ContextAttributeValues* {Small, Medium, Large}. *Dimension* represents a collection of related *ContextAttributes*. By grouping *ContextAttributes* by *Dimension*, the context model is easier to understand. An example of a *Dimension* is Team, which can be used to group team attributes like Team Size and Team Capabilities. A *Context* model is a collection of *Dimensions*.

For example, the context model of the CC51A RE process is depicted in Fig. 8. This context model has three dimensions: project, product and team. This model also has six context attributes: domain knowledge, complexity, project type, duration, size and training. The first three attribute types are associated with predetermined context attribute values (through enumeration) for each project, while the last three attributes are constants provided that this process is followed by students as part of a course.

A *ContextConfiguration* is a set of *ContextAttributeConfigurations*, where each *ContextAttributeValue* is a valid *ContextAttribute*. Therefore, each *ContextAttributeConfiguration* is associated with a *ContextAttribute* and to one unique *ContextAttributeValue*. For example, a possible *ContextAttributeConfiguration* is the ProjectTypeConfiguration for a development project, where one of its *ContextAttributes* is Project Type and its associated *AttributeValue* is "Development".

Table 1 shows how the different context attributes affect the variable elements of the CC51A RE process. If the team's domain knowledge is Medium to High, then the team can avoid doing a prototype-based validation of the software requirements, skipping the *Operational Prototyping* activity. The team's level of domain knowledge does not affect the optional *Exploration* activity, since the realization of this activity depends solely on the project type.

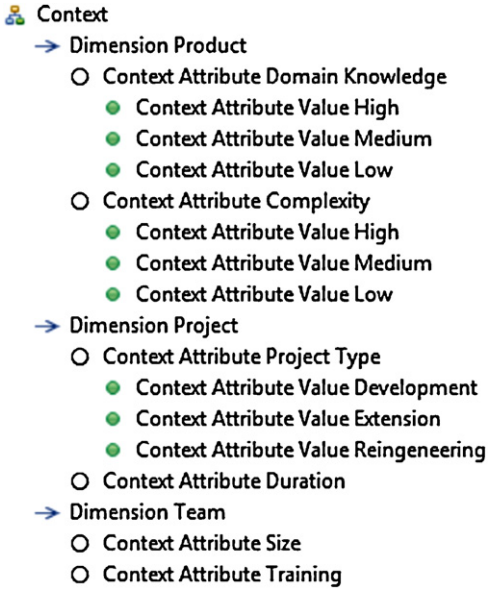

Fig. 8. Context model for the CC51A requirements engineering process.

However, the type of project does affect whether or not the *Exploration* activity is carried out. Complexity does not affect the process.

3.3. Tailoring by model transformation

We use ATL (Jouault et al., 2006) for defining the tailoring transformation rules. Rules about tailoring the general process model according to the values of different 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. Matched rules constitute the core of an ATL declarative

Table 1
Process scoping according to CC51A context model.

Context attribute	Attribute value	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	–	–
	Reengineering	True	–	–

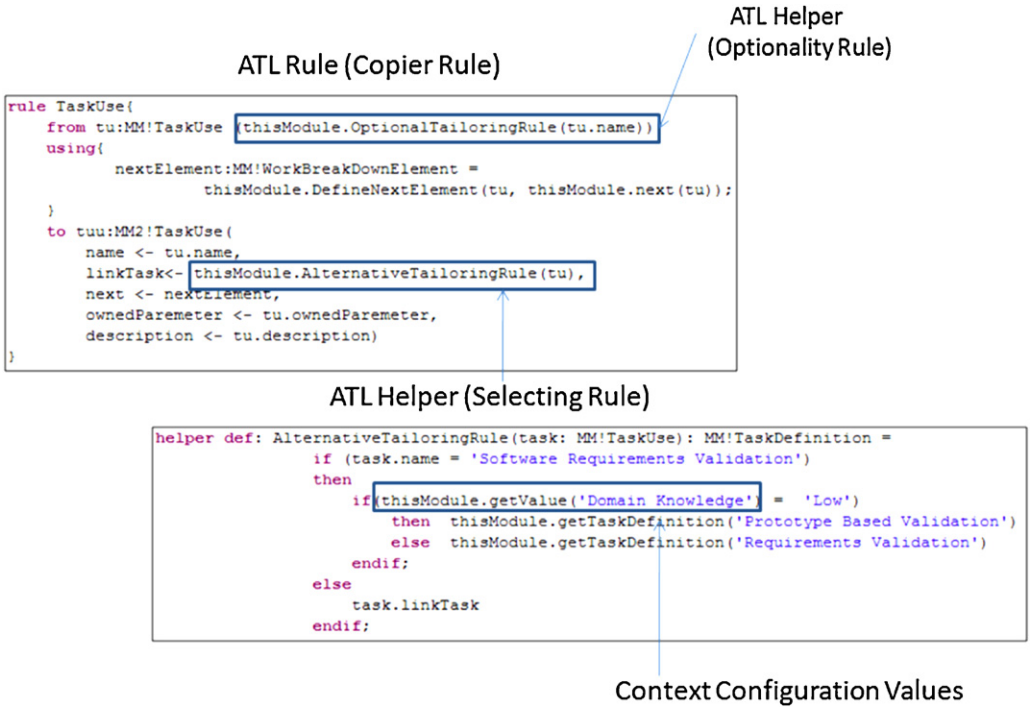

Fig. 9. ATL tailoring transformation.

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.

Our tailoring transformation is endogenous (Czarnecki and Helsen, 2006) because its output conforms to the same metamodel as the input. However, it is not *in place* since we want to preserve the organizational process model for future configurations. We use *ATLCopier*[3] as a basic template, and we modify it so that only those elements whose rules evaluate to true are actually copied to the target model. All common elements of the eSPEM model are copied over to the adapted model, and we specify rules for the variation points. Each variation point has an associated helper called from the matched rule.

Fig. 9 shows the 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. 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 (not shown in Fig. 9) that makes of the context configuration. When this rule returns false, the element needs to be removed from the process. 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 (e.g., *Task Definition*). Therefore, we define an *AlternativeTailoringRule* as a selecting rule that returns the method element chosen according to the helper rule. The *AlternativeTailoringRule* chooses the appropriate *TaskDefinition* variant, according to the Domain Knowledge value in the context (in the figure, this value is "Low"). If there were more variability points, a conjunction of rules would be applied, also specifying priorities to make trade-offs, if necessary.

3.4. Project adapted process model

The project adapted process model also conforms to the eSPEM metamodel, but it cannot have variabilities, so all variabilities identified as part of the organizational process model are resolved by the tailoring transformation. Fig. 10 shows the adapted process after applying the rules in Fig. 9 to an extension project with a low domain knowledge.

3.5. Discussion

This proposal is only applicable in companies that have their software process formalized. Large companies are more likely to have their process formalized, but their survival does not depend on small improvements in productivity. This is not the case for SSO, where inefficiency may lead them to failure. Moreover, even large companies tend to organize their development in smaller entities

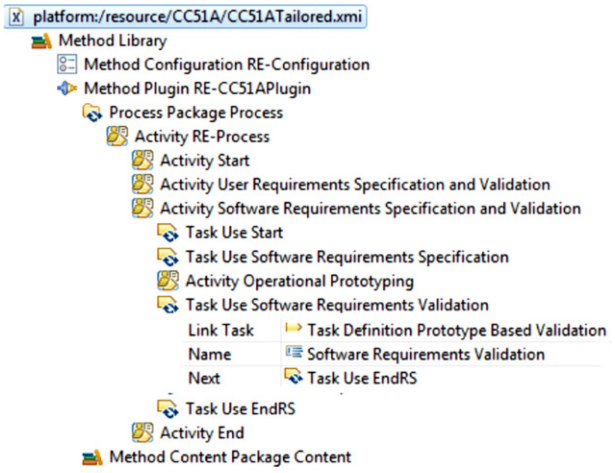

Fig. 10. Adapted process for an extension project in a non-familiar domain.

[3] http://www.eclipse.org/m2m/atl/atlTransformations/.

(Anon., 2011) that can also benefit from our approach. According to OECD (2005), most software companies around the world are small, so the proposal has the potential of having a big impact. In Chile, we have seen that there is a growing interest in process formalization among small software companies (Ruiz et al., 2012).

Software development in small entities may vary according to a small set of variables: there is only a small group of people that may be assigned to a project, there are only one or a few customers, and the projects are always short in duration. Even more, not all the combinations of these variables are valid, e.g., if the type of project is maintenance the customer must be known, if the project is incident the project duration cannot be long, thus there is a limited amount of valid context configurations. Our proposal allows to determine a unique process to be followed once the context configuration is defined, i.e., it is a function:

$$Tailoring : Context \rightarrow Process$$

So, for each context configuration there is a unique process that is the most appropriate one, and for two identical context configurations, the process is exactly the same. Moreover, different contexts may lead to the same process. Therefore, the number of potentially different processes that may be generated is at most equal to the number of context configurations.

When the number of context variables grows, then the number and complexity of the tailoring rules also grows. This makes the modeling activity more difficult, and programming the rules, that is already the most sophisticated part of the approach even more complex. These issues challenge the proposal's adoption, at least until we develop tools for supporting modeling and rule generation.

4. Tailoring the requirements engineering process at KI

We have formalized the general requirements engineering (RE) process used by KI,[4] a small-sized Chilean software company. This company is CMMI Level 2 certified, and its organizational software process is based on RUP. Its process is part of the Tutelkán project (Valdés et al., 2010), and is publicly available.

This RE process is accompanied by a set of adaptation guidelines. These guidelines indicate which artifacts should or should not be included as part of the adapted process, according to certain project context values. This is the way KI has addressed the problem of manually tailoring a general process to a small, predefined set of project types: large development, small development, maintenance and incident. Each project type has been clearly identified by the company's process engineer as a valid project context.

In this section, we first describe the input organizational process and the context model. We then illustrate how the adaptation guidelines are turned into rules. Finally, we show the results of the automated tailoring process, which produced the expected process for each predetermined project type. We also present the results of applying our approach to an unexpected project context: a maintenance-enhancement project for an unknown customer. Our approach successfully generated an adequate process for this context. The results presented in this section have been analyzed and validated by the company's process engineer.

4.1. Organizational RE process model

The studied requirements engineering process consists of two activities, which are carried out in parallel: *Requirements Development* and *Requirements Management*. Fig. 11 shows the organizational process, and Fig. 12 shows its formalization in eSPEM. This process includes the Requirements Engineering and Management activities, and is part of a larger process.

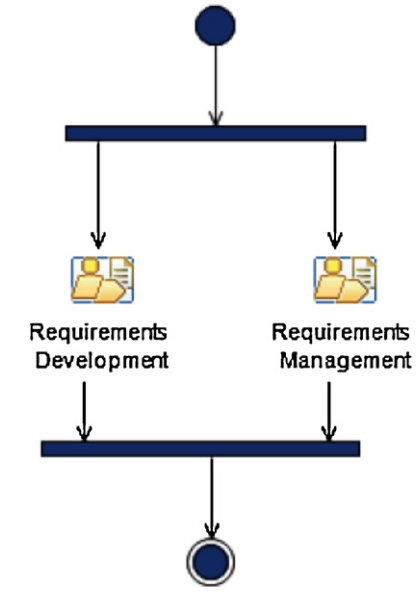

Fig. 11. KI general requirements engineering process.

The *Requirements Development* activity is depicted in more detail in Fig. 13. This activity may take on two different forms, depending on whether or not the project is in its inception phase. In the first case, the *Requirements Development* activity consists of two optional sub-activities that can be carried out in parallel: *Problem Analysis* and *Environment Specification*. In all other phases, this activity is broken down into three sub-activities that are carried out in parallel: *Requirements Specification*, *Requirements Analysis and Validation*

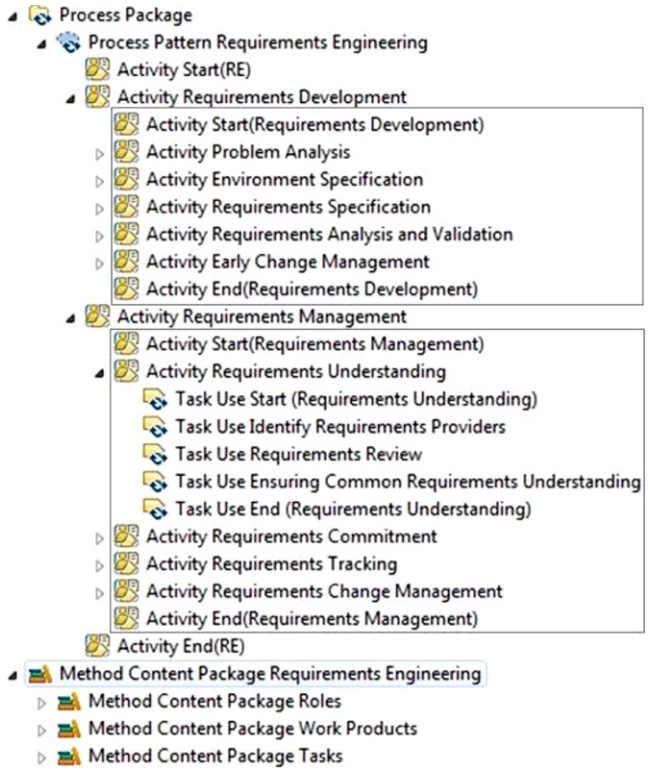

Fig. 12. KI – eSPEM specification of the general requirements engineering process.

[4] http://www.kiteknology.com.

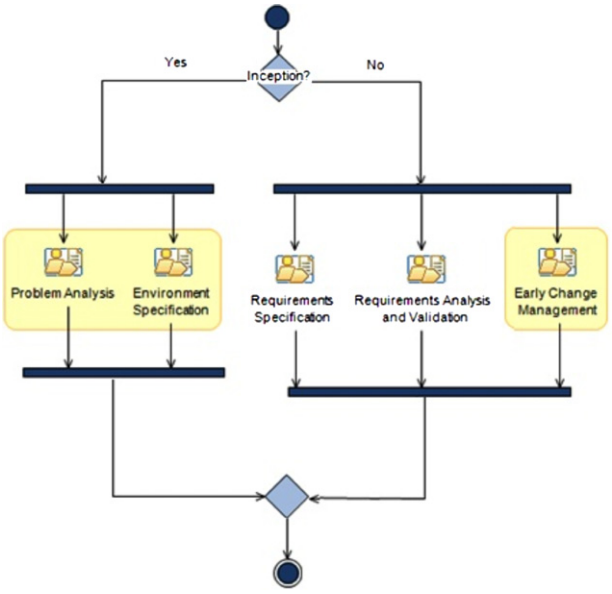

Fig. 13. KI – the *Requirements Development* activity.

and *Early Change Management*; only the last one is optional. Yellow rectangles are used to visually indicate process element optionality.

The *Requirements Management* activity consists of four sub-activities: the *Requirements Understanding* and *Requirements Commitment* activities are first carried out sequentially, after which the *Requirements Tracking* and *Requirements Change Management* activities are carried out in parallel (see Fig. 14). None of these sub-activities are optional. The *Requirements Understanding* activity is further subdivided into three sequential sub-tasks: *Identify Requirements Providers*, *Requirements Review* and *Ensuring Common Requirements Understanding* (see Fig. 15). Note that the *Identify Requirements Providers* task is marked as optional. This task is only carried out if the project is a new development project. Finally, the feature model in Fig. 16 shows both the common and variable process elements of the General Requirements Engineering process presented in this section.

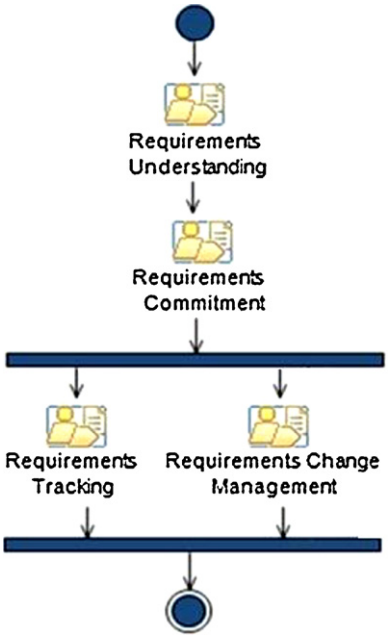

Fig. 14. KI – the *Requirements Management* activity.

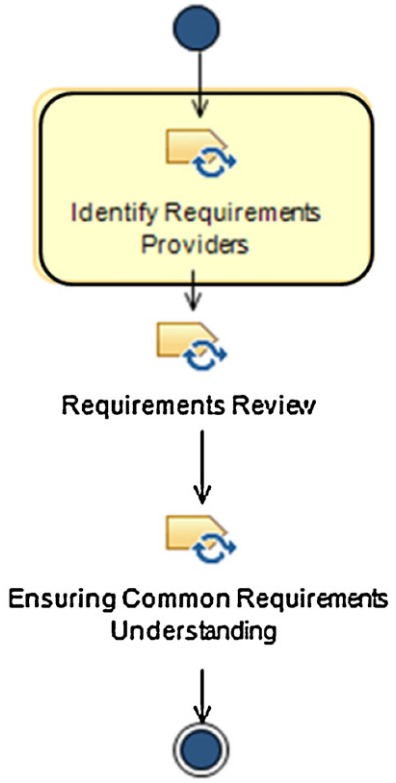

Fig. 15. KI – the *Requirements Understanding* activity.

4.2. Context model

The General Requirements Engineering process presented in the previous subsection is applied to different types of projects. We have defined a context model for KI, shown in Fig. 17. This context model has three dimensions: *Domain*, *Team* and *Management*. For example, the *Domain* dimension has three attributes: *Application Domain*, *Development Environment* and *Source of Documentation*. The first two attributes can take on two possible values: "known" or "unknown", whereas the *Source of Documentation* has three possible values: "exist", "does not exist" and "expert" (an expert is the source of information). The other two dimensions are similarly disaggregated into attributes, each with its own possible values.

We show the context characterization of two typical projects in Table 2. The parts of the context model that are relevant to the tailoring process are listed in the first two columns, and each of the remaining columns represents a particular project context characterization. For example, the third column lists the context attribute values that characterize a small, new development project, within an unknown application domain. In this type of project there is no preexisting documentation, both the development environment and customer type are unknown, the provider is in-house, and the project duration is small (short). We expect that the RE process tailored to such a project includes all optional tasks, roles and work products, as this is one of the most complex types of project that KI develops.

On the other hand, the attribute values listed in the fourth column of Table 2 describe a simple corrective-maintenance project, where the application domain, development environment and customer type are known, documentation exists, the provider is in-house and the project duration is medium. In this case, we expect that the resulting tailored process will be much simpler than the one for small development projects: during the Inception phase, the *Requirements Development* activity just includes the *Problem Analysis* sub-activity (in addition to the mandatory sub-activities),

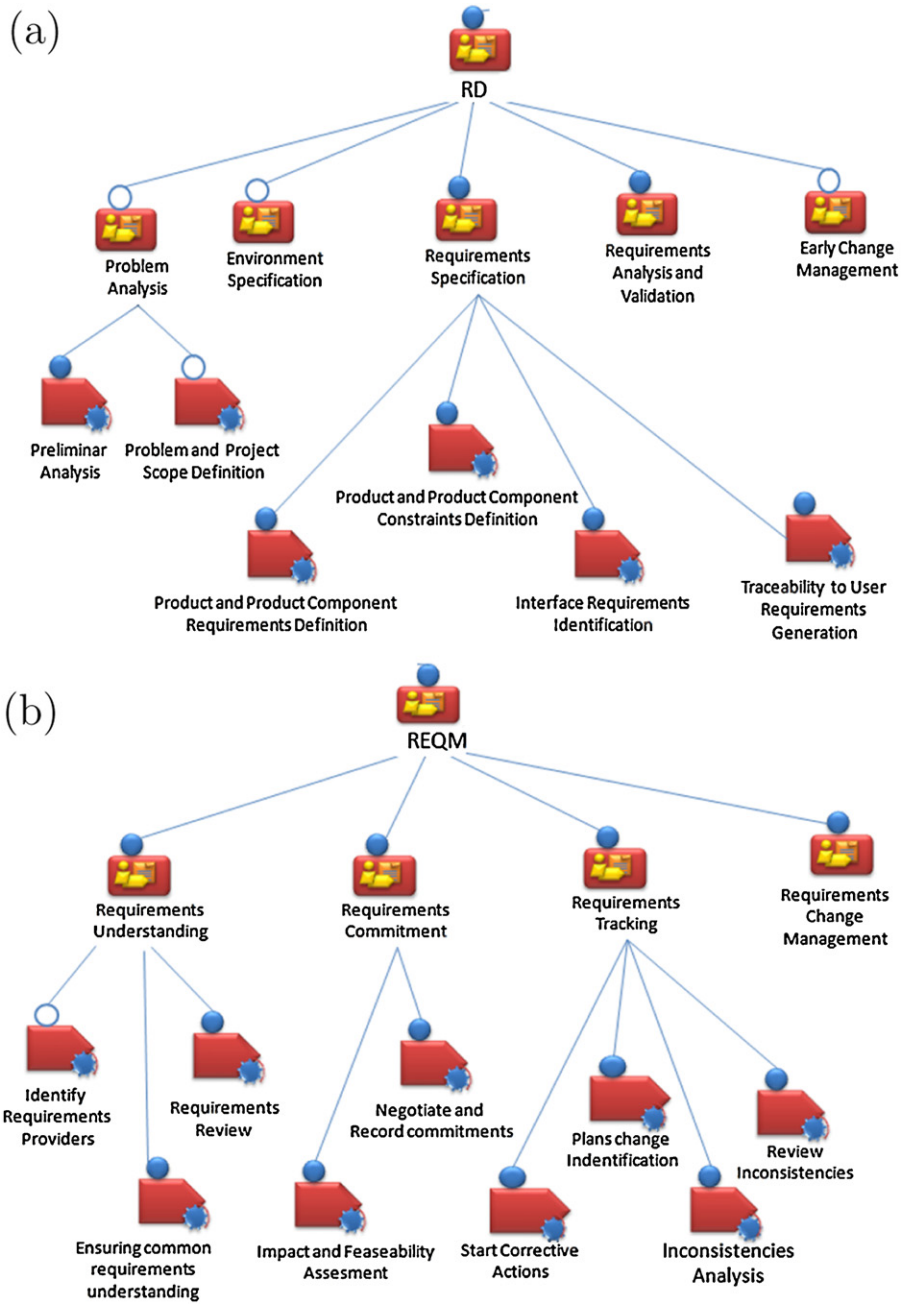

Fig. 16. KI – feature model corresponding to: (a) the *Requirements Development* activity and (b) the *Requirements Management* activity.

while in the rest of the phases, the *Requirements Development* activity just consists of the two mandatory sub-activities *Requirements Specification* and *Requirements Analysis and Validation*.

The *Requirements Understanding* activity for both types of project should consist of just two sequential activities: *Requirements Review* and *Ensuring Common Requirements Understanding*. This is because the requirements provider is "in-house". Fig. 18 shows the resulting *Requirements Development* and *Requirements Understanding* workflows, when tailored to the Simple Maintenance project context.

Table 2
KI – context characterizations of two typical projects.

Dimension	Context attribute	Small development	Simple Maintenance
Domain	Application domain	Unknown	Known
	Development environment	Unknown	Known
	Source of documentation	Does not exist	Exists
Management	Project type	New development	Maintenance-correction
	Provider	In-house	In-house
	Customer type	Unknown	Known
	Project duration	Small	Medium

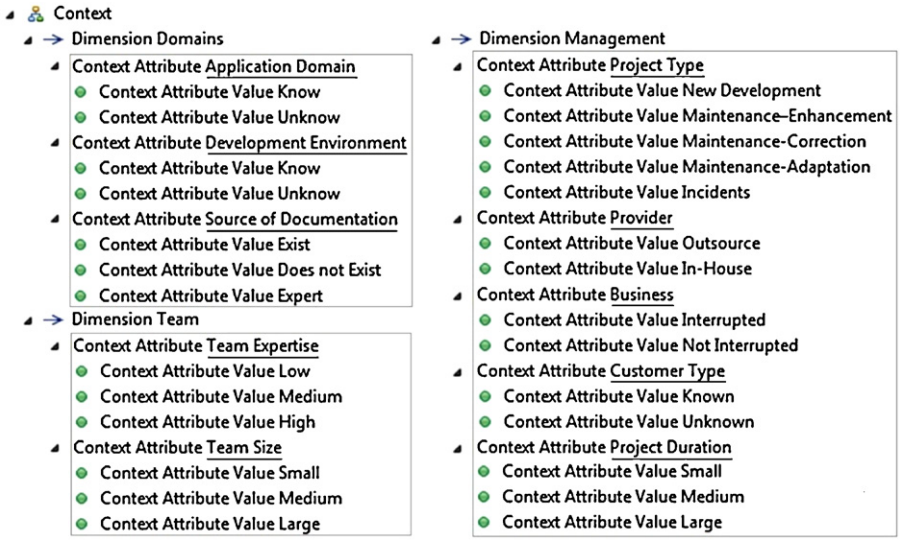

Fig. 17. KI – context model.

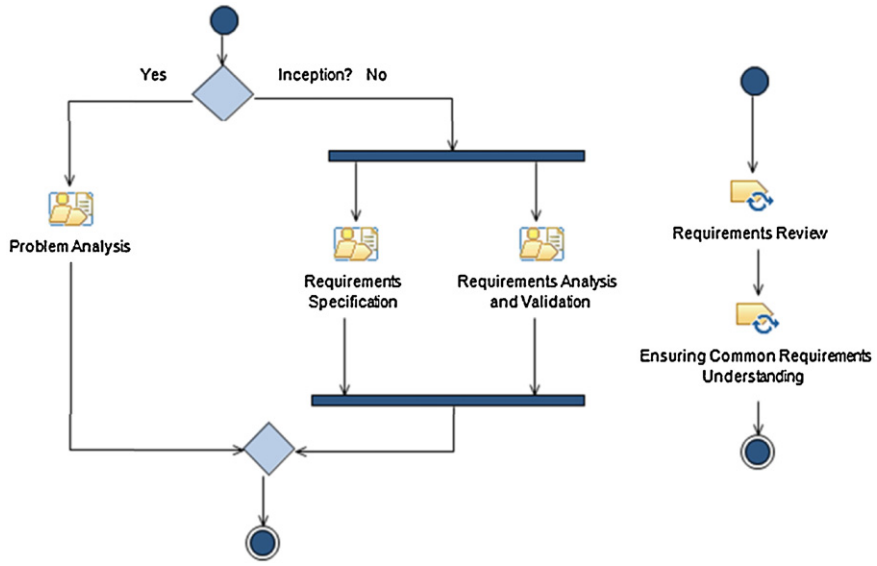

Fig. 18. KI – the *Requirements Development* and *Requirements Understanding* activities, tailored to the Simple Maintenance project context.

4.3. Tailoring

KI has a set of adaptation guidelines, which they use to manually tailor their process. Table 3 is a partial listing of the adaptation guidelines provided by KI. These guidelines are expressed as guarded actions, where the process elements that appear in the "Action" column are those variation points indicated in Fig. 16. For example, if we want to tailor the general requirements process to a maintenance-enhancement project, then the *Problem and Project Scope Definition* task is required, i.e., the tailored process must include this task.

The adaptation guidelines for common contexts tend to be clear enough so that there is no ambiguity about what to expect in the adapted process. For example, the *Early Change Management* activity should never be carried out during a maintenance-correction project. However, certain attribute combinations are not fully documented in these guidelines. For example, if the Provider is in-house, then the *Problem and Project Scope Definition* task may or may not be required, depending on the values of other context attributes. In other situations, like the case in which there is a Source of Documentation ("Exist"), no specific action has been specified. Fig. 19 presents, as an example, the ATL implementation of Rule 2, which

Table 3
KI – adaptation guidelines (partial listing).

Context attribute	Value	Action
Project type	Maintenance-enhancement	*Problem and Project Scope Definition* task required
Project type	Maintenance-correction	*Early Change Management* activity not required
Provider	In-house	*Problem and Project Scope Definition* task may be required
Provider	Outsource	*Problem and Project Scope Definition* task required
Source of documentation	Does not exist	*Environment Specification* activity may be required
Source of documentation	Exists	No action suggested

```
-- 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;
```

Fig. 19. ATL rule, determines whether the *Environment Specification* activity is required.

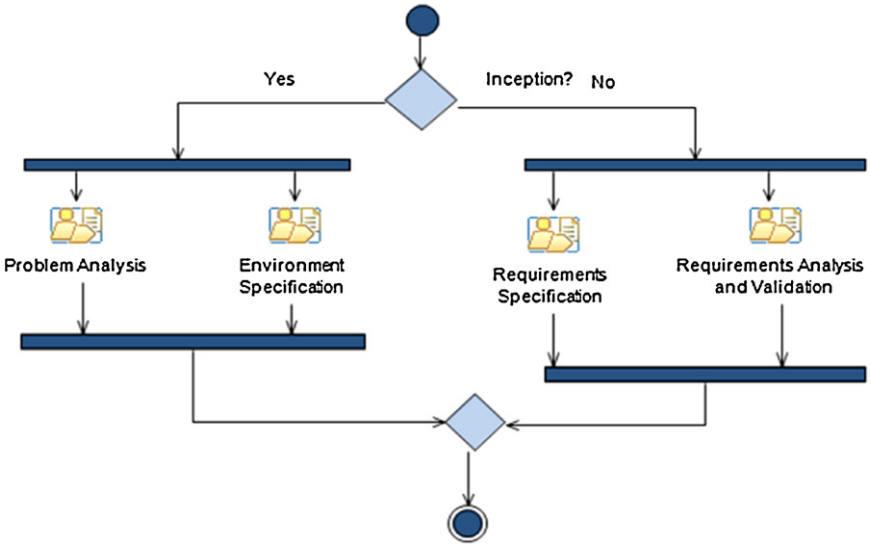

Fig. 20. KI – the *Requirements Development* process when no documentation exists.

determines when the *Environment Specification* activity is included in the tailored software process. If the rule returns "true", the activity must be included in the process.

Let us now consider the case of an unexpected context, i.e., a valid context configuration that was not recognized by the company as a typical project context. For example, KI could receive a maintenance-enhancement request from an unknown customer. This project context is similar to Simple Maintenance context (fourth column, Table 2); however, no documentation exists for this project (source of documentation does not exist). Therefore, this type of maintenance-enhancement represents a new project context for the organization. Although the adaptation guidelines do not contemplate this case, the guarded actions listed in Table 3 can be used to make some decisions about the variation points.

Table 4
KI – maintenance-enhancement without documentation.

Context attribute	Attribute value
Application domain	Known
Development environment	Known
Source of documentation	Does not exist
Project type	Maintenance-enhancement
Provider	In-house
Customer type	Known
Project duration	Medium

The context configuration for this new type of project is shown in Table 4. We can now apply the tailoring rules.

Fig. 20 shows the result of applying our tailoring rules to this new context. This process includes the *Environment Specification* activity, which was not included in the process configured to Simple Maintenance (see Fig. 18), since Rule 2 indicates that this activity must only be included when no documentation exists. According to the company's process engineers, this is the expected result, even though it was not explicitly stated in the original adaptation guidelines.

4.4. Evaluation

Our MDE-based tailoring solution was validated during a 4-h workshop that we organized at KI. Participants included the company's CEO, process engineer, and project managers. At this workshop, we presented the technical aspects of our proposal, as well as a demonstration of our tool chain. We efficiently generated each possible adaptation of the process, and validated each one in conjunction with the organization's process engineer. The tailored processes we produced were correct and suitable for each particular project context according to the company's stakeholders. Also, since the tailoring process is now replicable and fast, the company is considering to include this as the first step of any new project.

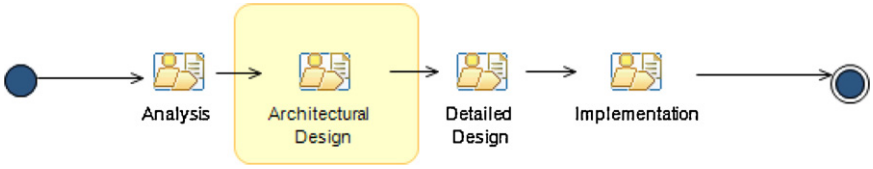

Fig. 21. Amisoft – organizational software process.

5. Tailoring the software process at Amisoft

In this section, we present the application of our approach to the development process of Amisoft,[5] another 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 software development process is based on the OpenUP process model.

In order to apply the proposed tailoring approach, we first formalized the organizational software process in EMF, with the help of the company's process engineer. As in the previous case study, process variability was modeled in a feature model. We then formalized the context model for Amisoft, as well as two context characterizations. After that, we defined the tailoring rules that specify how the process must change according to the context. Finally, we describe how our proposed approach was used to tailor the organizational software process to one of the project context characterizations.

5.1. Organizational process model

Amisoft's organizational process considers two predefined project types: *development* and *maintenance*. Since this organization typically addresses small software projects, particularly information systems, its development process is simple. Fig. 21 shows the specification of this process, which consists of four sequential activities: *Analysis*, *Architectural Design*, *Detailed Design* and *Implementation*.

The *Analysis* activity involves four tasks: *Elicitation*, *Requirements Analysis*, *Prototyping* and *Requirements Specification* (see Fig. 22). The *Elicitation* and *Requirements Analysis* tasks focus on determining the users' requirements, while the *Requirements Specification* task focuses on identifying the project's software requirements. If the project involves the development of a new software product the *Prototyping* task is carried out, helping validate users' requirements and translate them into software requirements. However, this task is not required for maintenance projects, thus it is optional. The *Requirements Specification* task should always be carried out; however, a formal requirements document is only produced when a new system is being developed, or when the development team does not have a lot of experience. For these reasons, the organization also considers this task as optional.

The *Architectural Design* activity is optional, because it is not required for maintenance projects. This activity does not have any variability in its workflow, and thus we do not show the corresponding model.

The *Detailed Design* activity starts with two sequential tasks (*Design Conception* and *Design Reuse*) and ends with three concurrent tasks (*Components Design*, *Database Design* and *Interfaces and Communication Design*). The specification of this activity is shown in Fig. 23. During the *Design Conception* task, designers determine which components must be part of the solution and during the *Design Reuse* task, they identify which component designs can be reused from previous projects. These two initial tasks are optional, since they are not required during maintenance projects. *Components Design* and *Database Design* are mandatory tasks. The first one addresses the modeling of components that have yet to be designed, while the second one focuses on the database or dataspace model. Since this company typically develops information

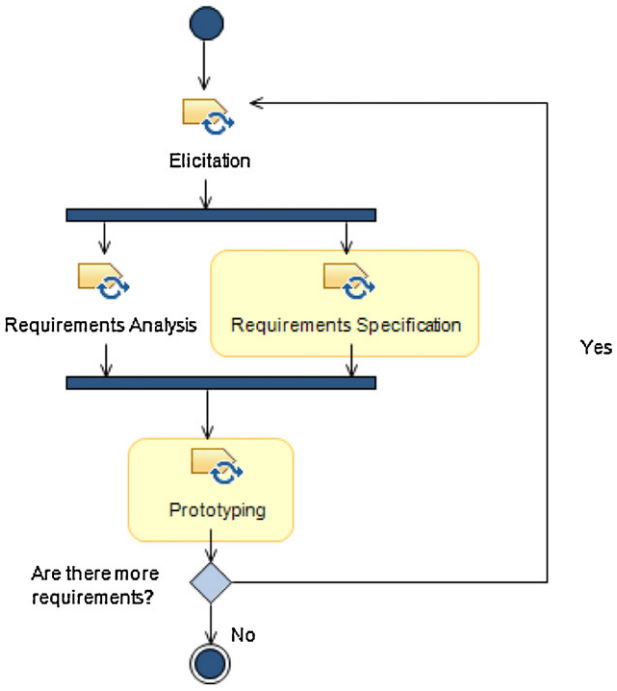

Fig. 22. Amisoft – the *Analysis* activity.

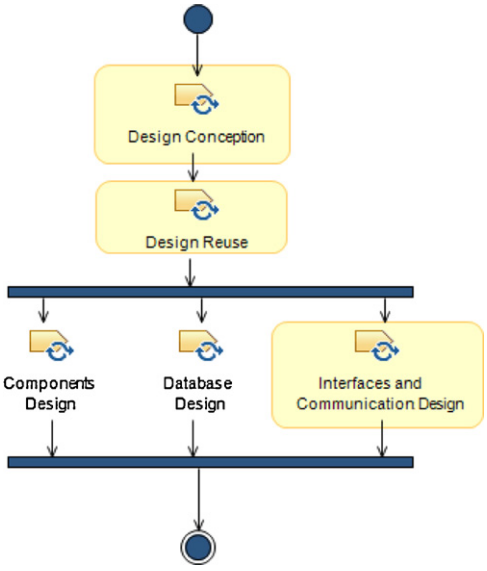

Fig. 23. Amisoft – the *Detailed Design* activity.

[5] http://www.amisoft.cl.

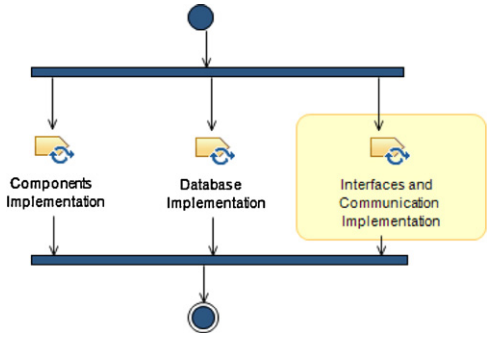

Fig. 24. Amisoft – the *Implementation* activity.

systems, the *Database Design* task is mandatory. Some maintenance projects do not require database changes, but even then this task is carried out, since changes made to the software may trigger some database adjustments, e.g., requiring the creation of new indexes. The *Interfaces and Communication Design* task deals with the Human–Computer Interaction aspects of the software product and also communication with other systems. This task is optional because it is not required during maintenance projects.

Finally, the *Implementation* activity consists of three concurrent tasks: *Components Implementation*, *Database Implementation* and *Interfaces and Communication Implementation* (see Fig. 24). During these tasks, the roles responsible for them implement the designs produced during the *Detailed Design* activity.

Fig. 25 is the feature model associated to this software process. There are two *requires* constraints between the *Interfaces and Communication Design* and *Interfaces and Communication Implementation* tasks: (1) from the implementation to the design task, which models that the design task must be carried out before coding can occur and (2) from the design to the implementation task, which is to ensure that the design is not a waste work product.

5.2. Context model

Four dimensions were identified by the organization as relevant for characterizing its projects: *Team*, *Project*, *Business* and *Product*. The corresponding context model is shown in Fig. 26. The *Team* dimension has two attributes: *Technical Knowledge* and

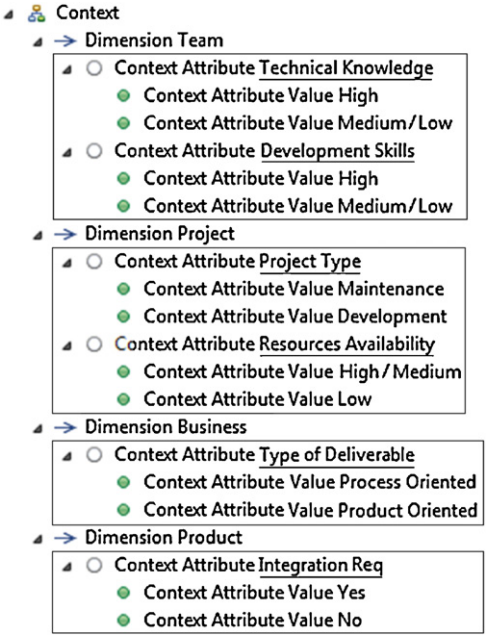

Fig. 26. Amisoft – context model.

Development Skills, which model the team's required level of technical knowledge and development skills, respectively. The *Project* dimension also has two attributes: *Project Type*, which can take on one of two values ("Maintenance" or "Development"), and the *Resources Availability* that models the level of availability of human resources. The *Business* dimension considers just one attribute: *Type of Deliverable*, which can be "process-oriented" or "product-oriented". When this attribute is set to "process-oriented", a deliverable must be generated by each phase of the process (e.g., a requirements specification, a software architecture document, and so on). When the attribute value is "product-oriented", only the source code of the product must be delivered to the client organization. Finally, the *Product* dimension considers just an *Integration Req* attribute, which indicates whether the project must be integrated to an existing solution.

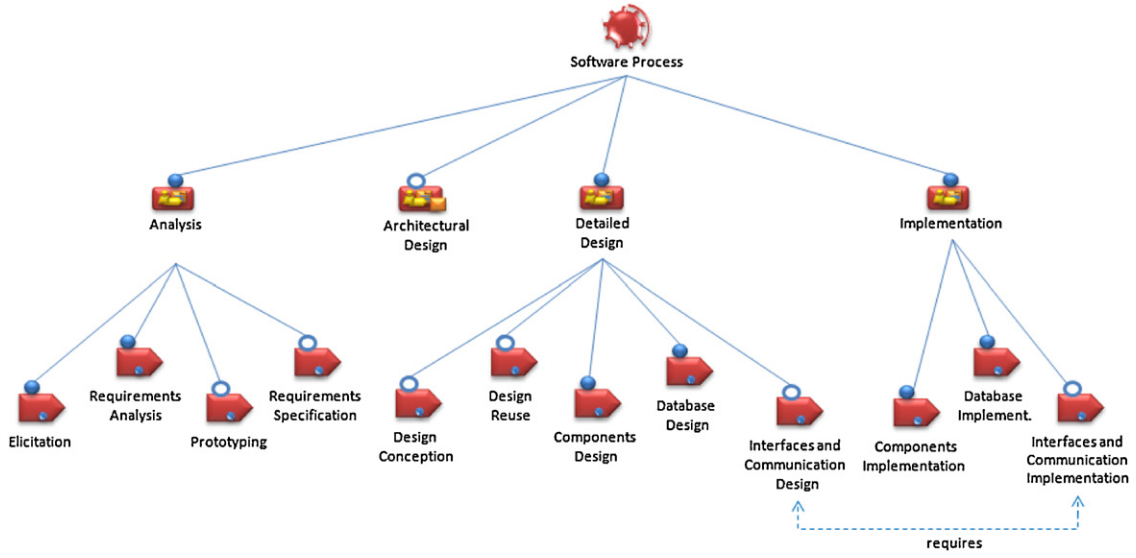

Fig. 25. Amisoft – feature model associated to the organization's software process.

			Analysis				Architectural Design	Detailed Design				
Dimension	Attribute	Attribute Values	Elicitation	Req. Analysis	Prototyping	Requirements Specification		Design Conception	Design Reuse	Components Design	Database Design	Interfaces & Comm. Design
Team	Technical Knowledge	High			True-							
		Medium/Low										
	Development Skills	High										
		Medium/Low			True-	True-						
Project	Project Type	Development	True	True-	True	True-	True	True	True	True	True	
		Maintenance	True	True-	False		False	False	False	True	True	False-
	Resources Availability	High/Medium										
		Low				False						
Business	Type of Deliverables	Process-oriented				True+						
		Product-oriented										
Product	Integration Requirements	Yes										False
		No										True

Fig. 27. Amisoft – tailoring decisions.

```
-- Rule ReqSpecification - Development of a Formal Requirements Specification
helper def:ReqSpecification(elementName:String): Boolean =
    if('Requirements Specification' = elementName) then
        if(thisModule.getValue('Type of Deliverables') = 'Process-oriented') then
            true
        else
            if((thisModule.getValue('Resources Availability') = 'Low') then
                false
            else
                  if((thisModule.getValue('Development Skills') = 'Medium/Low') or
                     (thisModule.getValue('Project Type') = 'Development')) then
                      true
                  else
                      false
                  endif
             endif
        endif
     else true
     endif;
```

Fig. 28. ATL tailoring rule, determines whether the *Requirements Specification* task is required.

5.3. Tailoring rules

The tailoring rules for this process were specified using a table that relates the context dimensions and attributes to process features. Fig. 27 shows a partial view of this tailoring table. Since all the rules are specified in a single graphical representation, the process of validating when to apply each rule becomes easy to carry out. The intersection between each row and column characterizes the relationship between the corresponding attribute value (row) and process variation point (column). A cell is left empty if there is no relation between the referenced context attribute and variation point. Otherwise, we use the values *True* and *False* to indicate whether the referenced task must be included or removed from the tailored process, respectively. We additionally extend these values with +and − symbols, indicating whether a context attribute has a higher or lower priority with respect to other attributes.

For example, the *Requirements Specification* task is weakly selected when the team has medium/low *Development Skills*. However, this task will be strongly omitted from the tailored process if the project is product-oriented, i.e., the *Requirements Specification* task will not be included in the tailored process for this type of project. These tailoring guidelines were formalized using ATL. Fig. 28 shows a helper rule that determines when the *Requirements Specification* task must be included in this organization's tailored software process. Here, the +and − values determine the order in which the context attributes are evaluated: *Type of Deliverable* is evaluated first (*True+*), while the *Development Skills* and *Project Type* attributes are evaluated last.

5.4. Context-adapted processes

Table 5 presents two examples of project characterizations: Simple and Complex Maintenance. Both are maintenance projects (Project Type is "Maintenance"), but the first one does not require a high-level of technical knowledge or development skills, while the second one does. In both cases, the only deliverable is the code (Type of Deliverable is set to "Product-oriented"). The first context is probably the simplest type of project that this company deals with, while the second one is a bit more demanding.

As in the first case study, we tailored the general process using the ATL rules derived from the tailoring decisions specified in Fig. 27. The resulting processes are shown in Fig. 29. In the first

Table 5
Amisoft – two project characterizations.

Context attribute	Simple Maintenance	Complex Maintenance
Technical knowledge	Medium/low	High
Development skills	Medium/low	High
Project type	Maintenance	Maintenance
Resources availability	Low	Low
Type of deliverables	Product-oriented	Product-oriented
Integration req	No	No

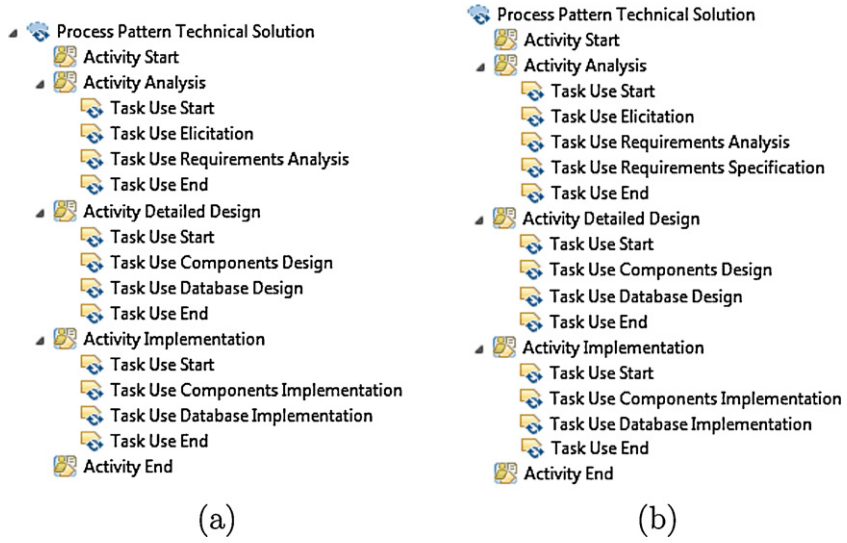

Fig. 29. Amisoft – the tailored process model for (a) a Simple Maintenance and (b) a Complex Maintenance project, respectively.

one, all optional process elements have been removed, as expected; whereas the second one is similar, but also includes the *Requirements Specification* task.

The adapted processes were analyzed by Amisoft's process engineer, who considered that they are appropriate for the corresponding contexts, and they are also consistent with the guidelines defined by the organization. In his opinion, the most valuable aspect of our proposed tailoring methodology is its usability, since he could quickly adjust the context attribute values and immediately see how these changes affect the tailored processes.

6. Conclusions and future work

In this article, we present an MDE-based approach for automatically tailoring software processes, where transformation rules are used to adapt a general process model to a specific context.

The tailoring strategy takes as input two models, a general process model and a project context model, and produces a context-specific process. This is done by applying a set of transformation rules that define how the general process model varies depending on the context. These rules are specified by a process engineer and validated by the organization's project managers. The project manager is afterwards only responsible for specifying the specific project's context.

This approach has a number of advantages. Tailoring is done in a systematic way because the transformation rules are uniformly applied according to a context model. The total time required to adapt a process is much shorter than before, so the process engineer can quickly see the effect of changing a rule or a context attribute value. We can also check the consistency of the process engineer's tailoring know-how. In practice, we also expect to see an improvement in productivity in projects that follow processes adapted using our approach. This is because the tailored process only includes the process elements that are strictly required by a project's context configuration, so only the essentially required resources should be invested in the project.

The case studies presented in this paper showed that it is possible to apply tailoring transformations built for adapting a general process to different project contexts in a planned manner. Being able to validate the transformations for particular known cases has given us confidence of their validity in the general case. Therefore, whenever unanticipated scenarios happen, a combination of already built and validated tailoring transformations can be applied, and as a consequence, an appropriate context adapted process can be obtained quickly and easily.

The experience has allowed us to conclude that: (1) the proposed technique is an effective tool to achieve process tailoring, (2) the approach is useful and practical because it was easily implementable by the process group, and (3) it allows to represent and evolve the tailoring knowledge of an organization. However, (4) the prototypical tool must be more usable, in particular to define the transformation rules. Additionally, the process engineers at one of the companies suggested that the triplet (Context Configuration, Tailored Process and Results) should be used to empirically validate and improve the context model and the tailoring decisions.

Although the case studies were conducted in niche companies with small processes where the number of the context variables and the number of process variation points is small, we expect that the approach could be successfully applied for larger processes as well. Generally, large companies are organized in smaller entities which behave much like small companies (Hamilton, 1999). Nevertheless, when a large company is managed as a whole, the number of processes from the process line grows exponentially according to the number of variation points and context configurations, and this adds complexity. Performance in process tailoring is not an issue since it is done automatically, but writing the rules and evolving them consistently may become difficult and expensive. Further validation with big process models is required to evaluate the actual scalability of the MDE approach.

We are currently experimenting with this approach in four other Chilean software companies, as part of the ADAPTE project,[6] a large, government-funded initiative. This will allow us to continue validating the feasibility of the proposed approach. Since the quality of the input models is important, we are currently integrating our tool chain with some process quality analysis tools we have already developed (Hurtado et al., 2012a). Moreover, we are now working on also improving methodological and usability issues of the proposed tailoring approach.

Acknowledgement

This work has been partly funded by project Fondef D09I1171 of Conicyt, Chile.

[6] http://www.adapte.cl.

References

Anon., 2011. Software Engineering – Lifecycle Profiles for Very Small Entities (VSEs) – Part 5-1-2: management and engineering guide: generic profile group: basic profile. Technical Report ISO/IEC TR 29110-5-1-2:2011, International Organization for Standardization, Geneva.

Aranda, J., Easterbrook, S., Wilson, G.,2007. Requirements in the wild: how small companies do it. In: Proceedings of the 15th IEEE Requirements Engineering Conference. IEEE CS Press, pp. 39–48.

Armbrust, O., Katahira, M., Miyamoto, Y., Münch, J., Nakao, H., Ocampo, A., 2009. Scoping software process lines. Software Process: Improvement and Practice 14 (3), 181–197.

Bai, X., Huang, L.G., Zhang, H., 2010. On scoping stakeholders and artifacts in software process. In: Münch, J., Yang, Y., Schäfer, W. (Eds.), Proceedings of the 2010 international conference on New modeling concepts for today's software processes: software process (ICSP'10). Springer-Verlag, Berlin, Heidelberg, pp. 39–51.

Bajec, M., Vavpotic, D., Furlan, S., Krisper, M.,2007a. Software process improvement based on the method engineering principles. In: Situational Method Engineering: Fundamentals and Experiences, Proceedings of the IFIP WG 8.1 Working Conference, Volume 244 of IFIP. Springer, pp. 283–297.

Bajec, M., Vavpotic, D., Krisper, M., 2007b. Practice-driven approach for creating project-specific software development methods. Information and Software Technology 49 (4), 345–365.

Batista, J., De Figueiredo, A.D., 2000. SPI in a very small team: a case with CMM. Software Process: Improvement and Practice Journal 5 (4), 243–250.

Beck, K., Andres, C., 2004. Extreme Programming Explained: Embrace Change, 2nd ed. Addison-Wesley Professional.

Belkhatir, N., Estublier, J., 1996. Supporting reuse and configuration for large scale software process models. In: Software Process Workshop, 1996. Process Support of Software Product Lines, Proceedings of the 10th International, pp. 35–39.

Boehm, B.W., Clark, B., Horowitz, E., Westland, J.C., Madachy, R.J., Selby, R.W., 1995. Cost models for future software life cycle processes: COCOMO 2.0. Annals of Software Engineering 1, 57–94.

Breton, E., Bézivin, J., 2001. Model driven process engineering. In: Computer Software and Applications Conference, 2001. COMPSAC 2001, pp. 225–230.

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). IEEE Computer Society, Washington DC, USA, pp. 357–362.

Clements, P., Northrop, L., 2001. Software Product Lines: Practices and Patterns, 3rd ed. Addison-Wesley Professional.

CMMI Product Team CMMI for Development, Version 1.2, 2006. Technical Report CMU/SEI-2006-TR-008, Software Engineering Institute.

Cockburn, A., 2000. Selecting a project's methodology. IEEE Software 17 (4), 64–71.

European Commission, 2005. The New SME Definition: User Guide and Model Declaration. http://ec.europa.eu/enterprise/policies/sme/files/sme_definition/sme_user_guide_en.pdf

Cusumano, M.A., Cormack, A.M., Kemerer, C.F., Crandall, W.(B.), 2009. Critical decisions in software development: updating the state of the practice. IEEE Software 26 (5), 84–87.

Czarnecki, K., Helsen, S., 2006. Feature-based survey of model transformation approaches. IBM Systems Journal 45 (3), 621–645.

Dai, F., Li, T., 2007. Tailoring software evolution process. In: 8th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, vol. 2, pp. 782–787.

Deelstra, S., Sinnema, M., Bosch, J., 2005. Product derivation in software product families: a case study. Journal of Systems and Software 74 (2), 173–194.

Dörr, J., Adam, S., Eisenbarth, M., Ehresmann, M., 2008. Implementing requirements engineering processes: using cooperative self-assessment and improvement. IEEE Software 25 (3), 71–77.

Firesmith, D., 2004. Creating a project-specific requirements engineering process. Journal of Object Technology 3 (5), 31–44.

Hamilton, M., 1999. Software Development: Building Reliable Systems. Enterprise Computing Institute Series. Prentice Hall.

Harris, M., Aebischer, K., Klaus, T., 2007. The whitewater process: software product development in small IT businesses. Communications of the ACM 50 (5), 89–93.

Henderson-Sellers, B., Ralyté, J., 2010. Situational method engineering: state-of-the-art review. Journal of Universal Computer Science 16 (3), 424–478.

Hofer, C., 2002. Software development in Austria: results of an empirical study among small and very small enterprises. In: Proceedings of the 28th Euromicro Conference, pp. 361–366.

Hurtado, J.A., Bastarrica, M.C., 2009. Process model tailoring as a mean for process knowledge reuse. In: 2nd Workshop on Knowledge Reuse, KREUSE, Falls Church, Virginia, USA.

Hurtado, J.A., Bastarrica, M.C., Bergel, A., 2012a. Analyzing software process models with AVISPA. In: Raffo et al. (2011), pp. 23–32.

Hurtado, J.A., Bastarrica, M.C., Quispe, A., Ochoa, S.F., 2012b. An MDE approach to software process tailoring. In: Raffo et al. (2011), pp. 43–52.

Hurtado, J.A., Ochoa, S.F., Quispe, A., Bastarrica, M.C., 2011. A context modeling language to support tailoring of software processes. Technical Report TR/DCC-2011-14, Computer Science Department, University of Chile.

Jacobson, I., Booch, G., Rumbaugh, J., 1999. The Unified Software Development Process. Addison-Wesley Longman Publishing Co. Inc.

Jantunen, S.,2010. Exploring software engineering practices in small and medium-sized organizations. In: Proceedings of the 2010 ICSE Workshop on Cooperative and Human Aspects of Software Engineering (CHASE'10). ACM, South Africa, pp. 96–101.

Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I., Valduriez, P.,2006. ATL: a QVT-like transformation language. In: Companion to the 21th Annual ACM SIGPLAN Conference on OOPSLA'2006. ACM, pp. 719–720.

JTC 1 Information technology /SC 7, 2008. ISO/IEC 12207:2008 systems and software engineering – software life cycle processes. Technical Report, International Organization for Standarization ISO.

Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S., 1990. Feature-Oriented Domain Analysis (FODA). Feasibility study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute.

Kettunen, P., Laanti, M., 2005. How to steer an embedded software project: tactics for selecting the software process model. Information and Software Technology 47, 587–608.

Killisperger, P., Stumptner, M., Peters, G., Grossmann, G., Stückl, T., 2009. Meta model based architecture for software process instantiation. In: Trustworthy Software Development Processes, International Conference on Software Process, ICSP 2009, LNCS 5543, pp. 63–74.

Koolmanojwong, S., Boehm, B.W., 2012. The incremental commitment model process patterns for rapid-fielding projects. In: Münch et al. (2010), pp. 150–162.

Kornyshova, E., Deneckère, R., Claudepierre, B.,2010. Contextualization of method components. In: Proceedings of the Fourth International Conference on Research Challenges in Information Science (RCIS). IEEE CS Press, pp. 235–246.

Kruchten, P., 2003. The Rational Unified Process: An Introduction. Addison-Wesley Longman Publishing Co. Inc., Boston, MA, USA.

Laplante, P.A., Neill, C.J., 2004. Opinion: the demise of the waterfall model is imminent. ACM Queue 1 (10), 10–15.

Ma, J., Wang, Y.,2006. A quantitive context model of software process patterns and its application method. In: Proceedings of the Sixth International Conference on Quality Software. IEEE Computer Society, pp. 243–250.

Martínez-Ruiz, T., García, F., Piattini, M., Münch, J., 2011. Modelling software process variability: an empirical study. IET Software 5 (2), 172–187.

Mirbel, I., Ralyté, J., 2006. Situational method engineering: combining assembly-based and roadmap-driven approaches. Requirements Engineering 11 (1), 58–78.

Münch, J., Yang, Y., Schäfer, W. (Eds.), 2010. International Conference on Software Process, ICSP 2010. Proceedings, Volume 6195 of LNCS. Paderborn, Germany, July 8–9, 2010. Springer.

Ocampo, A., Bella, F., Münch, J., 2005. Software process commonality analysis. Software Process: Improvement and Practice 10 (3), 273–285.

OECD, 2005. Small and medium enterprise (sme) outlook report. Technical Report, Organisation for Economic Co-operation and Development.

OMG, 2007. Software process engineering metamodel SPEM 2.0 OMG beta specification. Technical Report ptc/07-11-01, OMG.

Osterweil, L.J., 1987. Software processes are software too. In: 9th International Conference on Software Engineering, ICSE'1987, pp. 2–13.

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.

Pedreira, O., Piattini, M., Luaces, M.R., Brisaboa, N., 2007. A systematic review of software process tailoring. SIGSOFT Software Engineering Notes 32 (3), 1–6.

Pérez, G., El Emam, K., Madhavji, N.H., 1995. Customising software process models. In: 4th European Workshop Software Process Technology, EWSPT'95, Volume 913 of LNCS, Springer, pp. 70–78.

Pohl, K., Böckle, G., van der Linden, F.J., 2005. Software Product Line Engineering: Foundations, Principles and Techniques, 1st ed. Springer.

Raffo, D., Pfahl, D., Zhang, L. (Eds.), 2011. International Conference on Software and Systems Process, ICSSP 2011. Proceedings. ACM, Honolulu, HI, USA, May 21–22, 2011.

Ralyté, J., Deneckère, R., Rolland, C.,2003. Towards a generic model for situational method engineering. In: CAiSE 2003, LNCS 2681. Springer-Verlag, pp. 95–110.

Rolland, C.,2009. Method engineering: state-of-the-art survey and research proposal. In: Proceeding of the 2009 Conference on New Trends in Software Methodologies, Tools and Techniques. IOS Press, Amsterdam, The Netherlands, pp. 3–21.

Rolland, C., Nurcan, S.,2010. Business process lines to deal with the variability. In: Proceedings of the 2010 43rd Hawaii International Conference on System Sciences, HICSS'10. IEEE Computer Society, pp. 1–10.

Rombach, H.D., 2005. Integrated software process and product lines. In: International Software Process Workshop, SPW 2005, Beijing, China, pp. 83–90.

Ruiz, P., Quispe, A., Bastarrica, M.C., Hurtado Alegría, J.A., 2012. Formalizing the software process in small companies. Technical Report TR/DCC-2012-2, Computer Science Department, Universidad de Chile.

Schmidt, D.C., 2006. Model-driven engineering. IEEE Computer 39 (2), 25–31.

Simidchieva, B.I., Clarke, L.A., Osterweil, L.J., 2007. Representing process variation with a process family. In: Wang, Q., Pfahl, D., Raffo, D.M. (Eds.), International Conference on Software Process, ICSP'2007, LNCS 4470. Springer, Minneapolis, USA, pp. 109–120.

Simmonds, J., Bastarrica, M.C., 2011. Modeling variability in software process lines. Technical Report TR/DCC-2011-10, Universidad de Chile, Departamento de Ciencias de la Computacion.

Sutton, S.M., Osterweil, L.J.,1996. Product families and process families. In: ISPW '96: Proceedings of the 10th International Software Process Workshop. IEEE Computer Society, Washington, DC, USA, p. 109.

Tolvanen, J.P., Rossi, M., Liu, H.,1996. Method engineering: current research directions and implications for future research. In: Proceedings of the IFIP

TC8, WG8.1/8.2 Working Conference on Method Engineering: Principles of Method Construction and Tool Support. Chapman & Hall Ltd., London, UK, pp. 296–317.
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 EuroSPI, vol. 99. Communications in Computer and Information Science, Grenoble, France, pp. 142–152.
von Wangenheim, C.G., Weber, S., Hauck, J.C.R., Trentin, G., 2006a. Experiences on establishing software processes in small companies. Information and Software Technology Journal 48 (9), 890–900.
von Wangenheim, C., Varkoi, T., Salviano, C., 2006b. Standard based software process assessments in small companies. Journal of Software Process: Improvement and Practice 11 (3), 329–335.
Washizaki, H., 2006. Building software process line architectures from bottom up. In: Münch, J., Vierimaa, M. (Eds.), Product-Focused Software Process Improvement, LNCS 4034. Springer, Amsterdam, The Netherlands, pp. 415–421.
Weiss, D.M., Lai, C.T.R., 1999. Software Product-Line Engineering: A Family-Based Software Development Process. Addison-Wesley Professional.
Xu, P., 2005. Knowledge support in software process tailoring. In: Proceedings of the 38th Annual Hawaii International Conference on System Sciences, HICSS'05.
Zowghi, D., Firesmith, D., Henderson-Sellers, B., 2005. Using the OPEN process framework to produce a situation-specific requirements engineering method. In: Ralyté, J., Agerfalk, P.J., Kraiem, N. (Eds.), Proceedings of SREP05. Paris, France, pp. 59–74.

Julio Ariel Hurtado is an Associated Professor at the Systems Department, at the Universidad del Cauca. He is member of the IDIS group (Software Engineering Research and Development) since 2005. He received his Ph.D. in Computer Science from the Universidad de Chile in 2012 and a Bachelor in Electronic and Telecommunications Engineering from the Universidad Del Cauca in 1997. His main research topics are software engineering, software process models, software architecture, model-driven engineering, and software product lines. Lately, his work has focused on applying using MDE techniques and SPL approaches for designing and analyzing software process models.

María Cecilia Bastarrica is an Assistant Professor at the Computer Science Department, at the Universidad de Chile. She coordinates the MaTE group (Model and Transformation Engineering) since 2007. She received her Ph.D. in Computer Science and Engineering from the University of Connecticut in 2000, a Master of Science from the Pontificia Universidad Católica de Chile in 1994, and a Bachelor in Engineering from the Catholic University of Uruguay in 1991. Her main research topics are software engineering, software architecture, model-driven engineering, and software product lines. Lately, her work has focused on applying using MDE techniques for modeling software processes.

Jocelyn Simmonds is a Lecturer at the Departmento de Informatica at the Universidad Técnica Federico Santa María, Chile. She received her Ph.D. from the University of Toronto in 2011. Her main research area is software engineering, with specific interests in software testing, behaviour analysis, requirements engineering, and mobile and web development.

Sergio Ochoa is an Associate Professor of Computer Science Department at the Universidad de Chile. He received his Ph.D. in Engineering Science from Pontificia Universidad Católica de Chile in 2002 and a Bachelor in Software Systems Engineer from UNICEN, Argentina in 1996. His main research topics are Mobile/Ubiquitous/Social Computing (particularly context awareness, loosely-coupled work, positioning, ad hoc networking, systems modeling) and Software Engineering (software systems modeling, development in small software enterprises: requirements engineering, communication and coordination and software process tailoring). He is active member of the CARL and LACCIR networks; the SCCC conference, and the IEEE CS, ACM and CLEI journals.