Automatic Generation of Transformations for Software Process Tailoring

Luis Silvestre
Computer Science Department, University of Chile
Beauchef 851, Santiago - Chile
lsilvest@dcc.uchile.cl

***Abstract*—Tailoring software processes is an activity that allows process engineers to adapt organizational software processes to the needs of particular projects. Model-driven engineering (MDE) has been used for tailoring software processes using models and transformations. Even though there are some proposals for automatically generating part of the transformations, they are not easily applicable in the software industry because there are still factors that jeopardize its usage in small software enterprises. First, the potential users -process engineers- do not usually have the required knowledge for writing transformations. Second, current transformation languages and tools are not simple for defining and applying tailoring transformations. Trying to deal with these challenges, this research proposes a tool-set that balances the formality required by MDE and the usability needed by the users. We define a domain-specific language for defining tailoring rules. These rules are the input for a higher-order transformation that automatically generates tailoring transformations with no direct user interaction with the code. The tool-set reduces the complexity of defining tailoring rules and allows for the automatic generation of tailoring transformations. We illustrate the application of our approach in a small Chilean software company.**

I. Introduction

A software process is a structured set of activities required to develop a software system from traditional models such as the Waterfall [14], to more modern ones such as Scrum or XP [2]. Software process formal specification allows companies to rigorously document their development process and tool support for process analysis and evolution. Development projects faced by a particular company may be of different kinds, e.g., large or small, complex or simple, new development or evolution, and therefore the same software process is not equally appropriate for all of them.

Software process tailoring is the activity of adapting a general software process to match the needs of the project at hand. There are several proposals for software process tailoring, such as using the same process for addressing all projects, counting on a family of predefined processes or configuring a process by putting together appropriate pieces [18]. Empirical studies show that process tailoring is difficult because it involves intensive knowledge generation and application [4]. Therefore, the tailoring process is unrepeatable and difficult to evolve.

MDE [21] looks forward to improving development productivity and software quality by reducing the semantic gap between the problem domain and its solution, using models and transformations. We have worked with MDE-based tailoring for the last five years and it has proved to be technically feasible [9]. MDE-based tailoring consists of defining as a model the *organizational software process* along with its variability and the *project context*, and then use these models as input of a *tailoring transformation* whose output is the *project adapted process model* [12].

Motivation and Problem Statement.

Writing tailoring transformation requires not only knowledge about transformation programming languages, but also about the process model and the way its variability should be resolved according to the project context characteristics. These two kinds on knowledge –programming a model transformation and configuring a software process– are almost never mastered by the same person in the company, and it is even less frequent in Small Software Enterprises (SSEs), where the average expertise of the staff is not high [20]. Moreover, even if there is someone that counts on both kinds of knowledge, manually writing complex model transformation is still an error-prone activity.

MDE solutions are powerful and suitable for different application domains, but they are almost always complex and thus difficult to adopt. There are social and organizational factors that threaten MDE industrial adoption [10] (e.g., technological factors, resistance to change). In this sense, if SSEs would like to reuse their processes through automatic tailoring by defining and applying model transformations, they should not require a direct interaction with the source code of any transformation.

Related work. Proposals trying to address MDE solutions should balance the formality required by MDE and the usability needed by the process engineers. MOLA [11] and GREaT [1] allow specifying transformation rules through visual mapping patterns and VIATRA [28] provides a textual rule editor for specifying patterns. These proposals still need the process engineer to interact with metamodels, models, classes or code to adjust them to match the features of a specific project. Therefore, if the user does not count on the required experience for managing these elements, he could neither write nor maintain the transformation code.

Sijtema [23] proposes an extension to the Atlas Transformation Language (ATL) [19], which is based on feature models and requires the specification of the so-called variability rules. In the approach, variability rules are transformed to standard ATL code, i.e., variability is translated to *called rules* in ATL using a Higher-order Transformation (HOT). HOT [27] is a special kind of transformations that may take a transformation

as input and/or generate a transformation as output. Although the proposal raises the abstraction level, the process engineer still needs to understand the ATL extension and directly interact with code.

The Atlas Model Weaver (AMW) [6] tool includes an interactive interface for defining a weaving model that defines the relationships between two models. The weaving model can then be used as input for automatically generating model transformations. The purpose of AMW is to generate transformations for traceability or matching, so the rules are simple and they do not include complex structures; in particular it is not possible to include conditions for matching elements as we need for tailoring process models according to the project context. Nevertheless, we follow the structure of the AMW tool for our solution: defining the relationship between both input models and the output model, use this model as the input for a HOT and generate the ATL transformation.

There are some recent proposals such as Model Transformations By Example [30] and Model Transformation By Demonstration [26] that present innovative solutions for simplifying the implementation of model transformations using visual strategies and patterns. These strategies generate part of the code of the model transformations; however, it is still needed to complete such code. Therefore, this represents a semi-automatic approach to generate model transformations that is still not enough for process engineers.

Goal. This paper describes the automatic generation of tailoring transformations that improves the usability and hides the complexity of MDE components for final users in SSEs. We has two specific goals: (1) defining tailoring rules from a domain-specific language (DSL) and automatically generating tailoring transformations using a HOT, and (2) applying the generated tailoring transformation for obtaining project adapted software processes.

We illustrate the whole tool-set –the tailoring rule definition using the DSL and the tailoring transformation execution– by applying it to the process of an industrial partner. We show how this tool-set is able to generate tailoring transformations that are correct by construction, in a usable manner, requiring from the process engineer knowledge about the process tailoring, but hiding the complexity of managing models and writing model transformations.

II. Running Example

Mobius is a small Chilean company that was created four years ago. It develops integrated software and hardware for the public transportation system in Santiago, Chile. Mobius has 20 employees; 8 are directly working in software maintenance and development. Smaller projects typically last a couple of days, while larger development projects are three to four months long. The software development process is loosely based on Rational Unified Process [13] and is quite detailed in its definition, with 104 tasks, 10 roles and 44 work products. They formalized their software process three years ago using Eclipse Process Framework Composer (EPFC)[1]. EPFC is a software process modeling tool that supports Software and Systems Process Engineering Metamodel Specification (SPEM) [16] for modeling software process and UML activity diagrams for representing software process behavior.

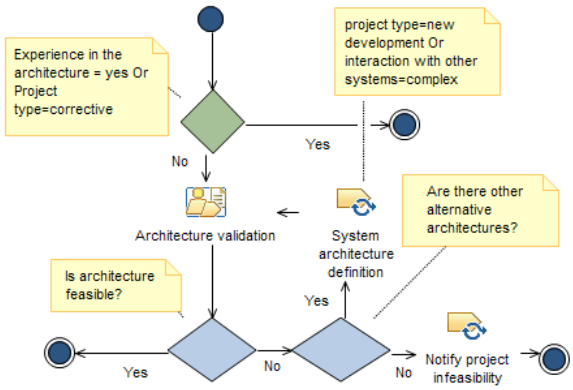

Fig. 1. Mobius's Architecture validation activity

Figure 1 shows the *Architecture validation* activity that is part of Mobius's process. Note that variability is represented in the figure as a set of annotations. Tasks with annotations denote optionality while dark decisions with annotations denote alternatives; however variability is formally specified using SPEM variability primitives in EPFC. Mobius's process has 16 variability points.

Mobius is part of GEMS[2] research project. Mobius with help of the GEMS team, specified a organizational software process model, organizational context model and decision rules for tailoring software process. We will show how the proposed tool-set is applied for automatically generating the tailoring transformation and executes the software process tailoring. For this purpose, we will illustrate the approach using the *System architecture definition* task (see Fig. 1).

III. General Approach

The approach requires two input models: an *organizational software process model* (component 2 in Fig. 2) that conforms to the eSPEM (experimental SPEM) metamodel that is a subset of SPEM, and a *project context* (component 7 in Fig. 2) that is a configuration of the *organizational context model* (component 1 in Fig. 2) that conforms to the Software Process Context Metamodel (SPCM). This approach uses a *tailoring transformation* (component 7 in Fig. 2) that is a model-to-model transformation to generate an *adapted software process model* (component 8 in Fig. 2) as output. The resulting process model also conforms to eSPEM but includes no variability. The proposed solution is called *A Tool-set for Automatically Generating Tailoring Transformations* (ATAGeTT). ATAGeTT is a model-based tool that allows process engineers to interactive and transparently define the tailoring rules and generate the tailoring transformation that can be used to adapt the

[1]www.eclipse.org/epf/

[2]www.dcc.uchile.cl/gems

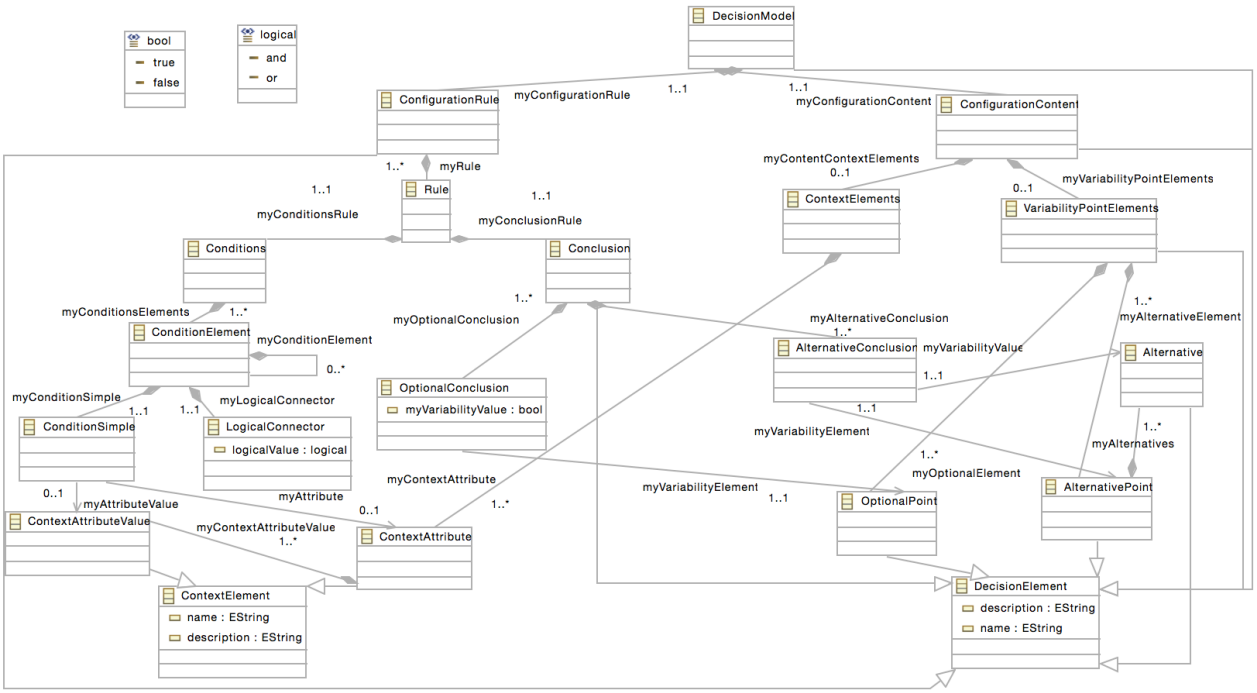

Fig. 3. Variation decision metamodel for transformation rules definition

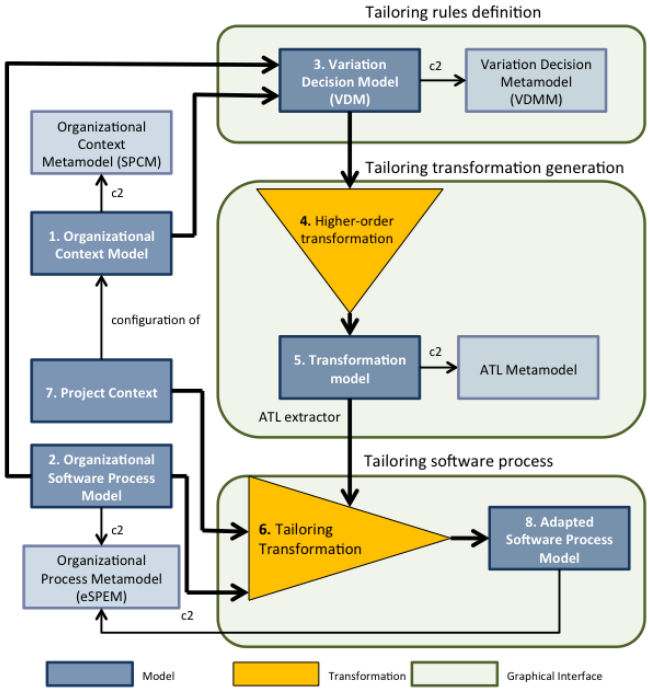

Fig. 2. Architecture of the proposed solution

organizational software process. Figure 2 shows the general architecture of the ATAGeTT.

A. Tailoring Rules Definition

Tailoring rules definition requires two input models.

The first input is the *organizational context model* (component 1 in Fig. 2). This model describes the potential values that context attributes may take for particular projects. The second input is the *organizational software process model* (component 2 in Fig. 2) defined by the SSE for guiding software development. This model specifies the potential variability of activities, roles and work products that are optional or have alternative implementations.

Tailoring rules definition involved two steps: 1) identifying the domain-specific concepts involved in tailoring rule definition and 2) developing the DSL as a language that allows defining tailoring rules. Next we explain these activities.

1) Conceptualizing the Process Tailoring Rules. We define process tailoring rules as a set of conditions, conclusions and logical operators that decide an action about a variable process element. Consequently, a decision model is a set of rules of the form $condition \Rightarrow conclusion$, where a condition is a predicate about the project context, and the conclusion indicates how a variation point is resolved when the condition holds. We define the *Variation Decision Model* (VDM) (component 3 in Fig. 2) for formally representing tailoring rules that conforms to Variation Decision Metamodel (VDMM) (see Fig. 3). The VDM is inspired by decision models [29] and Semantics of Business Vocabulary and Business Rules used for building decision rules [15].
2) Rule Definition. By keeping in mind the goal of reducing the skills required to define tailoring transformation rules, we developed a tool that allows the process engineer to create models through a graphical interface

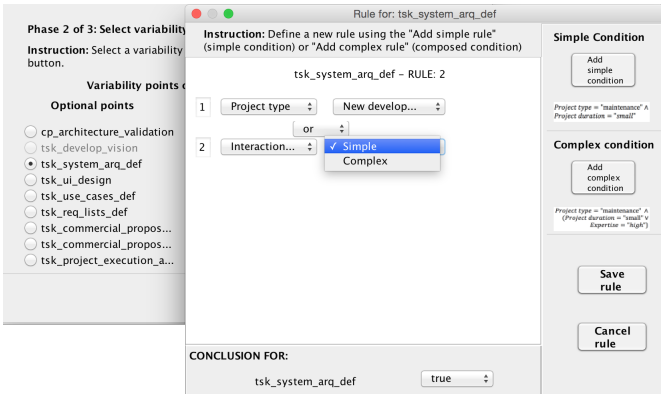

Fig. 4. Graphical interface for tailoring rules definition

(Fig. 4). This graphical interface allows defining the tailoring rules for each variation point of the software process to be tailored without interacting with the code. The graphical interface includes three activities: the selection of input models, the definition of tailoring rules and the automatic generation of the VDM. Concerning the first one, the tool allows the user to select the input models –organizational software process and organizational context model– in a simple way. Concerning the second activity, the tool guides the user presenting only the software process elements defined as variable and allowing to specify a rule either simple or complex for each variation point (optional or alternative). The tool also counts on a menu where the context attributes and their potential values can be selected not requiring any typing. Concerning the third activity, the tool automatically generates the VDM using a text-to-model transformation. This VDM is a DSL specifically defined for modeling tailoring rules (see Fig. 5).

In the running example (Fig. 4), the process engineer defines that the *System architecture definition* task (*tsk_system_arq_def* in the interface) should not be included when the *Project type* is "New Development", or when the *Interaction with other systems* is "Complex". These decisions are part of the adaptations defined by Mobius for its organizational software process (see Fig. 1). Figure 5 shows the VDM generated through the graphical interface. We can see that the Configuration Content is formed by the Context Elements and the process variability point elements. On the other hand, in the Configuration Rule we can see that *rule2* is highlighted and in the lower part we can see that the *Project type* attribute has been assigned the "New Development" value. As part of this rule conclusion, *System architecture definition* task (*tsk_system_arq_def* in the model) is set to *True*, i.e., it will be included as part of the adapted process; if the condition does not hold, the activity will not be included.

B. Tailoring Transformation Generation

The VDM is used as input for a HOT that automatically generates the process tailoring transformation. The tailoring

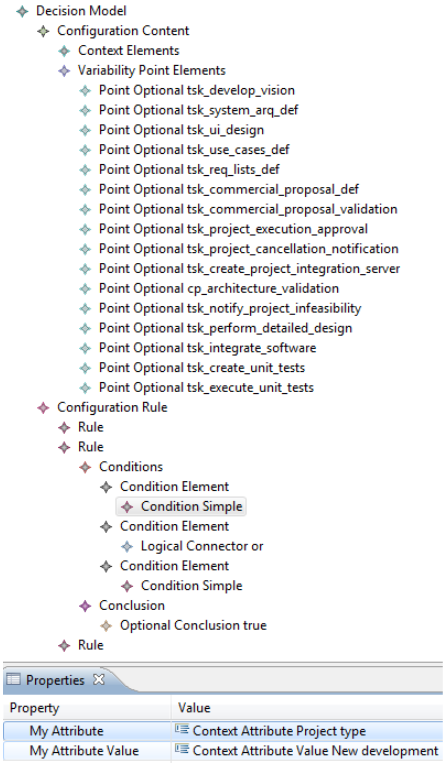

Fig. 5. Variation decision model for Mobius's process

transformation is generated only once, and it can be used for tailoring different projects' contexts.

Tailoring transformation generation has two steps: 1) generate the tailoring transformation model and 2) extract the tailoring transformation code.

1) The HOT (component 4 in Fig. 2) is a generic transformation and has the VDM as input. The HOT implements a kind of transformation called synthesis pattern for generating the tailoring transformation model using a model-to-model transformation. This HOT is an exogenous transformation because its input and output models are expressed using different languages (decision and transformation models, respectively). The resulting tailoring transformation model (component 5 in Fig. 2) conforms to the ATL metamodel.
2) The tailoring transformation code is obtained through a model-to-text transformation that takes the tailoring transformation model as input. This is a vertical transformation –code generation– because its source and target models are at different levels of abstraction (model representation and code generation).

Figure 6 shows the tailoring transformation automatically generated for tailoring Mobius's process. As stated in Fig. 4, *System architecture definition* (*tsk_system_arq_def* in the code) is optional, and it has an associated rule in the tailoring transformation. Figure 6 highlights the helper called by *rule2* for deciding about the inclusion of the *System architecture definition* task in the adapted process.

```
--Optional points
helper def: optionalRule(name:String): Boolean =
if(Sequence{'tsk_develop_vision',
   'tsk_system_arq_def'})
then(
   if('tsk_system_arq_def'= name) then
     thisModule.rule2()
   else (
      if('tsk_develop_vision'= name) then
        thisModule.rule1()
      else true
      endif
   endif)
else false
endif;
-- Rule 1 for tsk_develop_vision
-- ...
-- Rule 2 for tsk_system_arq_def
helper def:rule2():Boolean=
   if(thisModule.getValue('Project type') =
       'New development' or
       thisModule.getValue('Interaction with other
           systems') = 'Complex')
       then true
       else false
   endif;
```

Fig. 6. Generated transformation code

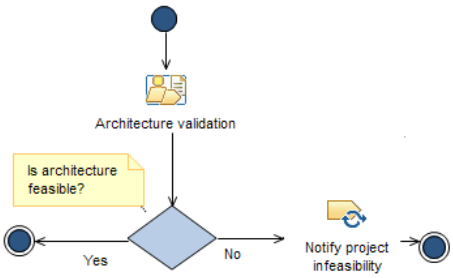

Fig. 7. Generated adapted software process

C. Tailoring Software Process

The *adapted software process model* (component 8 in Fig. 2) is the output of applying the generated transformation using the organizational software process (component 2 in Fig. 2) and the project context (component 7 in Fig. 2) as input. Such a process includes all the process elements that are needed for the specified context and nothing else.

We developed a tool for applying the tailoring software process in a usable way. The graphical interface allows the user to select the project context and automatically generates the adapted software process. The user does not need to interact with the tailoring transformation execution.

Mobius has a *maintenance project* and the project context has the following configuration: *Project Type* is "Maintenance", and *Interaction with other systems* is "Simple". Figure 7 shows the generated adapted software process using ATAGeTT and we can see that the *System architecture definition* task was not included because the result of *rule2* in Fig. 6 is False.

IV. Results and Contributions

The paper presents a tool-set for automatically generating model transformations for tailoring software processes, reducing thus the skills required by process engineers to perform this task. The proposal makes an extensive use of MDE technology and encapsulates highly sophisticated concepts within the tool-set that allows to address a practical industrial problem.

We tried using some of the most promising techniques and tools for developing HOTs such as AMW and MOFScript but we found limitations for dealing with two input models [24]. We built a proof of concept of ATAGeTT and applied it through a running example [25]. We automatically generated transformation rules using the graphical interface and the VDM. We also generated tailoring transformations for various other Chilean SSEs using the generic HOT.

We applied an exploratory case study [31] for evaluating *correctness* and *productivity* in two small Chilean SSEs. We analyzed three software projects for each small Chilean SSE. We compared the productivity of adapted software processes obtained by using two different approaches: template-based tailoring [5] and automatic tailoring [25]. The productivity was measured in terms of the missing and extra tasks between both tailoring approaches. We found that the automatic tailoring is always more efficient for generating the adapted process [8]. The correctness was evaluated by comparing the adapted software processes obtained manually and automatically.

We are currently applying a confirmatory case study [31] for evaluating *usability* and *expressiveness* in two Chilean SSEs. We are evaluating the usability by applying structured questionnaires to process engineers [3] about their perception about the graphical interface usability focusing on the elements of Quality in Use Integrated Measurement (QUIM) [17]. We are evaluating the expressiveness by verifying that it counts on all the required constructs for expressing the software process engineer's intentions for tailoring software processes. Usability and expressiveness will be analyzed using qualitative methods [22], while correctness and productivity evaluation use quantitative methods [7].

From our practical experience with our industrial partners, we have validated that the proposed tool-set is easy to learn and use for process engineers. Also, the generated tailoring transformations yield the expected tailored software processes, raising our confidence on the correctness on the tool-set. We have not found in practice any tailoring rule that could not be specified using the VDM, also raising our confidence about its expressiveness for this application domain. However, more practical experimentation is required to have conclusive evidence.

So far, the results are promising. Nevertheless, we are aware that process and tailoring strategies of larger software companies may be more complex, and thus the applicability of our tool-set is not guaranteed. For instance, it may be the case that some conditions for resolving variability do not just depend on context values, but also on the way other variation points are resolved; the tool-set does not support this kind of conditions yet.

Acknowledgments: This work has been partly funded by Project Fondef GEMS IT13I20010 and Luis Silvestre was also supported by PhD Scholarship Program of Conicyt, Chile (CONICYT-PCHA/2013-63130130).

REFERENCES

[1] D. Balasubramanian, A. Narayanan, C. vanBuskirk, and G. Karsai. The graph rewriting and transformation language: GReAT. *Electronic Communications of the EASST*, 1, 2007.

[2] K. Beck, M. Beedle, A. Van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, et al. Manifesto for agile software development. 2001.

[3] M. A. Campion, J. E. Campion, and J. P. Hudson. Structured Interviewing: A note on Incremental Validity and Alternative Questions Types. *Journal of Applied Psychology*, 79(6):998 1002, 1994.

[4] P. Clarke and R. V. OConnor. The situational factors that affect the software development process: Towards a comprehensive reference framework. *Information and Software Technology*, 54(5):433–447, 2012.

[5] A. Cockburn. *Crystal Clear a Human-powered Methodology for Small Teams*. Addison-Wesley Professional, first edition, 2004.

[6] M. D. Del Fabro, J. Bézivin, and P. Valduriez. Weaving Models with the Eclipse AMW plugin. In *Eclipse Modeling Symposium*, volume Eclipse Summit Europe 2006, Esslingen, Germany, 2006.

[7] T. Dybå, V. B. Kampenes, and D. I. K. Sjøberg. A systematic review of statistical power in software engineering experiments. *Information & Software Technology*, 48(8):745–755, 2006.

[8] F. Gonzalez, L. Silvestre, M. Solari, and M. C. Bastarrica. Template-Based vs. Automatic Process Tailoring. In *XXXIII International Conference of the Chilean Computer Science Society SCCC 2014*, Talca,Chile, November 2014.

[9] J. A. Hurtado Alegria, M. C. Bastarrica, A. Quispe, and S. F. Ochoa. MDE-based process tailoring strategy. *Journal of Software: Evolution and Process*, 26(4):386–403, 2014.

[10] J. Hutchinson, J. Whittle, and M. Rouncefield. Model-driven engineering practices in industry: Social, organizational and managerial factors that lead to success or failure. *Science of Computer Programming*, 89:144–161, 2014.

[11] A. Kalnins, J. Barzdins, and E. Celms. Model Transformation Language MOLA. In U. Aßmann, M. Aksit, and A. Rensink, editors, *MDAFA*, volume 3599 of *Lecture Notes in Computer Science*, pages 62–76, Twente, The Netherlands, 2004. Springer.

[12] G. Kalus and M. Kuhrmann. Criteria for Software Process Tailoring: A Systematic Review. In *Proceedings of the 2013 International Conference on Software and System Process*, ICSSP 2013, pages 171–180, New York, NY, USA, 2013. ACM press.

[13] P. Kruchten. *The Rational Unified Process: An Introduction*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3 edition, 2003.

[14] J. Lonchamp. A Structured Conceptual and Terminological Framework for Software Process Engineering. In *In Proceedings of the Second International Conference on the Software Process*, pages 41–53. IEEE Computer Society Press, 1993.

[15] OMG. Semantics of Business Vocabulary and Business Rules (SBVR) Version 1.0. Technical Report 2008-04-01, 2008. http://www.omg.org/spec/SBVR/1.0/.

[16] OMG. Software & Systems Process Engineering Metamodel Specification (SPEM) Version 2.0. Technical Report 2008-04-01, Object Management Group, 2008. http://www.omg.org/spec/SPEM/2.0/PDF/.

[17] H. Padda. *QUIM: A Model for Usability/Quality in Use Measurement*. LAP Lambert Academic Publishing, Germany, 2009.

[18] O. Pedreira, M. Piattini, M. R. Luaces, and N. R. Brisaboa. A Systematic Review of Software Process Tailoring. *SIGSOFT Softw. Eng. Notes*, 32(3):1–6, May 2007.

[19] A. E. Project. Atlas Transformation Language. Technical report, ATL Eclipse Project, January 2006. http://www.eclipse.org/atl/.

[20] A. Quispe, M. Marques, L. Silvestre, S. F. Ochoa, and R. Robbes. Requirements engineering practices in very small software enterprises: A diagnostic study. In S. F. Ochoa, F. Meza, D. Mery, and C. Cubillos, editors, *SCCC 2010, Proceedings of the XXIX International Conference of the Chilean Computer Science Society, Antofagasta, Chile, 15-19 November 2010*, pages 81–87. IEEE Computer Society, 2010.

[21] D. C. Schmidt. Guest Editor's Introduction: Model-Driven Engineering. *IEEE Computer*, 39(2):25–31, 2006.

[22] C. B. Seaman. Qualitative Methods in Empirical Studies of Software Engineering. *IEEE Trans. Software Eng.*, 25(4):557–572, 1999.

[23] M. Sijtema. Introducing Variability Rules in ATL for Managing Variability in MDE-based Product Lines. volume 10, pages 39–49. CEUR Workshop Proceedings, 2010.

[24] L. Silvestre, M. C. Bastarrica, and S. F. Ochoa. Implementing HOTs that Generate Transformations with Two Input Models. In *XXXII International Conference of the Chilean Computer Science Society SCCC 2013*, Temuco, Chile, November 2013.

[25] L. Silvestre, M. C. Bastarrica, and S. F. Ochoa. A Model-based Tool for Generating Software Process Model Tailoring Transformations. In L. F. Pires, S. Hammoudi, and J. Filipe, editors, *MODELSWARD*, pages 533–540. SciTePress, 2014.

[26] Y. Sun, J. White, and J. Gray. Model Transformation by Demonstration. In A. Schürr and B. Selic, editors, *MoDELS*, volume 5795 of *Lecture Notes in Computer Science*, pages 712–726, Denver, CO, USA,, 2009. Springer.

[27] M. Tisi, F. Jouault, P. Fraternali, S. Ceri, and J. Bézivin. On the Use of Higher-Order Model Transformations. In R. F. Paige, A. Hartman, and A. Rensink, editors, *ECMDA-FA*, volume 5562 of *Lecture Notes in Computer Science*, pages 18–33, Enschede, The Netherlands, 2009. Springer.

[28] D. Varró. Model transformation by example. In O. Nierstrasz, J. Whittle, D. Harel, and G. Reggio, editors, *Model Driven Engineering Languages and Systems*, volume 4199 of *Lecture Notes in Computer Science*, pages 410–424. Springer Berlin Heidelberg, 2006.

[29] D. Weiss, J. Li, H. Slye, T. Dinh-Trong, and S. Hongyu. Decision-Model-Based Code Generation for SPLE. In *SPLC*, pages 129–138, Sept 2008.

[30] M. Wimmer, M. Strommer, H. Kargl, and G. Kramler. Towards Model Transformation Generation By-Example. In *HICSS*, page 285, Big Island, HI, USA, 2007.

[31] R. Yin. *Case study research: design and methods*. Sage Publications, Newbury Park, CA, third edition, 2002.