Toward Lean Development in Formally Specified Software Processes

María Cecilia Bastarrica[1], Julio Ariel Hurtado Alegría[1,2], and Alexandre Bergel[1]

[1] Computer Science Department, Universidad de Chile
Blanco Encalada 2120, Santiago, Chile
[2] IDIS Research Group, University of Cauca
Street 5 # 4–70, Popayán, Colombia
{cecilia,jhurtado,abergel}@dcc.uchile.cl

Abstract

Formally specifying the software development process has been the way followed by several companies for making development more predictable. However this formality has frequently introduced bureaucracy into the process. Lean software development is an agile practice that promotes developing only those work products that are required, i.e., no waste should be included in the process. In this paper we present an automatic means of detecting and localizing the presence of certain type of waste in software processes that are formally specified using SPEM 2.0. We show our findings by analyzing the Scrum process model and the software development process model of a medium size software development company in Chile.

Keywords

Lean software development, software process improvement, software process model analysis

1 Introduction

Lean software development is an agile practice that promotes quality and productivity by focusing on core issues and not investing effort executing non essential tasks and building non required work products [16] . Software companies tend to define and formalize their development processes in an effort to make them more predictable. This formalization is a hard task and implies a huge investment, so it is natural to try to get the highest return of investment out of it. Trying to cover all possible cases, it is not rare to introduce unnecessary tasks and work products as part of the formalized process, and this may build up waste into the process. It is not easy to identify the existence of waste, and even harder to localize it within the process. A typical case of waste is developing work products that nobody needs, i.e., that are neither deliverables nor required for executing any task within the process. We will focus on this kind of waste.

We have developed *AVISPA*, a tool for visual analysis of software processes. It is able to identify a series of error patterns that we have found to be frequent in practice [5] . In this paper we extend *AVISPA* so that it is able to also identify certain type of waste –useless work products – in software processes formally specified in Eclipse Process Framework.

We apply the extended tool in two quite diverse scenarios: the Scrum process specification publicly available from the EPF Community, and the software development process of a medium size software company in Chile. In the former case, no waste of the type we are looking for (useless work products) has been found as expected, provided that Scrum is an agile method. In the latter case, we were able to identify and localize several waste elements, and all of them are opportunities for software process improvement.

The rest of the paper is organized as follows. In Section 2 a precise statement of the problem being addressed is detailed. The *AVISPA* tool and the mechanics of waste detection are described in Section 3. Section 4 shows the application of our tool for localizing waste in the two aforementioned software process models. Related work is discussed in Section 5. Finally, some conclusions and future work are described in Section 6.

2 Problem Statement: Localizing Waste in Formal Processes

Lean software development implies the application of seven principles [15] : eliminate waste, build quality in, create knowledge, defer commitment, deliver fast, respect people, and optimize the whole. One of the most important of these principles is eliminating waste. But it is not necessarily clear the form that waste may take within a software process, and even less clear how it could be identified, let alone automatically localized. In this paper we will not necessarily focus on agile processes. Nevertheless, eliminating waste not only applies to this kind of processes; it may even be more relevant in the context of non agile processes as will be apparent from the experimentation presented in Section 4.2.

Formal processes unambiguously specify who does what and when, and which work products are built/modified as a result. SPEM 2.0 [13] is the OMG standard notation for specifying software processes, and the Eclipse Process Framework1 is a platform that allows the specification of SPEM 2.0 processes. In a lean software development process there should be no waste, therefore any work product should be either a deliverable or an input for some other task within the process itself, and it should be specified accordingly. SPEM 2.0 provides primitives for specifying that a work product is a deliverable. Therefore, if we find a work product that is neither specified as a deliverable nor as an input for any other task, then we are in one of the following scenarios, all of them problematic:

1. the work product is actually necessary for performing some task, but we have forgotten to specify it, so we have found an underspecification;

[1] EPF: http://www.eclipse.org/epf/

2. the work product is actually a deliverable, but it was not defined as so, so we have found another form of underspecification;
3. the work product is not really necessary, so we have found waste in the process: a useless work product.

3 Automatic Waste Detection

This section describes the whole procedure for automatically localizing the useless work product kind of waste in formal processes specified in EPF. First, we describe *AVISPA*, a tool for analyzing formal software processes. Afterwards, we describe the extension we propose to add to *AVISPA* so that it could also be applied for localizing waste.

3.1 Software Process Blueprints and AVISPA

Software processes may be composed by several hundreds of elements of diverse kinds. This issue makes sometimes difficult to analyze the quality of a process just through inspections. In [6] , we have proposed Software Process Blueprints that are partial views of the software process that allow the process engineer to visually analyze its quality. Each of these views focuses on one essential SPEM modeling element: role, task and work product, and thus we have a Role Blueprint, a Task Bluprint and a Work Product Blueprint. All of them are graphs formed by nodes and edges/arcs.

In the Role Blueprint nodes represent roles whose size corresponds to the number of tasks in which the role is involved. Also, an edge between two nodes represents the existence of collaboration between the two roles to perform a task. Therefore nodes that are too big may reveal overloaded roles, and disconnected nodes show roles that do not collaborate.

In the Task Blueprint nodes represent tasks, whose height is the number of input work products and whose width is the number of output work products for that task. An arc from one task to another, represents precedence, i.e., an output work product of the former task is an input work product for the latter task. In this way, very wide nodes suggest tasks without a clear goal, i.e., whose purpose is to produce a variety of work products.

In the Work Product Blueprint nodes are work products, where their height represents the number of tasks that require the work product as an input, and their width is the number of tasks that create/modify it. In this case a node that is too high reveals that certain work product is required by several tasks, and it therefore may become a bottleneck. Figure 1 depicts the Work Product Blueprint of the software process of a medium size company; there we can clearly see that the node corresponding to System Requirements Specification is much higher than most of the others suggesting that many tasks require it.

Also the existence of disconnected subgraphs in any blueprint reveals a misspecification in the process. However, in Process Model Blueprints, nodes that are much larger than others could suggest anomalies, but it is the responsibility of the process engineer to determine if they are actual errors, improvement opportunities, or if on the contrary they are defined that way on purpose. Moreover, it is not clear how big could be considered too big.

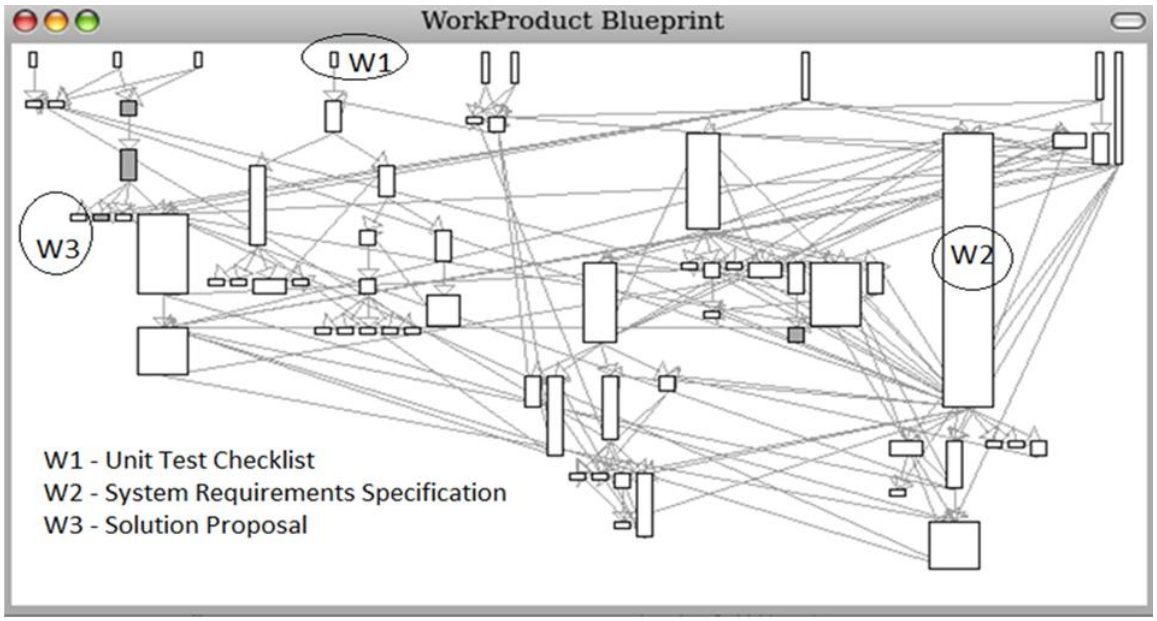

Figure 1: Work Product Blueprint of a Chilean software company

In [5] , we proposed *AVISPA*, a tool based on blueprints that automatically identifies and localizes a series of error patterns[2]. Table 1 describes those error patterns already identified. In *AVISPA* error patterns are highlighted in color, so that it is evident when there is a possible error. For this purpose some assumptions have been made such as defining that an element that is more than one standard deviation larger than the mean is considered too big and suggests the presence on an error. This assumption has worked fine in practice so far. For example, in Figure 2 we show the Work Product Blueprint for the Scrum process; *AVISPA* has highlighted the Project Backlog as a work product that is too demanded, and as such it may be a bottleneck in the process as a whole. A thorough analysis of the Scrum process using *AVISPA* can be found in [4] .

Table 1: Error patterns identified by *AVISPA*

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

[2] AVISPA (Analysis and Visualization for Software Process Assessment): http://www.moosetechnology.org/tools/ProcessModel. AVISPA is freely available under the MIT license.

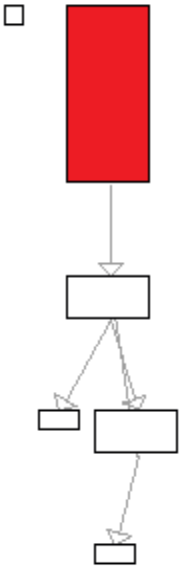

Figure 2: Identifying work products that are too demanded

3.2 Localizing Waste in Work Product Blueprints

Deliverables are those work products that need to be delivered to the customer as part of the final product. For example, a user requirements document and the source code are typical deliverable work products. In SPEM 2.0, some work products can be defined as deliverables so they could be easily identified.

As part of the software development process, not only deliverable work products are produced. There are other intermediate work products that are needed mainly for coordinating successive tasks probably performed by different people. For example, the test set is an output work product of the Design Test Set task and an input of the Execute Test Set task, but it is not necessarily a deliverable work product. However, if there are work products that are neither deliverables nor input for any other task within the process, they are a kind of waste we do not want to develop if we intend to have a lean process.

In the Work Product Blueprint, an arc connecting nodes represents precedence between work products. If there is a WP_a that precedes WP_b in the graph, that means that there is a task such that WP_a is its input and WP_b is its output. In this way, all leaves in the graph, i.e., nodes with no successor, should represent deliverable work products. In this paper *AVISPA* is extended so that it highlights in blue all those leaves that are not defined as deliverables. The process engineer then needs to analyze all highlighted nodes so that he/she could determine if each of the highlighted work products is actually required as an input of another task, and thus it is not a leaf, if it should have been defined as a deliverable and thus it should not have been highlighted, or if it is actually waste in the process and it is an improvement opportunity.

4 Application to Two Diverse Processes

In this section we apply the extended *AVISPA* for localizing waste in two software development processes. We apply our tool in two dramatically different scenarios. First we focus on the publicly available Scrum process model specification that can be found at the EPF Community web site[3]. A priori, Scrum, being an agile method, is expected to show no waste in its specification. Then we will proceed to analyze the software development process of a Chilean medium size software company. In this latter case we will see that looking for waste in real world software processes is not only much harder,

[3] Scrum: http://www.eclipse.org/epf/downloads/scrum/scrum downloads.php

but much more useful when identified provided that the size and complexity of the process model makes it almost impossible to analyze it manually.

4.1 Scrum

Scrum is an agile process used to rapidly develop software. It has been defined by Jeff Sutherland and more formally elaborated by Ken Schwaber [17] . Scrum stresses management values and practices, and it does not include practices for technical parts (requirements, design, and implementation); this is why it is usually used in combination with another agile method such as Extreme Programming.

The application of Scrum enforces a few simple rules that have the potential to make a team self-organize into a process that can achieve 5 to 10 times the productivity of a waterfall-based process. However, most Scrum teams never achieve this goal [18] . According to Sutherland, teams face difficulties to organize work in order to deliver working software at the end of each sprint. Moreover, they also experience trouble working with a Product Owner to get the backlog in a ready state before bringing it into a sprint. Also, organizing into a hyper-productive state during a sprint remains a challenging issue. Our findings analyzing the Scrum process model with *AVISPA* [5] are consistent with these ideas.

We claim that it may be the case that the publicly available Scrum process model may be misspecified, and thus people adopting it as it is may be using an inherently suboptimal process. We apply the extended *AVISPA* to the EPF community Scrum process model in order to look for waste and/or the other kinds of misspecification detailed in Section 2, from the point of view of the work products. Figure 3 shows the results.

The Potentially Shippable Product Increment (A) has been highlighted. This work product needs to be an input to the integration task, but the public Scrum process model does not specify this fact, so (A) is an underspecification. The Release Burndown Chart (B) and Sprint Burndown Chart (C) are clearly necessary for executing the development tasks, but the model does not specify these dependencies either. They are also underspecifications. Therefore, the extended *AVISPA* is able to identify this kind of underspecifications even in a very small software process. But also from this analysis, we can see that no false positives are identified: all highlighted elements correspond to errors in the process model specification. Moreover, analyzing each highlighted element, we can confirm that there is no waste of the kind useless work product in the Scrum process model, as expected for an agile method.

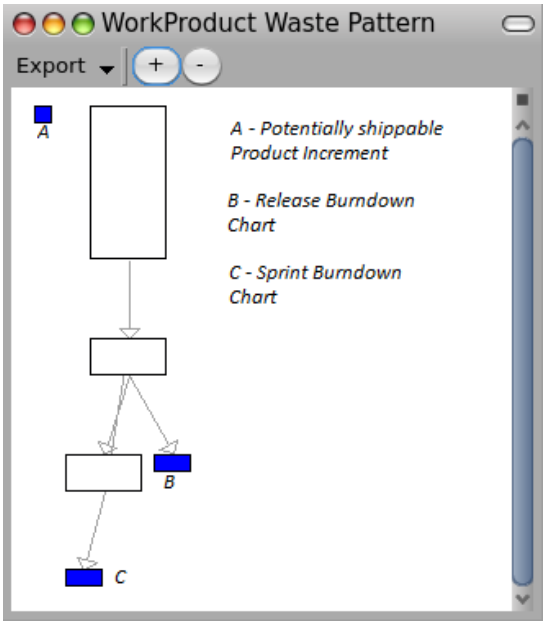

Figure 3: Work products that are potential waste in Scrum

4.2 Development Process of a Software Development Company

DTS is a Chilean software company that works in solutions for military and civil technology. It has around 250 employees, including engineers, certified technicians, operation workers and managers. Particularly, the Self-Service Systems Engineering Area in DTS (SSSEA-DTS) started to define its software process model in 2008, using the Rational Unified Process as a reference. In SSSEA-DTS software process improvement has been oriented toward recovering the software process currently applied in the organization, in order to formalize it, analyze it, and improve it if found necessary. SSSEA-DTS's process model is composed by 66 work products, 9 roles and 57 tasks. This model has been defined with a total effort of 0.5 person-months during 12 months.

Figure 4 shows the Work Product Blueprint in which blue nodes (dark colour on a black and white printout) identify potential waste work products or underspecifications. The tool highlights 22 problematic work products (33% of the whole), which include non-defined deliverables, underspecified task inputs and useless work products, i.e., actual waste.

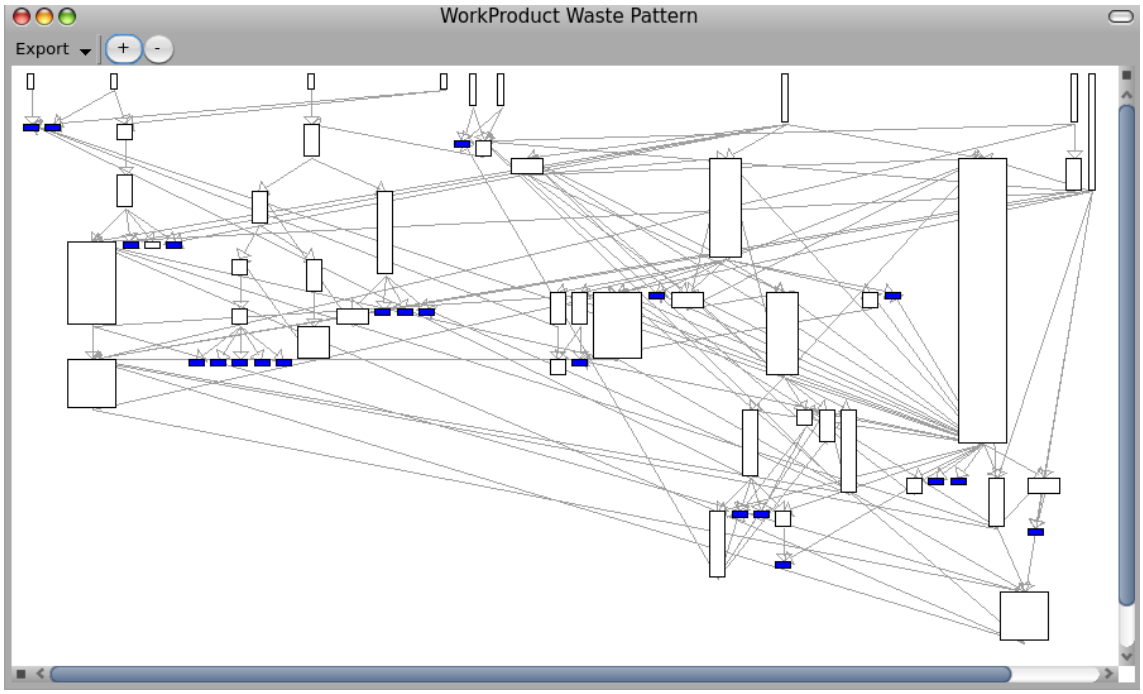

Figure 4: Work products that are potential waste in DTS

In order to validate our findings and assess the relevance of the waste work products, we confronted our blueprint with the SSSEA-DTS process engineer. From the 22 highlighted work products found, 3 have been confirmed to be non specified deliverables, 16 underspecified task inputs, and 3 are indeed waste work products.

In particular, these 3 work products correspond to evidence required for bureaucratic issues, i.e., evidence about the approval of other work products. For example, the Requirements Approval work product is the evidence of requirements acceptance by different stakeholders, with respect of the Requirements work product. This evidence could have been registered in each respective work product (for example as a field) instead of a new work product. But, according to the expert's opinion, and based on the implementation of software processes during the past five years, about 12 of the underspecified work products could have also been integrated as part of other work products, reducing bureaucratic work significantly. For example, activity specifications, operational state specifications, communication protocol specifications, and requirements observations from stakeholders can be all considered part of the system requirements work product, decreasing the effort to design, maintain, control and configure work documents. As a summary, the practical effectiveness of the tool has been

confirmed with DTS by identifying 22 problematic work products: 13.6% percent of deliverables that had not been identified, 31.9% underspecified tasks (where the work product should have been specified as a task input), 13.6% waste and 40.9% could be improved (refactoring and integration).

The actual findings not only allowed the process engineer in DTS to improve their software process specification, but also allowed them to gain more confidence about the quality of their process.

5 Related Work

Software process improvement through the Lean Measurement (SPI-LEAM) method is proposed by Petersen et al [14] . This method describes a way to implement lean principles through measurement in order to initiate software process improvement. The method uses collected data from projects executed to evaluate performance and quality aspects, particularly identifying causes of waste. Mujtaba et al [11] propose a case study to identify waste in a software process by using value stream maps (VSM). The empirical data is collected using document analysis, extraction of phase times from a requirement tracking tool and interviews. It is used to construct a value stream map that shows the present state of the process. Static validation showed that the VSM methodology is useful for identifying waste and to propose measures to avoid it. Middleton et al. [9] developed a study case where a company is followed using lean practices for two years. One of their most relevant findings was that the company had many non-value tasks. Data collected at the company showed an increment of 25% in productivity, schedule delay was reduced to 4 weeks from several months or years, and the time for defect fixing was reduced by 65% - 80%. In our approach, the waste error pattern is applied on the software development model, allowing part of the waste to be identified before the process model is actually enacted, whereas the previous approaches are based on data collected from the process applied in specific projects. So, the waste error pattern could be complementarily used with SPI-LEAM or VSM methodologies before a new process model is tested, as a form of static verification mechanism.

Visualization is regularly employed to identify deficiencies and errors in application source code. Polymetric views is a lightweight visualization representation, originally designed to analyze software source code. Polymetric views were first employed for reverse engineering [8] , code comprehension [1] and characterization purposes [2] . Even though the application range of polymetric views has greatly expanded over the last few years [3] [10] , all these views make use of pattern recognition to visually identify abnormal situations. The blueprints and error patterns applied in this paper are no exceptions. However, as far as we are aware of, our work is the first usage of polymetric views to identify anomalies in software processes.

Knab et al. [7] proposed a set of generic visual process patterns. With these patterns, the authors analyze effort estimation, length, and sequences of problem resolution activities. Based on the information obtained from issue tracking databases, the visual representation of a problem is classified as overestimated, underestimated and perfectly estimated. Our blueprints have a focus different from effort estimation. Instead of estimating the result of an effort already realized, we provide an indication and recommendation about how to prevent waste.

6 Conclusions

We have proposed an extension to *AVISPA* so that it is able to localize potential waste in software process models specified with EPF. We focus on the kind of waste represented by work products that are developed although they are neither necessary nor useful. These elements are all those leaves in a Work Product Blueprint that are not marked as deliverables. However, we have found that most elements that satisfy these conditions are due to incompleteness in the specification: they are either deliverables that are not defined as so, or they are not leaves because they should have been specified as input of a task within the process. In both cases colored elements highlight errors in the process specification, and thus they are also improvement opportunities.

Our method has proved to be effective in both scenarios applied and presented in the paper. In the case of Scrum, we corroborated that the process model has no waste in the form of useless work

products as expected from an agile method, but it was still useful for identifying misspecifications in the publicly available process model. In the case of the process of the medium size software company, the tool resulted highly useful for identifying all kinds of possible errors. In this case we were able to identify actual waste. This fact has been validated with the process engineer at the company and they agreed on the recommendations. They are currently in the process of restructuring their development model taking our findings into account.

AVISPA, along with this new extension, is only able to identify as waste some work products when they are not specified as deliverables and no task defines them as input either. However, we recognize the existence of other kinds of waste that we are not yet taken into account. If a work product is defined as an input for a task then it will be assumed as useful, even if the task does not use it at all. This analysis is of finer grain and it would require the analysis of the definition of each task steps and activities. This kind of waste cannot be automatically localized for processes specified using EPF since tasks are the finer formal elements and their internal descriptions are only informal. However, the possibility of identifying the existence of the waste pattern increases the power of the other problematic patterns. Specifically, the process model efficiency could be analyzed by identifying unnecessary work products (waste patterns) and bottleneck risks (demanded work products [5]). These issues are part of our ongoing work in developing *AVISPA*.

Acknowledgments

The work of María Cecilia Bastarrica and Julio Ariel Hurtado Alegría has been partly funded by project Fondef D09I1171 of Conicyt, Chile. The work of Julio Ariel Hurtado Alegría has been also partially funded by NIC Chile.

7 Literature

[1] Stéphane Ducasse and Michele Lanza. The Class Blueprint: Visually Supporting the Understanding of Classes. Transactions on Software Engineering (TSE), 31(1):75–90, January 2005.

[2] Tudor Gîrba and Michele Lanza. Visualizing and Characterizing the Evolution of Class Hierarchies. In WOOR 2004 (5th ECOOP Workshop on Object-Oriented Reengineering), 2004.

[3] Verónica Uquillas Gómez, Stéphane Ducasse, and Theo D'Hondt. Visually Supporting Source Code Changes Integration: The Torch Dashboard. In Working Conference on Reverse Engineering, pages 55–64, Los Alamitos, CA, USA, 2010. IEEE Computer Society.

[4] Julio A. Hurtado, María Cecilia Bastarrica, and Alexandre Bergel. Analyzing the Scrum Process Model with AVISPA. In XXIX International Conference of the SCCC, pp. 60 – 65, Antofagasta, Chile, November 2010, IEEE Computer Society.

[5] Julio A. Hurtado, Maríıa Cecilia Bastarrica, and Alexandre Bergel. Analyzing Software Process Models with AVISPA. Accepted for publication in the International Conference on Software and Systems Processes, ICSSP'2011, Hawaii, USA, May 2011.

[6] Julio A. Hurtado, Alejandro Lagos, Alexandre Bergel, and María Cecilia Bastarrica. Software Process Model Blueprints. In Münch et al. [12], pages 285–296.

[7] Patrick Knab, Martin Pinzger, and Harald C. Gall. Visual Patterns in Issue Tracking Data. In Münch et al. [12], pages 222–233.

[8] Michele Lanza and Stéphane Ducasse. Polymetric Views—A Lightweight Visual Approach to Reverse Engineering. Transactions on Software Engineering (TSE), 29(9):782–795, September 2003.

[9] Peter Middleton, Amy Flaxel, and Ammon Cookson. Lean Software Management Case Study: Timberline Inc. In Proceedings of the 6th International Conference on Extreme Programming and Agile Processes in Software Engineering, XP 2005, volume 3556 of LNCS, pages 1–9, Sheffield, UK, June 2005. Springer.

[10] Sébastien Mosser, Alexandre Bergel, and Mireille Blay-Fornarino. Visualizing and Assessing a Compositional Approach of Business Process Design. In Proceedings of 9th International Conference on Software Composition (SC'10), pages 90–105. LNCS Springer Verlag, July 2010.

[11] Shahid Mujtaba, Robert Feldt, and Kai Petersen. Waste and Lead Time Reduction in a Software Product Customization Process with Value Stream Maps. In Proceedings of the 21st Australian Software Engineering Conference, ASWEC'10, pages 139–148. IEEE Computer Society, 2010.

[12] Jürgen Münch, Ye Yang, and Wilhelm Schäfer, editors. New Modeling Concepts for Today's Software Processes, International Conference on Software Process, ICSP 2010, Paderborn, Germany, volume 6195 of LNCS. Springer, July 2010.

[13] OMG. Software Process Engineering Metamodel SPEM 2.0 OMG. Technical Report ptc/08-04-01, Object Managemente Group, 2008.

[14] Kai Petersen and Claes Wohlin. Software Process Improvement through the Lean Measurement (SPI-LEAM) Method. Journal of Systems and Software, 83:1275–1287, July 2010.

[15] Mary Poppendieck. Lean Software Development. In 29th International Conference on Software Engineering, ICSE Companion Volume, pages 165–166, Minneapolis, MN, USA, May 2007, IEEE Computer Society.

[16] Mary Poppendieck and Tom Poppendieck. Lean Software Development: An Agile Toolkit. Addison-Wesley Professional, May 2003.

[17] Ken Schwaber and Mike Beedle. Agile Software Development with Scrum. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2001.

[18] Jeff Sutherland, Scott Downey, and Bjorn Granvik. Shock Therapy: A Bootstrap for Hyper-Productive Scrum. In Yael Dubinsky, Tore Dyb°a, Steve Adolph, and Ahmed Samy Sidky, editors, AGILE, pages 69–73. IEEE Computer Society, 2009.

8 Author CVs

María Cecilia Bastarrica

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 PhD. 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 formodeling software processes.

Julio Ariel Hurtado Alegría

Julio Ariel Hurtado is Asociated Professor at the Systems Department at the Universidad del Cauca. He participates at the MaTE group (Model and Transformation Engineering) since 2007 and at the IDIS (Software Engineering Research) group since 2005. He received an Electrical and Telecomunication Engineer degreee from Universidad del Cauca – Colombia in 1997. He is a PhD(c) in Computer Science from the Universidad de Chile. His main research topics are software engineering, model-driven engineering, and software process lines. His doctoral thesis focuses on applying MDE techniques and software process lines for modeling software processes.

Alexandre Bergel

Alexandre Bergel is Assistant Professor at the University of Chile. He obtained his PhD in 2005 from the University of Berne, Switzerland, under the supervision of O. Nierstrasz and S. Ducasse. After his PhD, A. Bergel made a first postdoc at Trinity College Dublin, Ireland, and a second one at the Hasso-Plattner Institute, Germany. A. Bergel is the author of over 50 articles, and has intensively published in international and peer review scientific forums, including the most competitive conferences and journals in the field of software engineering. A. Bergel and his collaborators carry out research in diverse aspects of software engineering, software quality, static and dynamic analysis.