Analyzing the Scrum Process Model with *AVISPA*

Julio Ariel Hurtado Alegría
CS Department, Univ. de Chile
IDIS Group, Univ. del Cauca, Colombia
Email: jhurtado@dcc.uchile.cl

María Cecilia Bastarrica
Computer Science Department
Universidad de Chile
Email: cecilia@dcc.uchile.cl

Alexandre Bergel
Computer Science Department
Universidad de Chile
Email: abergel@dcc.uchile.cl

***Abstract*—Scrum is a widely known agile software process model specifically designed for guiding non-technical activities in software development. This process has been formally defined in EPF and adopted by several software companies around the world. But having a process definition does not necessarily mean that it is well specified. We have developed *AVISPA*, a tool for localizing error patterns in software process models specified with EPF. In this paper, we analyze the public community specification of Scrum using AVISPA and we report our findings.**

I. Introduction

Software process definitions are relevant because they improve development effectiveness [4]. According to Feiler and Humphrey [3] software process definitions must be both useful for practitioners and reasonably economical to produce. However, misconceptions or misspecifications may be introduced during the process definition and/or specification process with a high impact when it is applied by teams in projects. That is why several organizations have decided to adopt standard process models that have been already defined and are available for use. This is the case of widely known processes such as OpenUP [7], XP [2] or Scrum [12] that are publicly available at the Eclipse web site, and also Tutelkán [16][1] for the Chilean setting. Whereas we do not pretend to judge the motivation and rational behind these processes, their specification deserves a closer look. As far as we are aware of, the implementation and definition of these processes have not been objectively analyzed so that companies could have an idea about the actual quality of the process they are adopting.

Scrum is an agile software process frequently used to rapidly develop software. It has been defined by Jeff Sutherland and more formally elaborated by Ken Schwaber [13]. 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.

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 [15]. 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. We believe that one of the reasons for this situation is, at least in part, an improper definition and implementation of Scrum.

We have developed AVISPA[2] [5], a software visualization tool that helps the process engineer to localize a series of error patterns within software process models formalized using EPF. It automatically highlights potential errors and improvement opportunities in different process blueprints so that it is not only easier to localize the errors, but also less knowledge and experience is required. This paper uses *AVISPA* for analyzing the specification of the Scrum process model[3] published in the Eclipse Process Framework community where it has been defined as a SPEM2.0 model [9] using Eclipse Process Composer[4].

The rest of the paper is structured as follows. In Sect. II we provide some background concepts about software process modeling in SPEM 2.0 in general, and specifically about Scrum and its specification using SPEM 2.0. The AVISPA tool is introduced in Sect. III. Its application for analyzing Scrum is described in Sect. IV and a discusion of the obtained results is presented in Sect. V. Finally, Sect. VI discusses some related work and Sect. VII includes a series of conclusions and some further work.

II. Background

AVISPA is a tool that analyzes software process models specified in SPEM 2.0, and in this paper we apply it to the specification of Scrum. In this section we describe what SPEM 2.0 is, what Scrum is, and how Scrum is formalized using SPEM 2.0.

A. Software Process Modeling with SPEM 2.0

The modeling of a software process refers to its definition as a model [1]. Different model representations may describe, at different levels of abstraction, the organization of the elements of a current or planned process. They provide definitions of the process to be used, instantiated, enacted or executed. So, a process model can be analyzed, validated, simulated or executed if it is defined with any of these goals.

Software and Systems Process Engineering Metamodel–SPEM 2.0 [9]–is the OMG standard for process modeling.

[1] Tutelkán: http://www.tutelkan.org

[2] AVISPA: http://squeaksource.com/ProcessModel.html

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

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

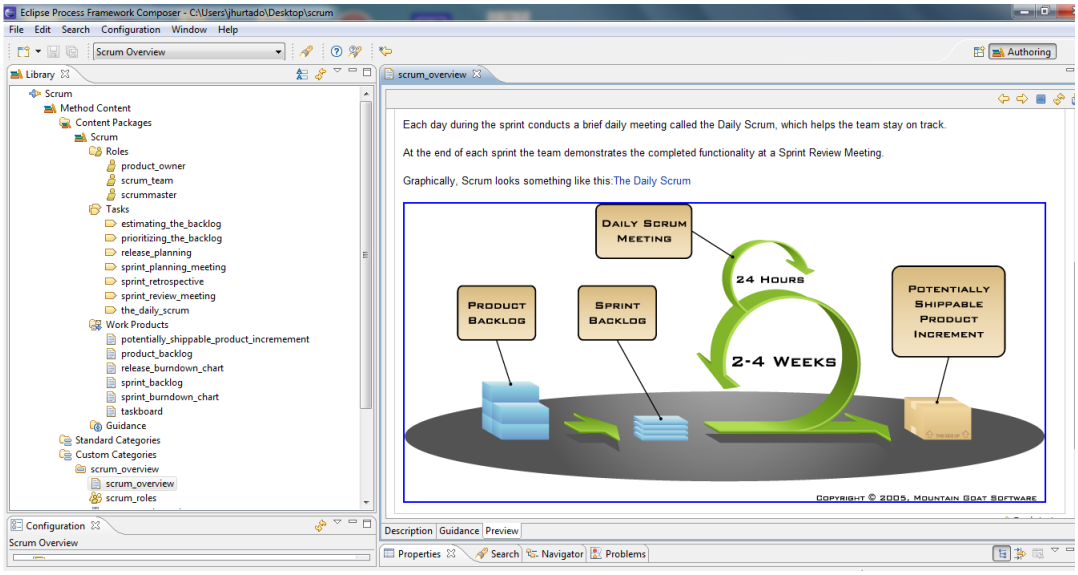

Fig. 1. Scrum Process Model

SPEM provides a standardized and managed representation of method libraries in order to allow reuse of method content. It aims to support development practitioners in defining a knowledge base for software development.

The SPEM 2.0 metamodel separates reusable method contents and their application in specific processes to promote reusability. Method content provides step-by-step explanations, describing how specific development goals are achieved independently of the placement of these steps within a development lifecycle. Processes take these method content elements and relate them into partially-ordered sequences that are customized to specific types of projects.

SPEM 2.0 is structured in seven packages:

Core Package contains classes and abstractions that build the basis for all other packages.

Process Structure Package defines the basis for defining process models as a breakdown of nested Activities with the related performing Roles, as well as input/output Work Products

Process Behavior Package extends the static structures of the process models with externally defined behavioral models, e.g. UML state and activity diagrams.

Managed Content Package introduces concepts for managing content of development processes documented and managed as natural language descriptions. These concepts can either be used as standalone or in combination with process structure concepts.

Method Content Package adds concepts for defining life cycles and process independent reusable method content elements that provide a basis of documented knowledge of software development methods, techniques, and concrete realizations of best practices. Method content describes how to achieve fine-grain development goals, by which roles, with which resources and results, independently of the placement of these elements within a specific development lifecycle. The basic concepts are Rol, Task, WorkProduct and Guidance.

Process with Methods Package facilitates integrating processes defined with Process Structure with instances of Method Content. Whereas Method Content defines fundamental methods and techniques for software development, processes place these methods and techniques into the context of a lifecycle model. **Method Plug-in Package** introduces concepts for designing and managing maintainable, reusable, and configurable libraries of method content and processes. The concepts introduced in this package allow arranging different parts of such a library based on different layers of concern.

B. The Scrum Development Process

Scrum is an agile software development method that is based on the idea that software processes are incompletely defined. So, Scrum assumes that the analysis, design, and development processes are inherently unpredictable. A control mechanism is used to manage this unpredictability and control the corresponding risk improving the process flexibility, responsiveness, and reliability [13]. Scrum is not a process or a technique for building products; it is rather a rule based framework where various processes and techniques may be applied. The goal of Scrum is to achieve the major efficacy in applying development practices while providing a framework where complex products can be developed [14].

The Scrum framework is formed by a set Scrum teams, time-boxes, artifacts and rules, as shown in Fig. 1.

Scrum teams are designed to maximize flexibility and productivity; Scrum teams are self-organizing and cross-functional, and they work in iterations. Each Scrum team

has three roles: the Scrum Master, who is responsible for ensuring that the process is understood and followed; the Product Owner, who is responsible for maximizing the value of the work that the Scrum Team does; and the Team, which does the work. The Team is formed by developers with all the skills required to transform the Product Owner's requirements into a potentially releasable piece of the product by the end of the Sprint.

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

Scrum artifacts The Product Backlog is a prioritized list of features required in the product. The Sprint Backlog is a list of tasks to perform in a Sprint, producing an increment of a potentially shippable product from the Product Backlog. A burndown is the measure of remaining Backlog over time. A release burndown measures remaining Product Backlog in the context of a release plan. A Sprint burndown measures remaining Sprint Backlog in the context of a Sprint.

Scrum lifecycle is defined by the Sprint and by three groups of phases: pregame, game and postgame. In the pregame, the planning and architecture phases are performed. In the planning phase a new release is defined according to the current Product Backlog, including an estimative of its schedule and cost. In the architecture phase an architectural and high level design is generated to determine how the backlog items will be implemented. In the game phase, the Sprints are performed. There are multiple, iterative development Sprints that are used to develop the system. In the postgame the Closure phase is performed. The release is prepared including final documentation, pre-release staged testing, and the release itself.

C. Scrum Process Model in SPEM 2.0

The Scrum process model presented by the Eclipse Process Framework Community has been defined as a SPEM 2.0 method plug-in using Eclipse Process Framework. This definition includes roles, work products and tasks, and a set of guidance, and is organized only in method packages and categories as shown in Fig. 2. The process structure has not been defined because, Scrum is inerently incomplete as a process and it is considered as a process framework more than a process itself. As a consequence, the EPF community has defined Scrum lifecycle as a Supporting Material element (a specific Guidance) where it is graphically and textually described. Although, this definition does not include all the phases defined in [13], the Scrum lifecycle could be defined and customized in a Delivery Process by each organization reusing the Scrum plug-in. The method package elements have been defined and linked according to a Scrum description as was presented above. However, the question that still remains is if the method elements will match this or other adapted life cycle. For example, are the tasks outputs and inputs consistent among the tasks within a value flow? These are relevant questions mainly when the model is used for first time, or for comparing or combining with other process models.

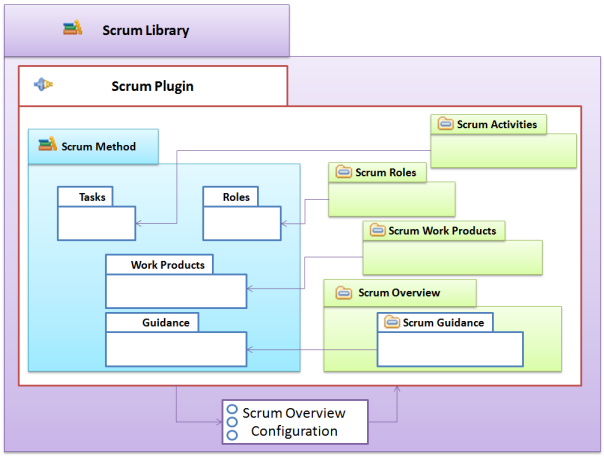

Fig. 2. Scrum specified as a SPEM 2.0 model

III. The *AVISPA* Tool

Process model blueprints are graphical representations meant to help process designers to assess the quality of software process models and identify potential anomalies [6]. The essence of these blueprints is to facilitate the comparison between elements using a graph metaphor, composed of nodes and edges. The size of a node or an edge tells us about their relative importance. We have defined three blueprints that help identifying opportunities for improving software process models, each one focusing on a particular process element: roles, tasks and work products, namely Role Blueprint, Task Blueprint, and Work Product Blueprint. The approach has been complemented with the *AVISPA* tool (Analysis and VIsualization for Software Process Assessment) [5], a visualization tool built on Mondrian, Moose and Glamour.

This tool imports process models defined in SPEM 2.0 from EPF and automatically produces specialized blueprints where empirically found error patterns are localized. Error patterns are identified structurally as either disconnected elements or elements whose relative size is beyond one standar deviation from the mean. These error patterns have been empirically found to occur frequently in industrial software process model conceptualization and specification. They include: having no guidance associated to certain element, having roles that have too many responsibilities or that do not collaborate with others, tasks that are not specific enough in their specification, work products that are required for too many tasks, and independent subprojects. Table I summarizes these error patterns and briefly describes in which blueprint they can be identified.

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 deviation larger than the mean.
Isolated role	A role that does not collaborate.	ROLE BLUEPRINT	A node that is not connected with an edge.
Multiple purpose tasks	Tasks with too many output work products.	TASK BLUEPRINT	Nodes whose more than one deviation wider than the mean.
Demanded Work products	Work products required for too many tasks.	WORK PRODUCT BLUEPRINT	Nodes more than one deviation higher than the mean.
Independent subprojects	Independent subgraphs.	TASK BLUEPRINT or WORK PRODUCT BLUEPRINT	Subgraphs that are not connected with edges.

TABLE I
ERROR PATTERNS IDENTIFIED BY *AVISPA*

IV. ANALYZING THE SCRUM PROCESS MODEL

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

A. No guidance associated

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

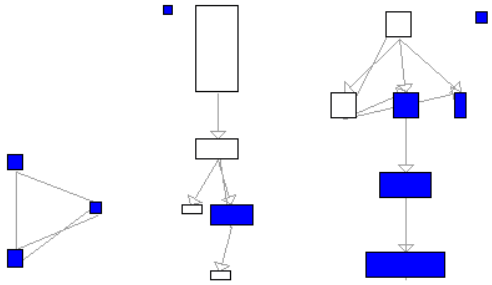

Fig. 3. ROLE BLUEPRINT, WORK PRODUCT BLUEPRINT and TASK BLUEPRINT identifying elements without guidelines

B. Overloaded role and Isolated role

Generating the ROLE BLUEPRINT, we found that there are neither overloaded nor isolated roles, as shown in Fig. 4. Thus, we can conclude that there are no problems in the specification of Scrum with respect to error patterns referring roles.

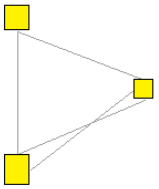

Fig. 4. Identifying overloaded or isolated roles

C. Multiple purpose tasks

A task that is too wide in the TASK BLUEPRINT will be colored in red in order to call the attention of the process engineer. A task with too many output work products does not have one clear goal, so it may be better to divide it into more specific subtasks. In Fig. 5, we can see that the *Daily Scrum* task is significantly wider than the others. This is expected because this task is defined as a black box hiding the complexity of the software development in Scrum, because each *Daily Scrum* is implemented according to the technicalities of another process model. But, as a software development process in itself, the specification of Scrum is not detailed enough. This is consistent with the literature and the fact that Scrum should be combined with other methods.

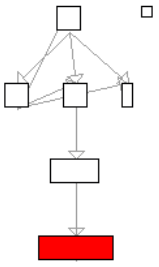

Fig. 5. Localizing task without a clear purpose

D. Demanded work products

A work product required for the execution of too many tasks could become a bottleneck, so a work product that is too demanded reveals a problem in process conceptualization. A node in the WORK PRODUCT BLUEPRINT that is too high identifies this kind of problem. This is the case of the *Product backlog* that can be clearly identified in red in Fig. 6.

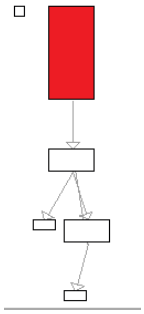

Fig. 6. Identifying work products that are too demanded

E. Independent subprojects

Having independent subprojects reveals a misspecification in the process model because all tasks and work products should be useful for the project's goals, and as such they should be connected in the TASK BLUEPRINT and the WORK PRODUCT BLUEPRINT, respectively. This error pattern may be seen in either blueprint. In Fig. 7 we show how independent subgraphs have different colors in the WORK PRODUCT BLUEPRINT in the left, and in the TASK BLUEPRINT in the right for Scrum. The work product in yellow, *Potentially Shippable Product Increment*, belongs to an independent graph. This implies that this work product is neither defined as an input nor as an output of any task in Scrum. A similar situation occurs in the green task *Sprint Retrospective* that is disconnected in the TASK BLUEPRINT.

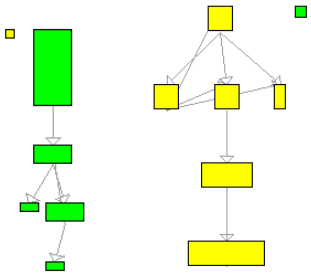

Fig. 7. Independent subprojects in the WORK PRODUCT BLUEPRINT and TASK BLUEPRINT

V. ANALYSIS AND DISCUSSION

Some incompleteness were found in the specification of the Scrum process model. An improved version can be completed: (i) guidance available in the Scrum definition should be associated to each role; for instance *Product Backlog* Example, *Story Points* Key Concept and *Priorization of the Backlog* guideline should be associated to the Product Owner Role, and the *Taskboard* Work Product should be completed with a Taskboard example, and (ii) some tasks need to be completed associating them with their required and produced work products; for instance, because the *Sprint Burn Down* and the *Taskboard* Work Products are used and modified in the *Spring Retrospective* task, they need to be associated as input and output, respectively, and the *Potentially Shippable Product Increment* should be defined as an output of the *Sprint Review Meeting* task.

Both, the *Daily Scrum* task that was found to be defined with too little detail, and the *Product backlog* that was found to be too demanded, are consistent with the agile phylosophy behind Scrum, and should not be considered as errors. They are rather warnings that causion should be taken with them.

VI. RELATED WORK

A close work by Osterweil and Wise [10] presents an analysis of Scrum using Little-JIL. This analysis determines a weak point when the product is integrated in each development work. So, because continuous integration is not part of Scrum, it can fall in long periods of development without integration, generating a bottleneck. Their approach consists of analyzing the whole process directly and this requires knowledge and experience for being effective, whereas *AVISPA* focuses in analyzing process models using a strategy based on metrics, a visual metaphor, and error patterns, and thus encapsulating the necessary knowledge. Therefore, our approach needs less experienced process engineers.

The Agile Software Solution Framework (ASSF) [11] has been used to evaluate Scrum along six aspects of an agile software development methodology: agility, process, people, product, tools and abstraction[5]. Their result shows that Scrum has a higher level of agility for its practices compared to other agile methods. The study realized with ASSF is orthogonal to ours. We focus on a widely used specification of Scrum, not on the benefit over other agile methods.

VII. CONCLUSIONS

A software process model definition can be left incomplete for different reasons, as is the case of some features of Scrum that need to be left as agile as possible. However, if a relevant principle of the process is not included, it could be applied in an inappropriate way. In this paper we have analyzed the Scrum process model with the AVISPA tool and we have found that the specification that is widely used by the community has been incompletely defined. This may explain, at least in part, the gap between the expected and the reported performance of Scrum.

We are currently applying AVISPA also for analyzing software process models that have been defined by Chilean companies. As part of this work we have been able to prove its usefulness for process engineers as a means for aiding their work mainly when the process evolves. We have also been able to validate and refine the error patterns that have been originally identified.

REFERENCES

[1] Silvia Teresita Acuña and Xavier Ferré. Software process modelling. In *World Multiconference on Systemics, Cybernetics and Informatics, ISAS-SCIs 2001, July 22-25, 2001, Orlando, Florida, USA, Proceedings, Volume I: Information Systems Development*, pages 237–242, 2001.

[5]ASSF is also employed to evaluate *Knowledge* and *IT governance*. We restrict the scope of our comparison to *Method core*.

[2] Kent Beck and Cynthia Andres. *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional, 2nd edition, November 2004.

[3] Peter H. Feiler and Watts S. Humphrey. Software Process Development and Enactment: Concepts and Definitions. In *ICSP, International Conference of Software Process*, pages 28–40, Berlin, Germany, 1993. IEEE Computer Society.

[4] Watts S. Humphrey. *Managing the software process*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[5] Julio A. Hurtado, María Cecilia Bastarrica, and Alexandre Bergel. AVISPA: Localizing Improvement Opportunities in Software Process Models. Technical Report TR/DCC-2010-6, Computer Science Department, Universidad de Chile, July 2010.

[6] Julio A. Hurtado, Alejandro Lagos, Alexandre Bergel, and María Cecilia Bastarrica. Software Process Model Blueprints. In Münch et al. [8], pages 285–296.

[7] Ivar Jacobson. *The Road to the Unified Software Development Process*. Cambridge University Press, July 2000.

[8] 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, July 8-9, 2010. Proceedings*, volume 6195 of *Lecture Notes in Computer Science*. Springer, 2010.

[9] OMG. Software Process Engineering Metamodel SPEM 2.0 OMG Specification. Technical Report ptc/07-11-01, OMG, 2008.

[10] Leon J. Osterweil and Alexander E. Wise. Using Process Definitions to Support Reasoning about Satisfaction of Process Requirements. In Münch et al. [8], pages 2–13.

[11] A. Qumer and B. Henderson-Sellers. A framework to support the evaluation, adoption and improvement of agile methods in practice. *J. Syst. Softw.*, 81(11):1899–1919, 2008.

[12] Ken Schwaber. *Agile Project Management with Scrum*. Microsoft Press, 1st edition, February.

[13] Ken Schwaber. SCRUM Development Process. In *Proceedings of the 10th Annual ACM Conference on Object Oriented Programming Systems, Languages, and Applications (OOPSLA)*, pages 117–134, 1995.

[14] Ken Schwaber and Jeff Sutherland. Scrum, February 2010. http://www.scrum.org/storage/scrumguides/Scrum

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

[16] Gonzalo Valdés, Hernán Astudillo, Marcello Visconti, and Claudia López. The Tutelkán SPI Framework for Small Settings: A Methodology Transfer Vehicle. In *Proceedings of the 17th European Conference on SPI, EuroSPI 2010, Systems, Software and Services Process Improvement*, volume 99, pages 142–152, Grenoble, France, September 2010. Communications in Computer and Information Science.