Is it Safe to Adopt the Scrum Process Model?

Julio Ariel Hurtado Alegría[1, 2], **María Cecilia Bastarrica**[1], **Alexandre Bergel**[1]
[1]MaTE, Computer Science Department, Universidad de Chile
Santiago, Chile
[2]IDIS Research Group, University of Cauca
Popayán, Colombia
{jhurtado,cecilia,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.

Keywords: Software Process Model, Scum, Process Analysis, Agile Methods.

1. 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 [13], 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 philosophy behind these processes, their specification deserves a closer look. As far as we are aware of, the implementation and definition of these processes hase 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 [12]. 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, in most cases with XP.

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 may be an improper definition and implementation of Scrum, i.e., adopting the public Scum process model as it is, without paying much attention to the quality of this specification that not always follows all the ideas behind 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 or improvement opportunities in different process blueprints so that it is not only easier to localize the errors, but also less

[1] Tutelkán: http://www.tutelkan.org
[2] AVISPA: http://squeaksource.com/ProcessModel.html

knowledge and experience is required for analyzing them and potentially fixing them. 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 SPEM 2.0 model [9] using Eclipse Process Composer[4].

The rest of the paper is structured as follows. In Sect. 2 we provide some background concepts about software process modeling in SPEM 2.0 in general, and a description of the Scrum process model and its specification using SPEM 2.0. The *AVISPA* tool is introduced in Sect 3; there all the error patterns it is able to identify are described. Its application for analyzing Scrum is described in Sect.**¡Error! No se encuentra el origen de la referencia.**, and a discussion of the obtained results is presented in Sect. 4. Finally, Sect. 5 discusses some related work and Sect. 6 includes a series of conclusions and describes some ongoing work.

2. BACKGROUND

AVISPA is a tool for analyzing 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.

2.1 Software Process Modeling with SPEM 2.0

Modeling a software process refers to its definition as a model [1]. Different model representations may describe, at different levels of abstraction and with different notations, the organization of the elements of a current or planned process. They provide definitions about the process to be used, instantiated, enacted and/or executed. So, a process model can be analyzed, validated, simulated or executed if it is defined with any of these goals. However, if we want to automatically analyze, validate, simulate or execute the process model, it must be defined using a formal notation.

Software and systems Process Engineering Metamodel (SPEM 2.0) [9] is the OMG standard for process modeling. 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 and maintenance.

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 particular development life cycle. Processes take these method content elements and relate them into partially ordered sequences that are customized to specific types of projects.

SPEM 2.0 is structured in seven packages:

- **Core Package** contains classes and abstractions that build the basis for all other packages.
- **Process Structure Package** defines the basis for defining process models as a breakdown of nested Activities with the related performing Roles, as well as input/output Work Products.
- **Process Behavior Package** extends the static structures of the process models with externally defined behavioral models, e.g. UML state and/or activity diagrams.
- **Managed Content Package** introduces concepts for managing content of development processes documented and managed as natural language descriptions. These concepts can either be used as standalone or in combination with process structure concepts.
- **Method Content Package** adds concepts for defining life cycles and process independent reusable method content elements that provide a basis of documented knowledge of software development methods, techniques, and concrete realizations of best practices. Method content describes how to achieve fine-grain development goals, by which roles, with which resources and results, independently of the placement of these elements within a specific development life cycle. The basic concepts are Role, Task, Work Product and Guidance.

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

- **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 life cycle 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.

2.2 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 programming processes are inherently unpredictable. Therefore, a control mechanism is used to manage this unpredictability and control the corresponding risk improving the process flexibility, responsiveness, and reliability [12]. Scrum is not a process or a technique for building software products, it is rather a rule based framework where various processes and techniques may be applied. The goal of Scrum is to optimize the efficacy in applying development practices while providing a framework where complex products can be developed [14].

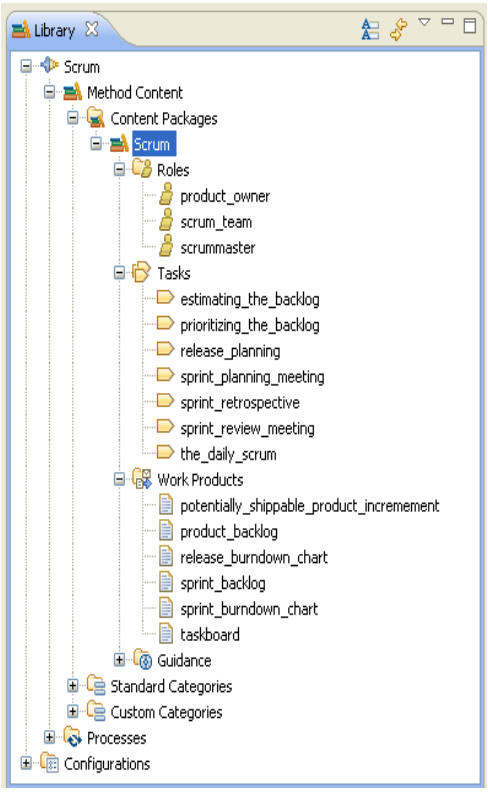


Figure 1: Scrum Process Model

The Scrum framework is formed by a set Scrum teams, time-boxes, artefacts and rules, as shown in Figure 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 *Sprint*s use the same Scrum framework, and all *Sprint*s deliver an increment of the final product that is potentially releasable.

Scrum artefacts. 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 the remaining of the *Product Backlog* in the context of a release plan. A *Sprint burndown* measures the remaining of the *Sprint Backlog* in the context of a *Sprint*.

Scrum life cycle 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 estimation of its schedule and cost. In the *architecture* phase an architectural and a high level design are generated in order to determine how the backlog items will be implemented. In the *game* phase, the *Sprint*s are performed. There are multiple, iterative development *Sprint*s 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.

Figure 2 shows the integration of Scrum elements into a complete process.

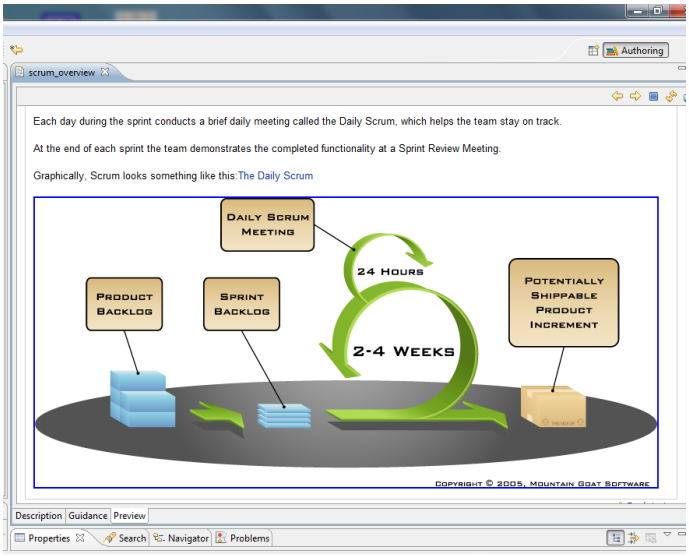


Figure 2: The Publicly Available Scrum Process Model

2.3 Scrum Process Model Specified 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 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 Figure 3. The process structure has not been defined because Scrum is inherently incomplete as a process and it is considered as a process framework more than a complete process itself. As a consequence, the EPF community has defined Scrum life cycle 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 [12], the Scrum life cycle could be defined and customized in a Delivery Process by each organization adopting 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 the first time, or for comparing or combining it with other process models.

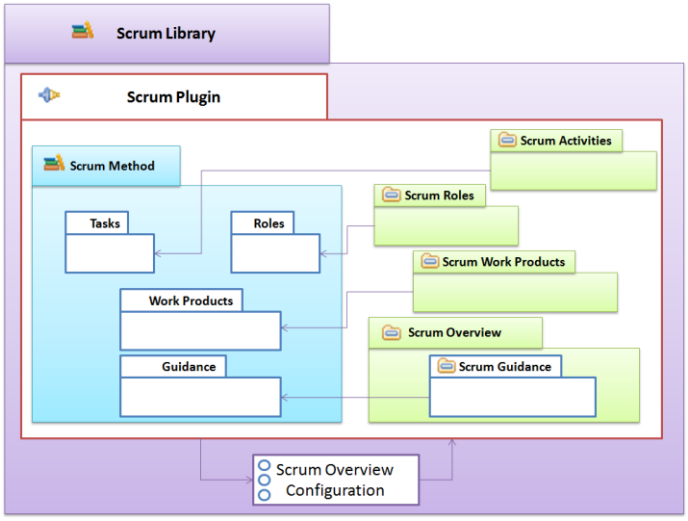


Figure 3: Scrum specified as a SPEM 2.0 model

3. THE *AVISPA* TOOL

Process model blueprints are graphical representations meant to help process designers to assess the quality of software process models and suggest 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 tells us about their relative importance, and the existence of an edge tells us about relationships between nodes. 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*.

In the *Role Blueprint*, nodes are roles whose size represents the number of tasks in which they are involved, and edges between two nodes indicate role collaboration (two roles working together in a task). In the *Task Blueprint*, nodes are tasks whose height and width represent the number of input and output work products of the task, respectively. Edges between two nodes represent precedence: a task T1 precedes another task T2 if there is an output work product of T1 that is an input work product of T2. In the *Work Product Blueprint* nodes represent work products whose dimensions represent the number of tasks that write and read the work product, respectively. An edge between two work products WP1 and WP2 implies that there is a task that consumes WP1 and produces WP2.

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 and highlighted. Error patterns are identified structurally as either disconnected elements or elements whose relative size is beyond one standard 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, independent subprojects, and useless work products. Table 1 summarizes these error patterns and briefly describes in which blueprint they can be identified.

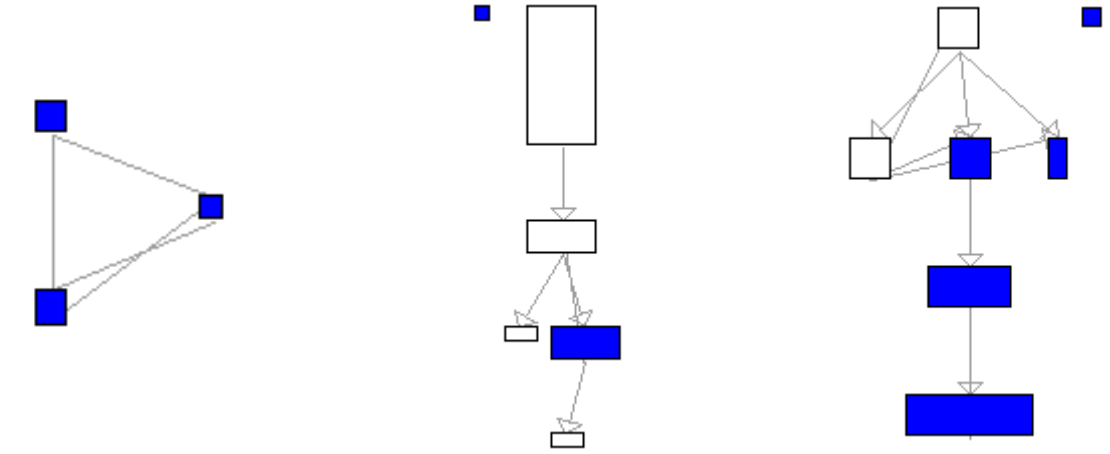
The *AVISPA* tool was used for analyzing the Scrum process model defined by the EPF process community. It is exported from EPF as an XML file and imported in *AVISPA*. The analysis is guided by the kind of error patterns we are trying to identify and localize, so the analysis is presented accordingly.

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 role that is not connected with an edge to other roles
Multiple purpose task	Task with too many output work products	Task Blueprint	Node that is more than one standard deviation wider than the mean
Demander work product	Work product required for too many tasks	Work Product Blueprint	Node 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
Waste work products	Useless work product	Work Product Blueprint	Work products that are neither deliverables nor input for any task

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 (Figure **4**4 (a)). In the *Work Product Blueprint*, the *Taskboard* and the *PotentiallyShippableProductIncrement* have no guidance either (Figure **4**4 (b)). 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 (Figure **4**4 (c)). 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. Moreover, the lack of guidance is especially dangerous for newcomers, making it less safe to adopt Scrum as it is specified.



(a) Roles without guidance

(b) Work products without guidance

(c) Tasks without guidance

Figure 4: Model elements without guidance

Overloaded role and Isolate role

Generating the *Role Blueprint*, we found that there are neither overloaded nor isolated roles, as shown in Figure 5: all nodes are similar in size and they are fully connected. Thus, we can conclude that there are no problems in the specification of Scrum with respect to these error patterns referring roles.

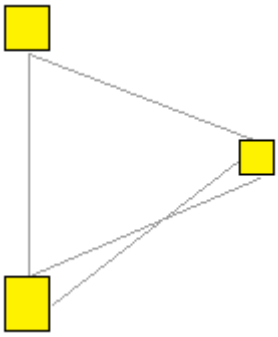


Figure 5: Identifying overloaded and isolated roles

Multiple purpose tasks

A task that is too wide in the *Task Blueprint* will be colored in red in order to call the attention of the process engineer. A task with too many output work products does not have one clear goal, so it may be better to divide it into more specific subtasks. In Figure 6, 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, due to the fact that 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.

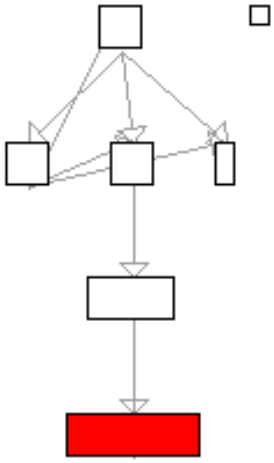
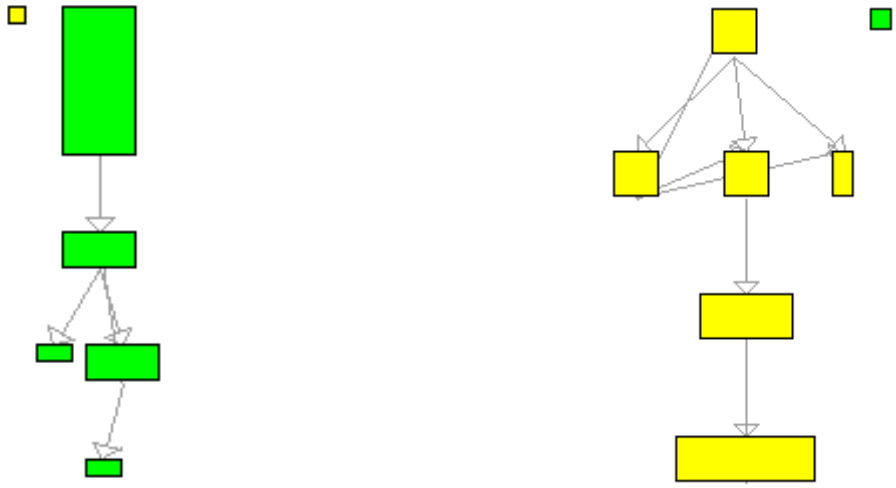


Figure 6: Localizing tasks without a clear purpose

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 both, the *Task Blueprint* and the *Work Product Blueprint*. This error pattern may be seen in either blueprint. In Figure 7 (a) we show how independent subgraphs have different colors in the *Work Product Blueprint*, and in Figure 7 (b) the *Task Blueprint* also shows the existence of independent subgraphs. 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*.



(a) Independent subprojects in the Work Product Blueprint

(b) Independent subprojects in the Task Blueprint

Figure 7: Localizing independent subprojects

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 the 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 Figure 8.

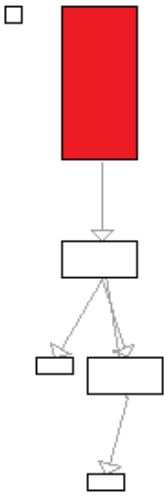


Figure 8: Localizing too demanded work products

Waste work product

Deliverables are those artefacts 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. Therefore, work products may be either deliverable work products or intermediate work products that are needed mainly for coordinating successive tasks. However, if there are work products that are neither deliverables nor input for any other task within the process, they are waste. All leaves in the *Work Product Blueprint*, i.e., nodes with no successor, should represent deliverable work products. *AVISPA* highlights in blue all those leaves that are not deliverables. The process engineer then needs to

analyze all highlighted nodes so that he/she could determine if they are actually required as input of a task, and thus they are not leaves, if they should have been defined as deliverables and thus they should not have been highlighted, or if they are actually waste in the process. Figure 9 shows three potential waste work products in the Scrum process model: *Potentially Shippable Product Increment*, *Release Burndown Chart*, and *Sprint Burndown Chart*. Analyzing them we find that they are all underspecifications and the Scurm process model does not include waste work products, as is expected from an agile method.

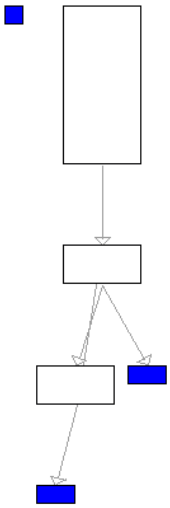


Figure 9: Potential waste work products

4. ANALYSIS AND DISCUSSION

In general no conceptual errors were identified in the Scrum process model. However, a series of incompleteness was found. An improved version can be completed as follows:

(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.

(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.

(iii) The *Potentially Shipable Product Increment* should be specified as an input of the *Integration* task; the *Release Burndown Chart* and the *Sprint Burndown Chart* are clear inputs for the *Development* task and they should be specified as so.

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 philosophy behind Scrum, and should not be considered as errors. They are rather warnings that caution should be taken with them.

5. 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. This is consistent with our findings about not having any useless work product included. But the study realized by ASSF is orthogonal to ours. We focus on a publicly available and widely used process model specification of Scrum, and we analyze how good it is, not necessarily on the benefits of applying Scrum in particular, or applying other agile methods in general either.

6. CONCLUSION

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 in some aspects. This may explain, at least in part, the gap between the expected and the reported productivity in projects that apply Scrum. Obtaining similar results than previous analyses of Scrum that have followed different methods gives us confidence on the effectiveness of our tool.

We are currently applying *AVISPA* also for analyzing industrial software process models that have been defined by Chilean companies. In this context the number and variety of errors identified is much larger since standard processes, as is the case o Scrum, have generally been improved over time with the collaboration of a user community. In the case of proprietary processes, evaluation can only come from trial and error if there is no analysis tool. This makes *AVISPA* much more useful. 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. We have recently added new error patterns such as having tasks without any role assigned to it, among others.

REFERENCES

[1] Silvia Teresita Acuña and Xavier Ferré. Software process modeling. 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.

[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

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

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. Journal of Systems and Software, 81(11):1899-1919, 2008.

[12] 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.

[13] Ken Schwaber. Agile Project Management with Scrum. Microsoft Press, 1st edition, February 2004.

[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.