Analyzing Software Process Models with AVISPA

Julio A. Hurtado Alegría
Computer Science Dept.
Universidad de Chile
IDIS Research Group
University of Cauca
jhurtado@dcc.uchile.cl

María Cecilia Bastarrica
Computer Science Dept.
Universidad de Chile
cecilia@dcc.uchile.cl

Alexandre Bergel
Computer Science Dept.
Universidad de Chile
abergel@dcc.uchile.cl

ABSTRACT

Software process models are sophisticated and large specifications aimed at organizing and managing software development. Their formal specification demands an enormous effort, but once specified there are few approaches and even fewer tools that aid the process engineer to analyze the quality of the process. For the last five years we have aided software companies in specifying their software processes and we have found a series of error patterns that indicate the potential presence of misconceptions or misspecifications. This paper presents these patterns, characterizes the kinds of errors they potentially reveal, and details how errors could be localized within a software process model. To assist process engineers to analyze the quality of their processes, we provide AVISPA, a tool that graphically renders different aspects of a process model and highlights potential errors as intuitive and comprehensible indicators. The approach and the supporting tool are illustrated by applying them for analyzing the software process models of three Chilean software companies.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management—*software process models*; D.2.8 [**Software Engineering**]: Metrics—*software process quality*

General Terms

Management, Verification

Keywords

Software process models, quality assessment, model-driven engineering

1. INTRODUCTION

Counting on a well defined software process model is determinant for achieving software quality and process productivity. Therefore, many companies have undertaken software process specification and improvement as a priority project. However, conceptualizing the software process model demands an enormous effort for making explicit common practices and defining practices that do not exist within the company yet. Standards such as ISO/IEC15504 [11] and maturity models such as CMMI [21] are generally used as guidelines for defining this process. But there are still no standard wide-spread mechanism for determining the quality of the specified process, and thus the return-of-investment of software process definition is not always clear.

For the last five years we have worked in aiding small software companies in Chile to define their development processes in an effort to improve national industry standards[1]. As part of this practical experience, we have found that there are some typical recurrent errors in software processes, some of them due to conceptual errors in the process design and other errors introduced during process specification. But none of them are easily identified, let alone localized, because of the enormous amount of process elements involved, multiple views, and informal notations that may sometimes introduce ambiguity.

In a previous work, we have proposed process model blueprints [10] as a means for visualizing and analyzing different perspectives of a software process model. The three blueprints considered (ROLE BLUEPRINT, TASK BLUEPRINT, and WORK PRODUCT BLUEPRINT) are applied to software process models defined using SPEM 2.0 [16], the OMG standard notation for process specification. The process blueprints we proposed enable the identification of *exceptional entities* [4], *i.e.,* exceptions in the quantitative data collected. Blueprints were successfully used to identify a number of flaws in an industrial process model, but a lot of experience from the process engineer is required for identifying these flaws. Since then, we have assessed several other industrial process models, and we discovered a set of *recurrent patterns* ranging from suboptimal modeling to misconceptions and misspecifications.

This paper is about presenting, formalizing and validating these recurrent error patterns. We rigorously define recurrent errors appearing in software process models, and we explain their potential consequences. We also show how each of these error patterns can be identified within a software process blueprint.

ICSSP '11, May 21-22, 2011, Waikiki, Honolulu, HI, USA

[1]Tutelkán: Achieving High Quality in National Software Industry by Applying Reference Processes (`www.tutelkan.org`).

We have built AVISPA (Analysis and VIsualization for Software Process Assessment)[2], a tool that builds blueprints and highlights error patterns. Counting on this tool, the process engineer only needs to analyze highlighted elements, demanding less experience and also less previous knowledge for effective process model analysis, and adding usability as well.

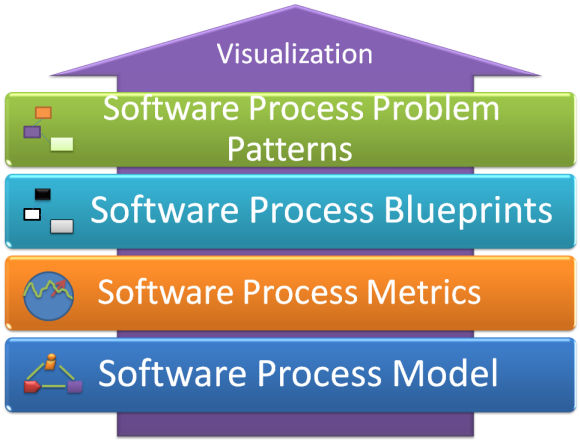

Figure 1: AVISPA in localizing software process model improvement opportunities

Using visualization to identify error patterns is not new. A large body of research use visual patterns to identify positive or negative properties of software systems [12, 19, 25]. However, none of the related work we are aware of elaborated on visual patterns for identifying problems in software process models.

Figure 1 depicts the pile of technologies involved in our proposal. Software process models specified in SPEM 2.0 are assumed to exist. We consider SPEM 2.0 for our work since it is the standard of OMG, and it has also been promoted within the Chilean software industry by the Tutelkán project. On top of them we define a series of software process metrics that will be used for identifying errors and improvement opportunities. Software process blueprints are built using these metrics. Error patterns are identified as those elements or constructs within blueprints whose values for certain metrics satisfy some constraints. The identified elements are visually highlighted using AVISPA.

We have applied AVISPA for analyzing the process models defined in three different Chilean software companies. We have been able to find several of the defined error patterns, and most of them resulted in actual errors, giving support to our intuition that a formal tool that helps the process engineer is useful. AVISPA has been highly welcomed in all companies we have worked with, and process engineers also pointed out that it is relevant for them to count on AVISPA for maintaining their software process model, an application we have not envisioned before. We report some of these experiences.

The rest of the paper is structured as follows. Section 2 presents a description of empirically found recurrent errors, as well as a description of their implications, and how they look in AVISPA. A detailed description of the AVISPA tool is included in Sect. 3 and its application for localizing error patterns in three industrial software process models is reported in Sect. 4. Related work is discussed in Sect. 5. Finally, some conclusions and further work are presented in Sect. 6.

[2]`http://www.moosetechnology.org/tools/ProcessModel`. AVISPA is freely available under the MIT license.

2. PROCESS MODEL ERROR PATTERNS

For the past five years we have conducted applied research in the area of software process models in small software companies in Chile [8, 23] and Iberoamerica [20, 24] as part of the Tutelkán and Competisoft projects. Along this work we have identified a number of common errors and problematic situations in software process model specifications, either due to misconceptions or misspecifications. In this section we report a series of these patterns, how they may be identified in AVISPA, and mainly how we are able to automatically highlight them as part of the blueprint where they appear. In this work, we say that there is a misspecification in the software process model if the development process is well designed but its specification does not necessarily reflect the actual practices, e.g., there exist some guidance for a role but it is not specified as part of the model. We say that there is a misconception whenever there is a flaw in the software process design, e.g., a task produces a work product that neither a task nor a role needs.

Section 2.1 is a summary of our previous work [10] on process model blueprints. It is necessary to introduce it since we augmented blueprint visualizations with new information to identify error patterns. Readers familiar with Process Model Blueprints may safely skip Sect. 2.1. Section 2.2 presents the error patterns we have identified.

2.1 Process Model Blueprints

ROLE BLUEPRINT, TASK BLUEPRINT and WORK PRODUCT BLUEPRINT are three graphical views of a software process model. Each of them focuses on a particular aspect of the process model, namely roles, tasks and work products. Each blueprint is depicted as a polymetric view [12], a lightweight software visualization technique enriched with software metrics information, that has been successfully used to provide software maps.

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). Figure 3 shows an example of a ROLE BLUEPRINT.

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. Figure 2 depicts a TASK BLUEPRINT.

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.

2.2 Potential Errors

There is a number of anomalies in software process specifications that we have realized that are fairly frequent. We here describe some of them along with their consequences

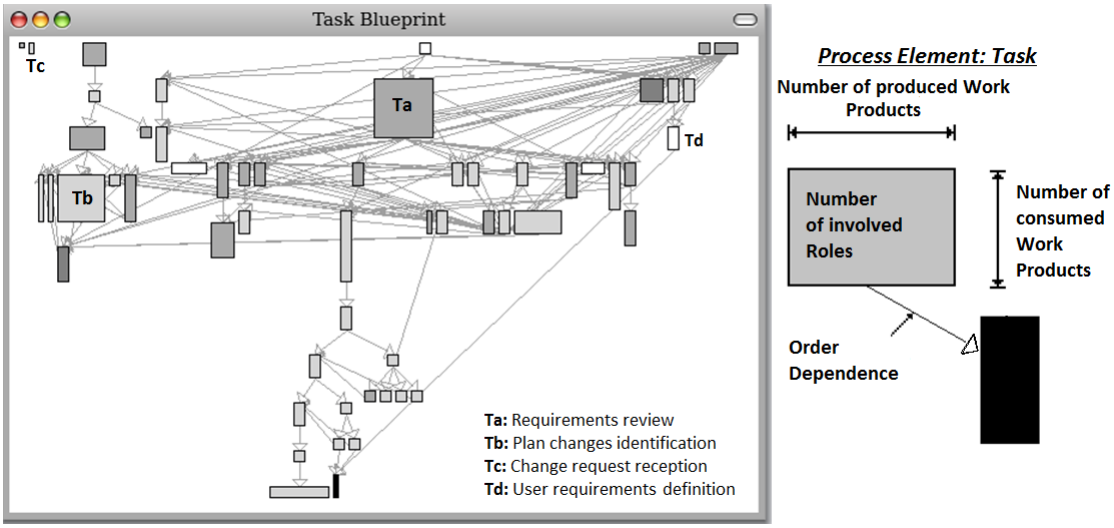

Figure 2: Task Blueprint **where potential errors are suggested but not localized**

and how they would look in the blueprint where they may be found. We also provide a tentative quantification for how bad may be considered too bad, so that it serves as a basis for automating their localization.

Overloaded roles. If a role is involved in a large number of tasks, it becomes a risk: if it fails, all the associated tasks within the process will fail as well. This is a clear anomaly in the process model conception. Better choices would be either specializing the role by dividing its responsibilities, or reassigning some tasks to other roles. We would say that a role is overloaded if it is more than one standard deviation larger than the average size. This error pattern is computed as part of the Role Blueprint, and we highlight overloaded roles in red. In Fig. 3 we can see that the Developer role is much larger than the others, and thus it may be overloaded.

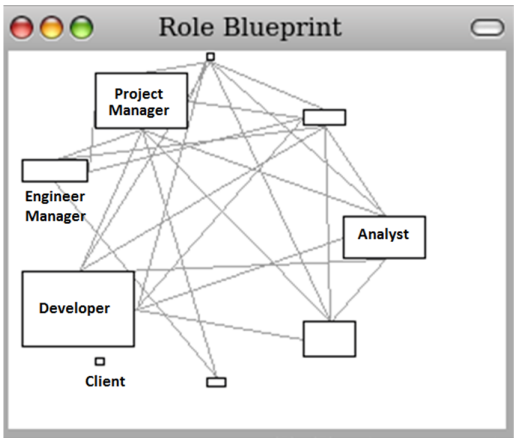

Figure 3: Role Blueprint

Isolated roles. There may be certain tasks that a role executes by itself, but it is not frequently right to have a role that never collaborates in any task with other roles. In general, this kind of error pattern shows a misspecification: a role should have been assigned to take part of certain task but appears to be left apart. This error pattern is also apparent in the Role Blueprint, and we highlight isolated roles in green. In Fig. 3, the Client role is isolated.

Roles, Tasks and Work Products without guidance. If a role, task or work product has no guidance about how to be executed, there is a big chance that it will not be properly done. This error is generally a misspecification, meaning that there should have been certain guidance associated with each element. In the respective Blueprint we highlight elements without guidance in blue.

Multiple purpose tasks. A process where tasks have too many output work products may reveal that these tasks are not specified with the appropriate granularity as may be the case of Ta and Tb in Fig. 2. A task with too many output work products may be too complex since its goal is not unique. This may reflect a misconception in the process model. This pattern is seen in the Task Blueprint where we highlight wide nodes (too many output work products) in red. We consider a task to be too complex if it is wider than one standard deviation from the average task width.

Work products required for too many tasks. Work products required for a high number of tasks may cause serious bottlenecks when they are not available, and thus it could reveal a misconception. This situation is seen in the Work Product Blueprint where we highlight in yellow nodes whose width (number of tasks that require it as an input) is more than one standard deviation from the average.

Independent subprojects. In a Task Blueprint and a Work Product Blueprint, tasks and work products are related with edges indicating precedence. Considering that the process model specifies the way to proceed when working on one unique project, it is conceptually odd to have disconnected subgraphs, both in the Task Blueprint and the Work Product Blueprint. In general, these situations arise due to underspecifications, when work products have not been specified as input or output work products for certain tasks when they should have been. We represent each subgraph with a different color in both, the Task

Table 1: Error patterns identified by AVISPA

Error pattern	Description	Localization	Identification
No guidance associated	An element with no guidance associated.	any blueprint	A blue 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.

BLUEPRINT and the WORK PRODUCT BLUEPRINT, in order to identify the existence of independent subprojects. So having a graph with more than one color nodes indicates that there are independent subprojects specified.

Table 1 summarizes the error patterns that have been identified so far.

3. LOCALIZING ERRORS WITH AVISPA

This section sketches the internals of the implementation of AVISPA. The scripts for implementing two of the error patterns are subsequently offered as examples of the way errors are computed. Finally, a description of the tool from the user point of view is provided.

3.1 Implementation of AVISPA

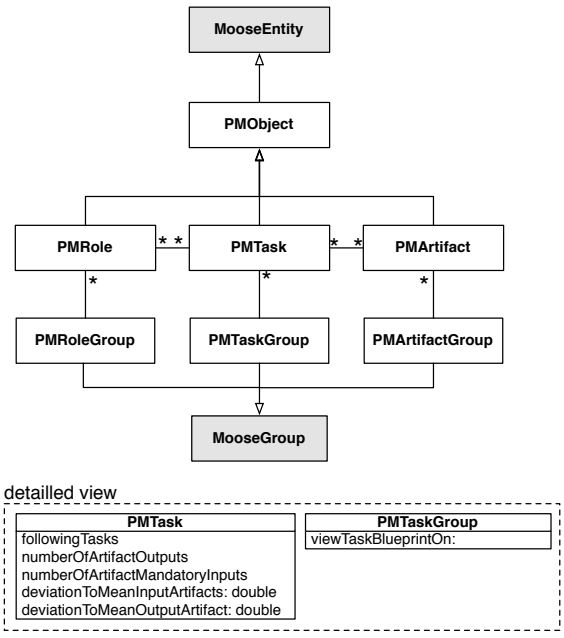

Figure 4: The AVISPA metamodel (gray classes belong to FAMIX)

The SPEM 2.0 error patterns presented in the previous section are implemented in AVISPA extending the MOOSE platform. As Fig. 4 shows, AVISPA extends the FAMIX family of metamodels of MOOSE[3] by subclassing MooseEntity and MooseGroup. The names of the classes that belong to AVISPA begin with PM, standing for Process Model. PMObject contains operations and attributes common to all SPEM elements (essentially a particular identifier). PMRole, PMTask and PMArtifact describe elementary components of SPEM 2.0. Each of these classes offers methods for computing metrics and navigating through a model. For example, each task is aware of its following tasks (*i.e.,* tasks that farther need to be completed) and its associated artifacts. A group of roles, tasks and artifacts are expressed as instances of PMRoleGroup, PMTaskGroup and PMArtifactGroup, respectively. The purpose of offering specialized collections is to enable dedicated visualization to be defined on these groups. For example, the method viewTaskBlueprintOn: is defined on PMTaskGroup which defines the enhanced task blueprint describe below.

AVISPA is visualized using the Mondrian visualization engine[4] [15]. Mondrian operates on any arbitrary set of values and relations to visually render graphs. As exemplified below, visualizations are specified with the Mondrian domain specific language.

3.2 Error Pattern Implementation in AVISPA

We illustrate the implementation of two error patterns: independent projects and multiple purpose tasks, i.e., tasks involving too many output work products. We provide the script for each of them, and the rationale in each implementation. The implementation of the other error patterns is conceptually similar to these ones.

Independent subprojects.

This kind of error is seen, for example, when the TASK BLUEPRINT has disconnected subgraphs. Thus, each independent subgraph is colored differently, and having a TASK BLUEPRINT with more than one color means that there are some missing dependencies. On the other hand, if the TASK BLUEPRINT is all the same color, this will mean that there are no independent subprojects, and therefore the process will be fine with respect to this error pattern. The following script builds a colored TASK BLUEPRINT where independent subprojects are identified. Independent subproject always reveal an error in the process model specification.

```
PMTaskGroup>> viewTaskBlueprintOn: view
  | ds components orderedComponents normalizer cycleColor |
```

[3] `http://www.moosetechnology.org/docs/famix`
[4] `http://www.moosetechnology.org/tools/mondrian`

```
"Compute disjoint sets"
ds := MalDisjointSets new.
ds nodes: self.
ds edges: self from: #yourself toAll: #followingTasks.
ds run.

components := ds components.
orderedComponents := Dictionary new.
components doWithIndex: [:roles :index |
        roles do: [:role |
                orderedComponents at: role put: index]].

"Assign a color to each set"
normalizer := MONIdentityNormalizer new.
(1 to: (components size * 10) ) do: [:v | normalizer moValue: v].
cycleColor := [:v | normalizer moValue:
        ((orderedComponents at: v) + 10)].


"Display the blueprint"
view shape rectangle
        borderColor: Color black;
        borderWidth: 1;
        fillColor: [:v | cycleColor value:v];
        width: [:each | each numberOutputs * 10];
        height: [:each | each numberInputs * 10].
view nodes: self.
view shape arrowedLine.
view edges: self from: #yourself toAll: #followingTasks.
view treeLayout
```

First a cycle is computed so that edges of connected subgraphs are painted with the same color. Then, individual nodes are built assigning them a size and a color. Tasks are represented as rectangular nodes whose color is that of the subgraph it belongs to. Their width is related to the number of output work products, and the height shows the number of input work products. Arrows between two nodes exist if they are related with the *followingTasks* relationship. The whole blueprint is shown as a tree.

Multiple Purpose Tasks.

Here again the error will be seen in the TASK BLUEPRINT, but now nodes that are wider than one standard deviation from the average number of output work products will be highlighted as potential errors. Highlighted tasks reveal complexity in the task specification, but they are not necessarily errors. One standard deviation in a normal distribution function was the empirical value calibrated from a preliminar analysis.

A serie of metrics are precalculated so that the script can be executed. $numberOutputs_i$ is the number of output work products of task i in the process. Then, considering that there are n tasks in the process, we can calculate the mean number of output work products for the whole process as follows:

$$MeanOutWP = \frac{\sum_{i=1}^{n} numberOutputs_i}{n} \quad (1)$$

And then, the standard deviation can be calculated as follows:

$$sigmaOut = \sqrt{\frac{\sum_{i=1}^{n}(numberOutputs_i - MeanOutWP)^2}{n}} \quad (2)$$

Also, the distance from the mean value to $MeanOutWP$ is calculated as follows:

$$distToMeanOutWP_i = numberOutputs_i - MeanOutWP \quad (3)$$

These metrics are used as part of the script in order to determine the color of each node in the TASK BLUEPRINT.

```
viewTaskWarningBlueprintOn: view
view shape rectangle
  fillColor: [:each | (each distToMeanOutWP >
                        self myModel sigmaOut)
        ifTrue: [Color red]
        ifFalse:[Color white]];
  borderColor: Color black;
  width: [:each | each numberOutputs * 10];
  height: [:each | each numberInputs * 10].view nodes: self.
view shape arrowedLine.
view edges: self from: #yourself toAll: #followingTasks.
view treeLayout.
view root interaction item:
                'inspect group' action: [:v | self inspect]
```

The main part of the script is devoted to determining the color of each node according to its relative size. If the distance from the number of output work products to the mean is larger than one standard deviation, then the node will be red. Otherwise, the node will be white. Edges will be drawn according to the *followingTasks* set that should have been precalculated. The whole blueprint is presented as a tree.

Obtaining a TASK BLUEPRINT that is all white means that all tasks have similar complexity with respect to the number of output work products. Several red tasks clearly suggests a poor design because the purpose of the tasks is not always uniquely defined.

3.3 AVISPA User Interface

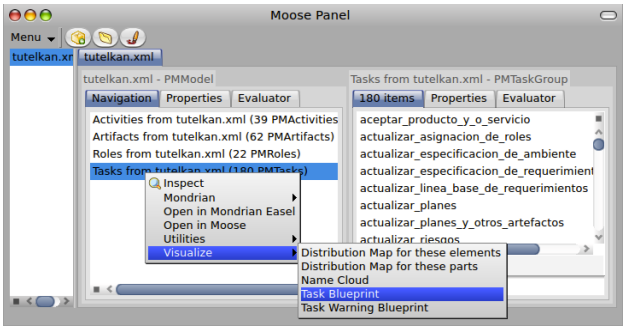

Figure 5: The AVISPA main user interface

AVISPA has become a useful tool to import and visualize SPEM 2.0 based process models. It is built on top of Moose and the Pharo programming language[5], and so it benefits from a large toolset for navigation and visualization. Figure 5 shows the main user interface. The Tutelkán model has been loaded and it is ready to be analyzed. The navigation panel shows four entry points to begin an analysis: activities, artifacts, roles and tasks. Navigation is realized through the

[5] `http://www.pharo-project.org`

information available in the metamodel (see Sect. 3.1). Although not depicted, metrics and other specific information (*e.g.,* descriptions and annotations) are also available under the *properties* tab.

4. APPLYING AVISPA TO REAL WORLD SOFTWARE PROCESS MODELS

AVISPA was applied on three Chilean software companies: Amisoft, BBR Engineering and DTS. First, we briefly present the context of each company, and then we describe the results of applying AVISPA to analyze the process models defined in each of them. In order to be able to compare results we choose to analyze the three processes according to the same error patterns: disconnected subgraphs in the TASK BLUEPRINT and tasks with too many output work products. Also, disconnected subgraphs could have been analyzed in the WORK PRODUCT BLUEPRINT. The process models used in this research were developed in the last two years; these models were obtained from the respective libraries using the exporter feature of EPF. The process models were analyzed in the MaTE (Model and Transformation Engineering Group) Laboratory and then the results were discussed with the respective process engineers.

4.1 Application Scenarios

Amisoft is around ten years old and it is formed by thirteen qualified employees. Its main goal has been to deliver specialized and quality services. Its development areas are: client/server architecture solutions, enterprise applications based on J2EE and Systems integration using TCP/IP and MQ Series. Amisoft has started its software process improvement project in 2009, and it is currently implementing the ISO9001:2008 standard and the CMMI model. Its software process model has been inspired by OpenUP.

BBR Engineering is one of the main software factories of BBR, an international consulting company since 1994. It is formed by twenty four employees specialized in different roles including architects, project managers, developers, quality assurance specialists and analysts. BBR Engineering has developed solutions mainly in the area of retail; specifically, its main areas are: points of sale, payment systems, communications and interfaces, e-business, and integration. The company has started its software process improvement project in 2009 using the Tutelkán Reference Process as a reference for its implementation.

DTS started business around 1990 from a joint venture between a Chilean Aeronautic company, EANER and ELTA Electronics Industries. DTS works in solutions for military and civil technology. It has 250 employees, including engineers, certified technicians, operation workers and managers. DTS started to define its software process model in 2008, using the Rational Unified Process as a reference. In DTS there is no specific software process improvement project; its effort has been oriented toward recovering the software process actually applied in the organization, in order to formalize it, analyze it, and eventually improve it.

4.2 Results

Amisoft. In this study case, AVISPA helped to identify 5 instances of the pattern *independent subprojects* corresponding to the nodes with a color different than blue in Fig. 6. These nodes represent the tasks: *Configuration Items Update*, *Non-Compliant Communications*, *Delivery Document Generation and Sending*, *Getting Configuration Items* and *Execute Unitary Test to Interfaces and Communications*. These disconnected subgraphs (in this case disconnected tasks) represent a high risk because the configuration management process could be chaotic (everybody needs to know how to get and put configuration items) and the testing of interfaces and communications could be forgotten just when it is required the most. Looking for independent subprojects in the WORK PRODUCT BLUEPRINT is a dual case: whenever there are errors in one, there are also errors in the other. Independent projects based on the WORK PRODUCT BLUEPRINT facilitates to find isolated work products: *Directory Structure, Case Test Template, Client Satisfaction Survey and Glosary*. These work products are not adequately linked with the rest of the process elements being this ambiguous for the process users.

The *multiple purpose task* pattern was applied on Amisoft Proces Model (red nodes in Fig. 7). In this case the result was 9 potential errors of multiple purpose tasks out of 93 tasks in total (9.7%). However, reviewing these tasks, many of them refer to management tasks where different inputs are required to evaluate the project advance or to make some decisions, and as result these tasks modify many work products. So, the granularity cannot be finer, but more guidance could be added. However, the task *Document Requirements* could have been better decomposed into two different tasks separating abstraction levels (concerns covered or users of the requirements). On the other hand the task *Measure Data Collection* is shown as a multipurpose task and really it is, because the measurement results are not available directly in a unique work product; instead of this, the data is available in many work products according to the metrics established in the measurement plan.

BBR Engineering. Similarly, we have applied AVISPA for finding disconnected graphs and multiple purpuse tasks to the process model of BBR Engineering, and the results are shown in Figs. 8 and 9, respectively. Many tasks were found disconnected, 29 out of 79 (36.7%), and this situation shows that the process presents many underspecifications, increasing the risk of not applying it as intended. Most of the problems are related to project management and configuration management. The configuration management issues reveal the process immaturity in BBR Engineering contrary to project management. However, both process components must be specified with more precision. Nevertheless, there are some tasks underspecified which could be problematic when the process is instanciated: *User Needs Understanding, Requirements Priorization, Measure Data Collection, Unit Data Base Testing, Unit Component Testing* and *Interfaces and Communication Testing*. When the independent projects pattern was also applied on the WORK PRODUCT BLUEPRINT using AVISPA, similar problems arose on project management, but excluding these general problem, the problematic work products were: *User Interface Model, Integration Plan* and *Design Document*. This process would inject many technical problems at instanciation time because it includes many imprecisions in tasks and work products of requirements, technical solution, testing and project management areas.

The process in BBR includes 79 tasks, and 9 of them are identified as problematic (11.4%) according to the multiple purpose task pattern (red nodes in Fig. 9). Similarly to the

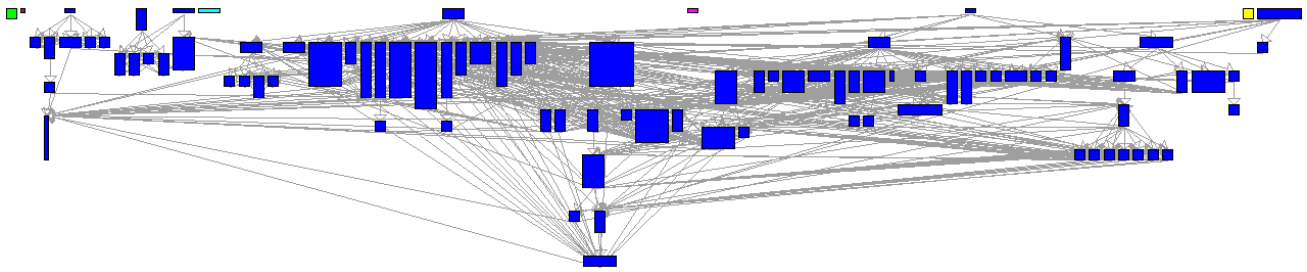

Figure 6: Task Blueprint **for localizing disconnected subgraphs in Amisoft.**

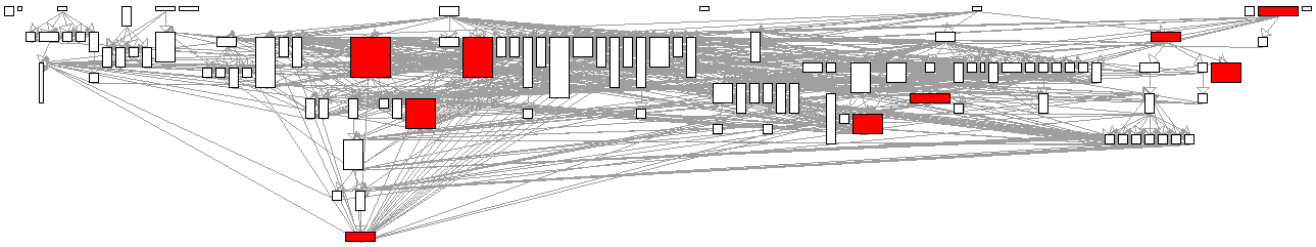

Figure 7: Task Blueprint **for localizing multiple purpose tasks in Amisoft.**

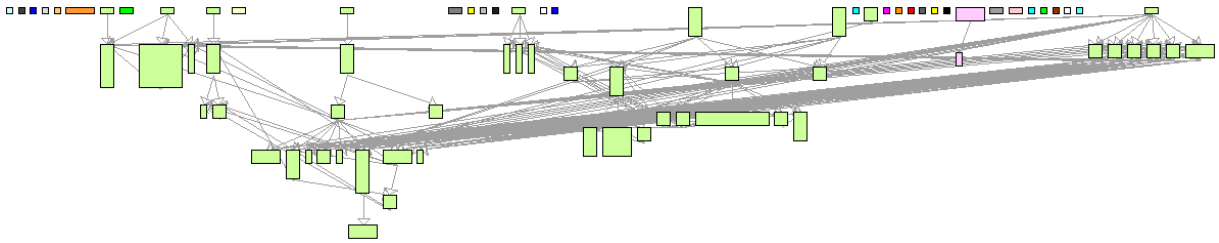

Figure 8: Task Blueprint **for localizing disconnected subgraphs in BBR Engineering.**

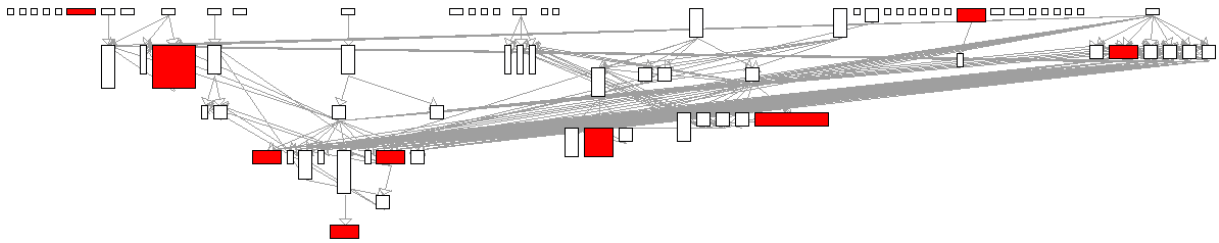

Figure 9: Task Blueprint **for localizing tasks involved with too many work products in BBR Engineering.**

previous case study, the project management task defines many outputs and for the same reasons cannot be changed. However, the tasks *Data Base Design* and *Component Design* could be decomposed to reach a homogeneous process model definition.

DTS. Avispa was also applied to the process of DTS for identifying and localizing both kinds of error patterns (see Figs. 10 and 11). Only two tasks were found disconnected showing a careful specification job: *Identify Requirements Provider* and *Change Requirements Reception*. These tasks are critical to manage requirements change, so this part of the process would not be instanciated adequately at projects.

The process in DTS includes 57 tasks, and 6 of them were found to be multi purpose tasks (10.5%) painted in red in Fig. 11. Similar to the previous cases, most of these tasks are part of the project management and this characteristic cannot be changed. But, the *Generating Implementation Document* and *Requirements Review* tasks could be decomposed,

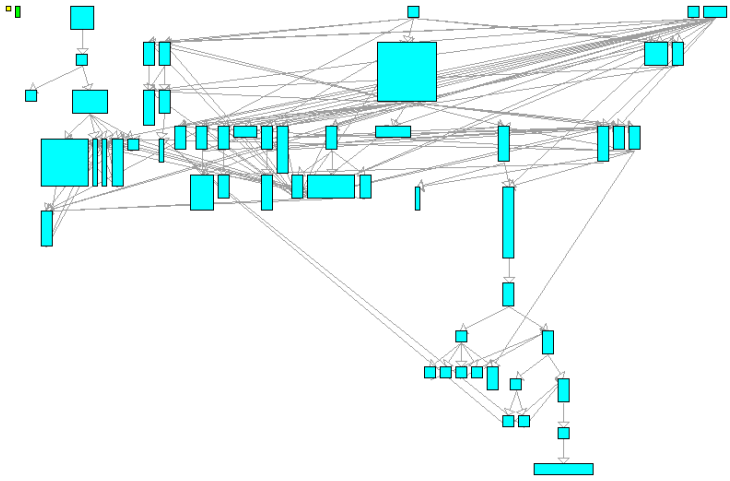

Figure 10: TASK BLUEPRINT **for localizing disconnected subgraphs in DTS.**

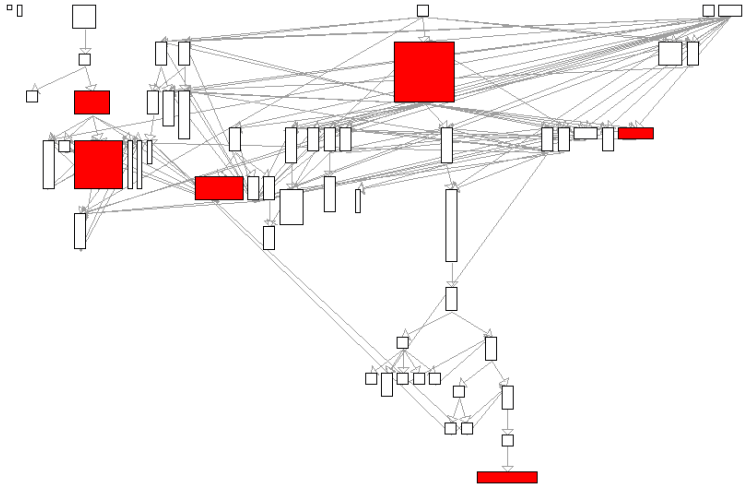

Figure 11: TASK BLUEPRINT **for localizing tasks involved with too many work products in DTS.**

whereas *Help Diagram Development* could be specialized for each specific help diagram to be designed.

5. RELATED WORK

Software process model quality can be addressed from different approaches: metrics [2], testing [11,21], simulation [7], or formal verification [6]. Metrics work fine for data of the overall software process model, but metrics for partial portions of the process or individual process elements are usually not a suitable presentation for a reviewer. Process testing is an effective way to evaluate a process model; assessments and audits are based on data of executed projects, but executing the process is expensive. Our approach provides a means for a priori evaluation of the software process quality. Cook and Wolf [3] present a formal verification and validation technique for testing and measuring the discrepancies between process models and actual executions. The main limitation of testing is that it can only be carried out once the process model has already been implemented, tailored and enacted. Simulation has a shorter cycle, but it still requires enactment data to be reliable. Formal checking is effective too but it presents semantic limitations [26]. In AVISPA we use metrics as a basis for building two visualization layers that help process engineers to localize problems in the models. We propose a complementary approach to analyze software process models in an early way, based on reviewing the architectural views of a software process model defined as Software Process Blueprints [10]. There, each blueprint is built following a model-driven strategy where the process model is separated in a set of partial views that may be more illustrative for finding errors. However, the usability and complexity of process model blueprints was threatened when dealing with large and complex process models. According our practical experience with AVISPA, it has improved usability by identifying a set of common error patterns, and highlighting them, but more importantly it encapsulates specialize knowledged of an expert software process engineer for identifying improvement opportunities and thus requiring less experienced process engineers.

As stated by Osterweil [17], software processes are software too, so techniques that apply to software programs can be also applied in process models. Finding error patterns in source code has been fairly successful [5, 13], so following a similar approach we have been able, based on a vast empirical experience, to automatically identify and localize a series of error patterns in software process models. The classical domain for software visualization is software source code. There has been a great deal of work on visualizing classes and methods [12], software architecture [14], and even source code annotations [1]. The work presented in this paper has the same rationale: providing concise information about an engineering artifact in order to maintain and improve it. By taking some of these ideas and applying them to analyze software process models, the analysis obtained similar benefits to those achieved with other software artifacts.

Osterweil and Wise [18] demonstrated how a precise definition of a software development process can be used to determine whether the process definition satisfies some of its requirements. A definition of a Scrum process written in the Little-JIL process definition language is presented to motivate their contributions. We developed a similar analysis [9] to Scrum process model using AVISPA where we found a number of specification problems. In general both approaches show the advantages of a precise specification of the process. Applying AVISPA implies also to import a EPF process, measure it and to generate a set of views with interactive instances of the patterns. So, the process engineer can analyze the software process quality. Soto et. al. [22] present a case study that analyzes a process model evolution using the history of a large process model under configuration management with the purpose of understanding model changes and their consequences. The goal of Soto et. al. is oriented toward the impact of the changes in the software process models whereas our approach is oriented to early analyzing a new or changed process model just before the model is used in a specific project. Soto's approach also analyzes the changes on the process elements using many XML files.

6. CONCLUSIONS

In this paper we have presented a set of error patterns, implemented as part of AVISPA, a tool for process model analysis that localizes a set of identified potential errors within a process model specified in SPEM 2.0. These errors may come either from process conceptualization or from misspecifications. We describe how each of the error patterns identified are found in the appropriate process blueprint, and we made AVISPA highlight them.

The process models of some of the Chilean companies we were working with for the last years have been analyzed using AVISPA. Some errors were found, as well as some improvement opportunities, showing the effectiveness of the patterns and the tool. These errors were not foreseen by process engineers, but they agreed they wer real improvement opportunities. We have also applied AVISPA for analyzing the publicly available process model of Scrum[6] [9]. The errors found in this case are consistent with other reported analyses.

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

The quality of the analysis highly depends on the quality of the definition of the error patterns. Even though the error patterns presented in this paper have shown to be effective in finding improvement opportunities, there is some room for fine tuning them. For example, determining that a task is too complex if it is more than one standard deviation from the mean, may not help discriminating really badly specified elements, and maybe two standard deviations is a better measure. We may also define the error pattern as parametric in the number of standard deviations considered.

The tool is targeted to those software process models formally specified in SPEM 2.0. This may be one of its main limitations since it is hard and expensive to formally define a complete process mainly for small software organizations. However, if a company decides it is worth the effort, then AVISPA provides an added value to this investment assuring, at least partially, the quality of the specified process.

As part of our ongoing work, we are defining a set of solution patterns that the tool will suggest, so that each error pattern found could be solved in an assisted manner. In this way, the round trip for software process improvement will be complete. Therefore, we will be ready to apply AVISPA to ten Chilean software enterprises as part of ADAPTE, a large government funded project.

Acknowledgments

This work has been partly funded by project Fondef D09I1171 of Conicyt, Chile. The work of Julio A. Hurtado has been also partially funded by a scholarship of NIC Chile.

7. REFERENCES

[1] A. Brühlmann, T. Gîrba, O. Greevy, and O. Nierstrasz. Enriching reverse engineering with annotations. In *International Conference on Model Driven Engineering Languages and Systems*, volume 5301 of *LNCS*, pages 660–674. Springer-Verlag, 2008.

[2] G. Cánfora, F. García, M. Piattini, F. Ruiz, and C. A. Visaggio. A family of experiments to validate metrics for software process models. *Journal of Systems and Software*, 77(2):113–129, 2005.

[3] J. E. Cook and A. L. Wolf. Software process validation: quantitatively measuring the correspondence of a process to a model. *ACM Transactions On Software Engineering Methodology*, 8(2):147–176, 1999.

[4] S. Demeyer, S. Ducasse, and O. Nierstrasz. *Object-Oriented Reengineering Patterns*. Morgan Kaufmann, 2002.

[5] J. Durães and H. Madeira. Emulation of Software Faults: A Field Data Study and a Practical Approach. *IEEE Transactions on Software Engineering*, 32(11):849–867, 2006.

[6] J. Ge, H. Hu, Q. Gu, and J. Lu. Modeling Multi-View Software Process with Object Petri Nets. *ICSEA 2006*, 0:41, 2006.

[7] V. Gruhn. Validation and verification of software process models. In *Proc. of the Software development environments and CASE technology*, pages 271–286, 1991.

[8] J. A. Hurtado and M. C. Bastarrica. Tutelkán Implementation Process: Adapting a Reusable Reference Software Process in the Chilean Software

Industry. Technical Report TR/DCC-2010-4, Computer Science Department, Universidad de Chile, June 2010.

[9] J. A. Hurtado, A. Bergel, and M. C. Bastarrica. Analyzing the Scrum Process Model with *AVISPA*. In *Proceedings of the SCCC'2010*, Antofagasta, Chile, November 2010. To Appear.

[10] J. A. Hurtado, A. Lagos, A. Bergel, and M. C. Bastarrica. Software Process Model Blueprints. In *Proceedings of the International Conference on Software Process*, volume 6195 of *LNCS*, pages 273–284, July 2010.

[11] ISO. /IEC 15504 : Information technology - software process assessment and improvement. Technical report, Int. Organization for Standardization, 1998.

[12] M. Lanza and S. Ducasse. Polymetric Views-A Lightweight Visual Approach to Reverse Engineering. *Transactions on Software Engineering*, 29(9):782–795, Sept. 2003.

[13] B. Livshits and T. Zimmermann. Dynamine: finding common error patterns by mining software revision histories. *SIGSOFT Softw. Eng. Notes*, 30(5):296–305, 2005.

[14] M. Lungu and M. Lanza. Softwarenaut: Exploring Hierarchical System Decompositions. In *Proceedings of the 10th European Conference on Software Maintenance and Reengineering*, pages 351–354, 2006.

[15] M. Meyer, T. Gîrba, and M. Lungu. Mondrian: an agile information visualization framework. In *Proceedings of the ACM 2006 Symposium on Software Visualization, SOFTVIS*, pages 135–144. ACM, 2006.

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

[17] L. J. Osterweil. Software Processes Are Software Too. In *International Conference on Software Engineering*, pages 2–13, 1987.

[18] L. J. Osterweil and A. E. Wise. Using Process Definitions to Support Reasoning about Satisfaction of Process Requirements. In J. Münch, Y. Yang, and W. Schäfer, editors, *ICSP*, volume 6195 of *LNCS*, pages 2–13. Springer, 2010.

[19] F. Perin, T. Gîrba, and O. Nierstrasz. Recovery and analysis of transaction scope from scattered information in Java enterprise applications. In *Proceedings of International Conference on Software Maintenance 2010*, Sept. 2010.

[20] F. J. Pino, J. A. H. Alegria, J. C. Vidal, F. García, and M. Piattini. A process for driving process improvement in VSEs, international conference on software process, icsp 2009 vancouver, canada, may 16-17, 2009 proceedings. In *ICSP*, volume 5543 of *Lecture Notes in Computer Science*, pages 342–353. Springer, 2009.

[21] SEI. CMMI for Development, Version 1.2. Technical Report CMU/SEI-2006-TR-008, Software Engineering Institute, 2006.

[22] M. Soto, A. Ocampo, and J. Münch. Analyzing a software process model repository for understanding model evolution. In *Proceedings of the International Conference on Software Process: Trustworthy Software Development Processes*, ICSP '09, pages 377–388, Berlin, Heidelberg, 2009. Springer-Verlag.

[23] G. Valdés, H. Astudillo, M. Visconti, and C. López. The Tutelkán SPI Framework for Small Settings: A Methodology Transfer Vehicle. In *Proceedings of the 17th EuroSPI*, volume 99, pages 142–152, Grenoble, France, September 2010. Communications in Computer and Information Science.

[24] R. Villarroel, R. Fajardo, and O. Rodríguez. Implementation of an Improvement Cycle using the Competisoft Methodological Framework and the Tutelkán Platform. *CLEI Electronic Journal*, 13(1), April 2010.

[25] R. Wettel, M. Lanza, and R. Robbes. Software systems as cities: A controlled experiment. In *Proceedings of ICSE'11*, 2011. to appear.

[26] I.-C. Yoon, S.-Y. Min, and D.-H. Bae. Tailoring and Verifying Software Process. In *8th Asia-Pacific Software Engineering Conference (APSEC 2001)*, pages 202–209, Macau, China, 2001.