AVISPA: Localizing Improvement Opportunities in Software Process Models

Julio Ariel 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 evaluate the quality of the process. According to the industrial experience we conducted over the last five years, we have found a series of software process model patterns that indicate the potential presence of misconceptions or misspecifications. This paper presents these patterns, characterizes the kind of error they potentially reveal, and details the graphical indicator we used to localize potential errors within a software process. To assist process engineers to assess the quality of their processes, we provide AVISPA*, a tool that graphically renders different aspects of a process model. Potential errors are highlighted using intuitive and comprehensible indicators. The approach and the supporting tool are illustrated by applying them for evaluating the software process models of three industrial case studies.*

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 [10] and maturity models such as CMMI [17] are generally used as guidelines for this process. But there are still no standard widespread 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 been working 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 others just specification errors. 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. For example, a role that does not collaborate with others, even though is not an error in itself, may indicate a misconception in the software process. Similarly, if we consider that a task precedes another one if the output artifacts of the former are inputs for the latter, then having islands of disconnected subgraphs suggests an error in the software process specification. Also, having tasks that involve too many input and/or output work products, may become bottlenecks and a better process design may require dividing these tasks, but it is not always clear how many is too many.

In a previous work, we have proposed process model blueprints [9] as a means for visualizing and analyzing different perspectives of a software process model. The three blueprints we consider (Role Blueprint, Task Blueprint, and Work Product Blueprint) are applied to software process models defined using SPEM 2.0, the standard notation for process specification. The process blueprints we proposed are effective

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

in identifying *exceptional entities* [4], *i.e.,* exceptions in the quantitative data we collected. Blueprints were successfully used to identify a number of flaws in an industrial process model. Since then, we have assessed a number of 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 patterns.

We rigorously define a series of 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 as part of a software process blueprint. We have built *AVISPA* (Analysis and Visualization for Software Process Assessment)[2], a tool that builds blueprints and highlights error patterns. It has been developed on top of MOOSE and using the Mondrian environment for displaying blueprints. Counting on this tool, the process engineer only needs to analyze highlighted elements, demanding less experience for effective process model evaluation, and adding usability as well.

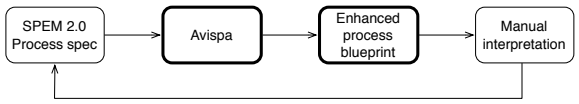

Figure 1. AVISPA in the software process model improvement process

The specification of a software process model requires a domain specific language. 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. The whole software process improvement will be benefited with the use of AVISPA, and the methodology that is followed is that shown in Figure 1. This paper is about describing AVISPA and the enhanced generated process blueprints (**bold** components in Figure 1).

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. All processes analyzed had some error, giving support to our hypothesis that a formal tool that helps the process engineer is completely required. We report some of these experiences.

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

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 a process model blueprint. 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. Problematic Process Model Patterns

For the past five years we have been conducting applied research in the area of software process models in small software companies in Chile [8, 18, 19] as part of the Tutelkán project. Along this work we have identified a number of common errors and problematic situation 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 within a process model blueprint, and mainly how, through a reverse engineering approach, we are able to automatically highlight them as part of the blueprint where they appear.

Section 2.1 is a summary of our previous work [9]. It is necessary to introduce it since we augmented our previous visualizations with new information to identify problematic model patterns. Readers familiar with Process Model Blueprints may safely skip Sect. 2.1. Section 2.2 informally presents the problematic patterns we have identified. They are revisited in more detail in later on.

2.1. Process Model Blueprints

ROLE BLUEPRINT, TASK BLUEPRINT and WORK PRODUCT BLUEPRINT are three graphical representations of a process model. Each of them focuses on a particular aspect of the process model, namely roles, tasks and work products. Each blueprint is visualized as a polymetric view [11], 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 that work together in the same task). Following the principle of the exceptional entities reengineering pattern [4], nodes that are significantly larger than others may be a symptom of overloaded roles, and isolated roles could suggest a specifi-

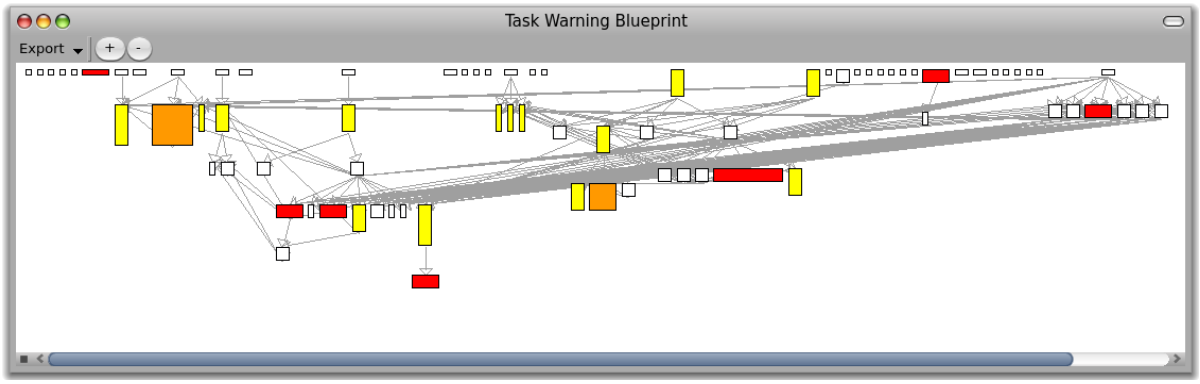

Figure 2. Enhanced TASK BLUEPRINT with potential errors highlighted

cation error since those roles are not collaborating with any other in the whole process.

In the TASK BLUEPRINT, nodes are tasks whose hight 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 task T2 if there is an output work product of T1 that is an input work product of T2. A node that is significantly longer compared to others is likely to be a bottleneck of the process since this task has to wait for all input work products to be available in order to begin. Also having too many initial tasks (with no predecessors) may be an indication of a misspecification since the process generally starts as a consequence of a decision that triggers the rest of the process. An example of an enhanced TASK BLUEPRINT is depicted in Figure 2. The original version of TASK BLUEPRINT uses gray scale to indicate associated guidelines. As it is described later on, in this enhanced version, colors indicate potential anomalies.

In the WORK PRODUCT BLUEPRINT nodes are work products whose dimensions represent the number of tasks that write and read the work product, respectively. Here, a large node may also represent a bottleneck since significantly large work products are required for a great deal of tasks. An edge between two work products WP1 and WP2 implies that there is a task that consumes WP1 and produces WP2. Even though there may be many initial and final work products not suggesting any error, disconnected subgraphs strongly suggest the existence of independent subprojects, revealing conception or specification problems.

2.2. Potential Errors

As outlined in the previous section, there is a number of patterns likely to reveal anomalies in software process specifications that we have realized that are fairly frequent. We here describe some of them along with their consequences 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 could serve as a basis for automating localization.

Overloaded roles. If a role is involved in a large number of tasks, it becomes a risk: if it fails, 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 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 can be 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.

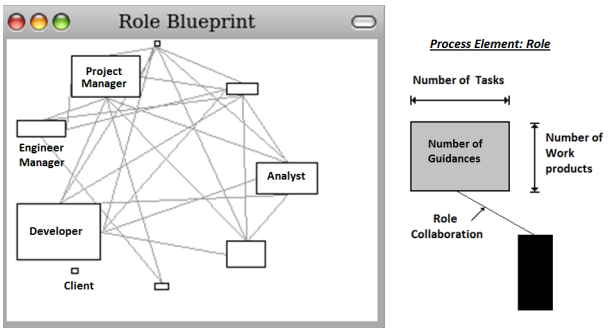

Figure 3. Role Blueprint

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 a certain task but it was forgotten. 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.

Tasks that involve too many work products. A process where tasks are involved with too many work products may reveal that tasks are not specified with the appropriate granularity. A task that needs too many input work products may reveal an extremely complex task, and not only it would be difficult to accomplish as planned but also it should wait until all these work products are available before the process can proceed. Similarly, a task with too many output products may be too complex since its goal is not unique. This pattern can be seen in the TASK BLUEPRINT where we highlight wide nodes (too many output work products) in red, high nodes (too many input work products) in yellow, and large nodes (where both dimensions are big) in orange, as shown in Fig. 2. We consider a task to be too big if its hight or width are larger than one standard deviation from the average task hight or width, respectively.

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. This situation can be seen in the WORK PRODUCT BLUEPRINT where we highlight in orange 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 software process model, tasks and work products are related with edges indicating precedence. In the TASK BLUEPRINT, a task T1 precedes a task T2 if there is an output work product of T1 that is an input workproduct of T2. Similarly, in the WORK PRODUCT BLUEPRINT a work product WP1 predecedes another work product WP2 if there is a task where WP1 is an input and WP2 is an output. Considering that the process model specifies the way to proceed when working on one 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 misspecifications, when work products have not been specified as input or ouput artifacts for certain tasks when they should have been. We identify each subgraph with a different color in both, the TASK BLUEPRINT and the WORK PRODUCT BLUEPRINT, in order to identify the existence of independent subprojects.

3. Tool Support for Localizing Potential Errors

This section sketches the internals of the implementation of AVISPA. The scripts for implementing two of the error patterns are subsequently offered in Sect. 3.2. Finally, a description of the tool from the user point of view is provided in Sect. 3.3.

3.1. Implementation of AVISPA

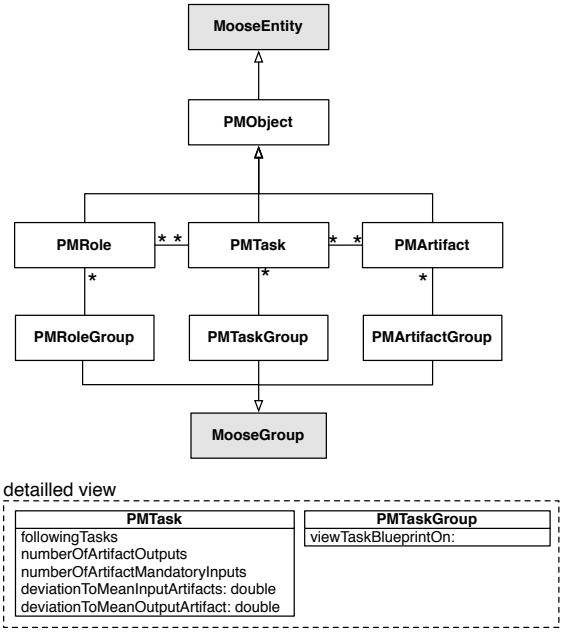

Figure 4. The AVISPA metamodel (gray classes belong to FAMIX)

The error pattern detection described in the previous section is implemented in AVISPA, an extension of MOOSE that deals with process models specified in SPEM 2.0. The FAMIX family of metamodels[3] has been augmented with the AVISPA metamodel as shown in Fig. 4 for describing and representing software processes. FAMIX has been extended by subclassing MooseEntity and MooseGroup. The name of the classes that belongs to AVISPA begin with PM. 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

[3] http://www.moosetechnology.org/docs/famix

offers methods for computing metrics and navigating through a model. For example, a 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.

Process Blueprints (Sect. 2.1) are visualized using the Mondrian visualization engine[4] [14]. 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 tasks involving too many work products. We provide the script for each of them, and the rationale in each implementation. The other error patterns are similar to these ones.

Independent subprojects. This kind of error can be seen when the Task Blueprint has disconnected subgraphs. Thus, each independent subgraph will be colored differently, and having a Task Blueprint with more than one color will mean 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 can be identified. Independent subproject always reveal an error in the process model specification.

```
PMTaskGroup>> viewTaskBlueprintOn: view
  | ds components orderedComponents normalizer |
  "Compute disjoint sets and assign a color to each set"
  cycleColor := ...

  "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
```

[4] http://www.moosetechnology.org/tools/mondrian

Fist 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 filling 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 artifacts, and the height shows the number of input artifacts. Arrows between two nodes exist if they are related with the *following* relationship. The whole blueprint is shown as a tree.

Tasks that involve too many work products. Here again the error will be seen in the Task Blueprint, but now the nodes that are larger than the standard deviation from the average number of input or output work products will be highlighted as potential errors. As usual, the standard deviation is the square root of the mean of the differences from each element's value to the mean value. Highlighted tasks reveal complexity in the process, but they are not always errors.

A series of metrics are precalculated so that the script can be executed. $numberOutputs_i$ and $numberInputs_i$ are the number of output and input 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 input and output artifacts for the whole process as follows:

$$MeanInArt = \frac{\sum_{i=1}^{n} numberInputs_i}{n} \quad (1)$$

$$MeanOutArt = \frac{\sum_{i=1}^{n} numberOutputs_i}{n} \quad (2)$$

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

$$sigmaIn = \sqrt{\frac{\sum_{i=1}^{n}(numberInputs_i - MeanInArt)^2}{n}} \quad (3)$$

$$sigmaOut = \sqrt{\frac{\sum_{i=1}^{n}(numberOutputs_i - MeanOutArt)^2}{n}} \quad (4)$$

Also, the distance to the mean value to both *MeanInArt* and *MeanOutArt* are calculated as follows:

$$distToMeanInArt_i = |MeanInArt - numberInputs_i| \quad (5)$$

$$distToMeanOutArt_i = |MeanOutArt - numberOutputs_i| \quad (6)$$

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

```
view shape rectangle
  fillColor: [:each |  ((each distToMeanInArt abs > self sigmaIn) &
        (each distToMeanOutArt abs > self sigmaOut))
        ifTrue: [Color orange]
        ifFalse: [(each distToMeanIn abs > (self sigmaIn)
        ifTrue: [Color yellow]
        ifFalse: [(each distToMeanOut abs > self 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 dimensions. If the distance to both mean values, the number of input and output artifacts, is larger than the standard deviation, then the node will be colored in orange. If only the distance of the number of output artifacts from the mean output artifacts is larger than the standard deviation, the node will be red. If only the distance of the number of input artifacts is larger than the standard deviation from the mean, then the node will be yellow. 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 input and output work products. Several orange tasks show a poor design. Yellow tasks have too many input work products, and even though it may suggest task complexity, it may not be too bad. On the other hand, red tasks, i.e. those with a lot of different output work products, are worse because the purpose of the task is not unique and it clearly suggests a poor design.

3.3. Avispa User Interface

Avispa has become a robust tool to import and visualize SPEM 2.0 based process models. It is built in 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 is ready to be analyzed. The navigation pane shows four entry points to begin an assessment: artifacts, roles and tasks. Navigation is realized through the information available in the metamodel (see Section 3.1). Although not depicted, metrics and other specific information (*e.g.,* description and annotations) are available under the *properties* tab.

[5] http://ww.pharo-project.org

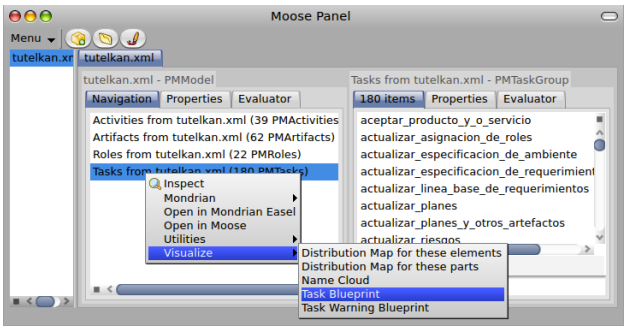

Figure 5. The Avispa main user interface

4. Applying Avispa in Real World Processes

The process model evaluation was carried on three small Chilean software companies: Amisoft, BBR Engineering and DTS. First, we briefly present the context in each company, and then we describe the results of applying Avispa to evaluate the software processes defined in each of them.

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 via TCP/IP and MQ Series. Amisoft has started its software process improvement project in 2009, and it is currently implementing the ISO9000:2000 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, a consulting company since 1994. It is formed by 24 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.

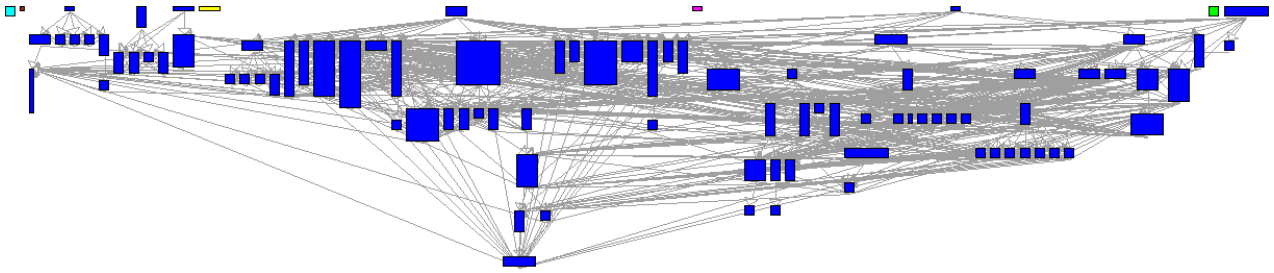

Figure 6. TASK BLUEPRINT **for localizing disconnected subgraphs in Amisoft.**

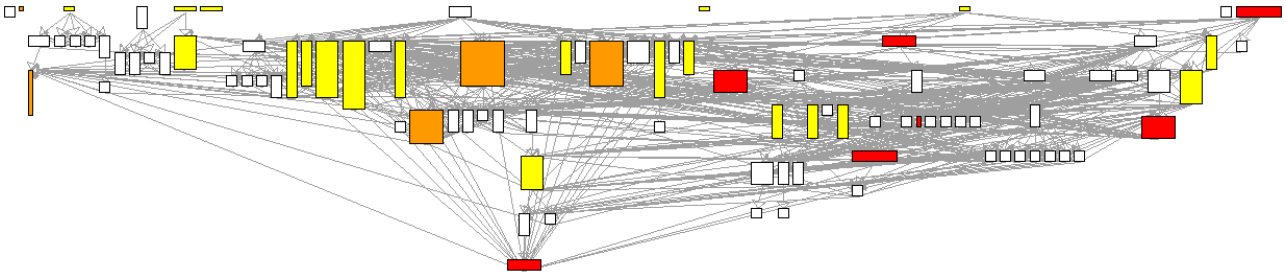

Figure 7. TASK BLUEPRINT **for localizing tasks involved with too many work products in Amisoft.**

DTS was born 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 counts on 250 employees, including engineers, certified technicians, operation workers and managers. DTS started to define its software process model in 2008, using the Rational Unifed Process as a reference. In DTS there is no specific software process improvement project; its effort has been oriented towards recovering the software process actually applied in the organization, in order to formalize it, evaluate it, and eventually improve it.

4.2. Tool Application

Amisoft. In this case, applying AVISPA for identifying disconnected subgraphs resulted in very few disconnected elements, even though it is clear that the tool easily help finding which they are, as shown in Fig. 6. AVISPA was also applied for identifying tasks that are involved with too many work products, as can be seen in Fig. 7; in this case the results were not that good. The process in Amisoft includes 93 tasks, 17 roles and 57 work products. The blueprints show a lot of problems in both patterns: 5 misspecifications (5.4%) and 32 potential errors of complexity of the tasks (34.4%). So, these results allow claiming that the process has an appropriate specification but, many tasks should be reviewed with respect to their complexity.

BBR Engineering. Similarly, we have applied AVISPA for finding disconnected graphs and complex tasks to the process model of BBR Engineering, and the results are shown in Figs. 8 and 9, respectively. The process in BBR includes 79 tasks, 18 roles and 42 work products. The blueprints show many problems according to both patterns: 23 potential errors of task complexity and 26 misspecifications. There is a large number of misspecifications corresponding to isolated tasks (35.4%). The 29.1% of the tasks had a complexity very far to the mean. So, the design of numerous tasks should be reviewed in detail.

DTS. AVISPA was also applied to the process of DTS for identifying and localizing both kinds of error patterns, and the output is shown in Figs. 10 and 11. The process in DTS includes 57 tasks, 9 roles and 66 work products. The results are far better than that of the other processes: in this case only 2 misspecifications were found (3.5%) and 15 potential complexity errors (26.3%). However the complexity of the tasks is a recurrent problem.

5. Related Work

In this paper, we start with a SPEM 2.0 software process model, that has already been formally specified

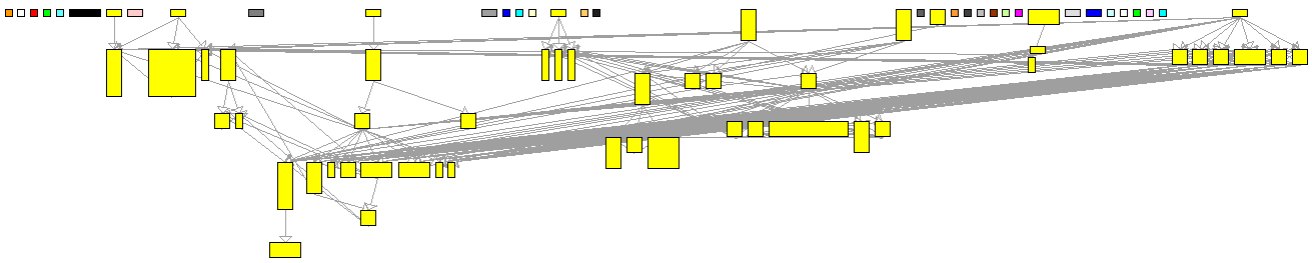

Figure 8. TASK BLUEPRINT **for localizing disconnected subgraphs in BBR Engineering.**

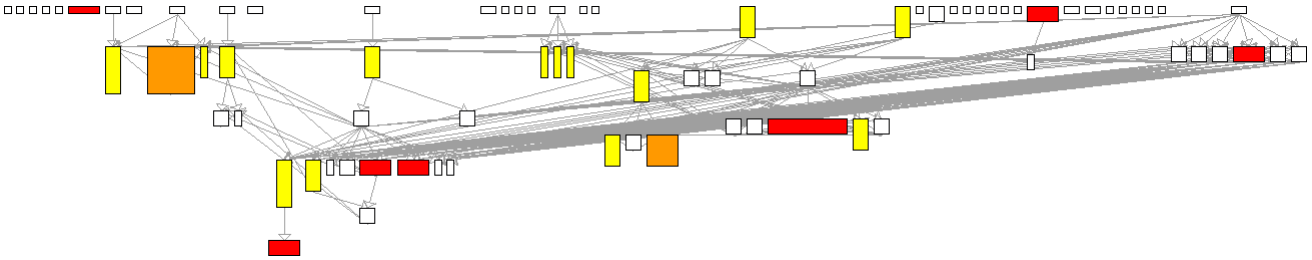

Figure 9. TASK BLUEPRINT **for localizing tasks involved with too many work products in BBR Engineering.**

following any strategy, and we apply AVISPA for evaluating the quality of the process itself and/or the specification as a technical review support tool, depending on the error pattern that we are able to find. Determining whether a software process is appropriate for a particular organization remains an open problem and we do not pretend this paper to solve it.

Software process model quality can be addressed with different perspectives: metrics, testing, simulation, or technical reviews. Canfora et.al. [2] define some ratio metrics for measuring overall software processes. However, based on this general data it is hard to know what is wrong, where the error is located, and how to find opportunities for improvement when there is no apparent error. In Process Model Testing, process models are checked against their specifications, similarly to software testing. An example of process testing is a software process assessment, where a process and its corresponding model are evaluated based on a capability maturity framework. This kind of approach is present in CMMI [17] and ISO/IEC15504 [10]. This kind of testing activities can only be carried out once the process model has already been implemented, tailored and enacted. This approach checks for the adherence to a standard but it does not evaluate the quality of the process model in itself. Gruhn [7] has proposed a verification technique based on simulation results and execution trace evaluation. Simulation has a shorter cycle, but it still requires enactment data. This is an appropriate verification technique if the process model is known to be correct, but if the model is incomplete, underspecified or its design is not appropriate, then the process model simulation will not reflect the expected results. Cook and Wolf [3] present a formal verification and validation technique for identifying and measuring the discrepancies between process models and actual executions. They do not address completeness or consistency of the process model. Pérez et al. [16] suggest to evaluate the congruence between the process model and a given environment based on past data. However, obtaining these measurements is hard and the results are not precise. Formal specifications and checking based on Petri Nets in a multi-view approach are presented in [6], but formal checking has semantic limitations.

As Osterweil has said [15], 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 [12] [5], so following a similar approach we have been able, based on a vast empirical experience, to identify a series of error patterns in software process models, to describe them in detail, and to formalize the way they look so that they can be automatically located.

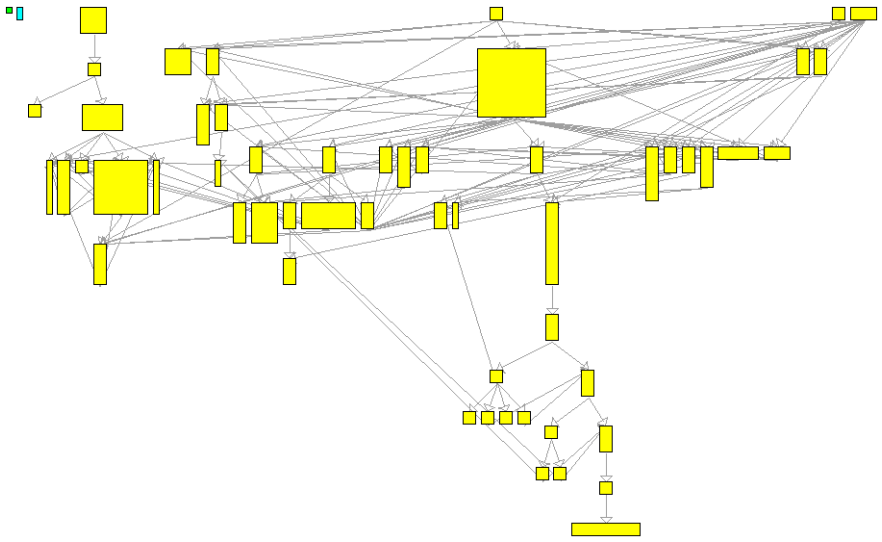

Figure 10. TASK BLUEPRINT **for localizing disconnected subgraphs in DTS.**

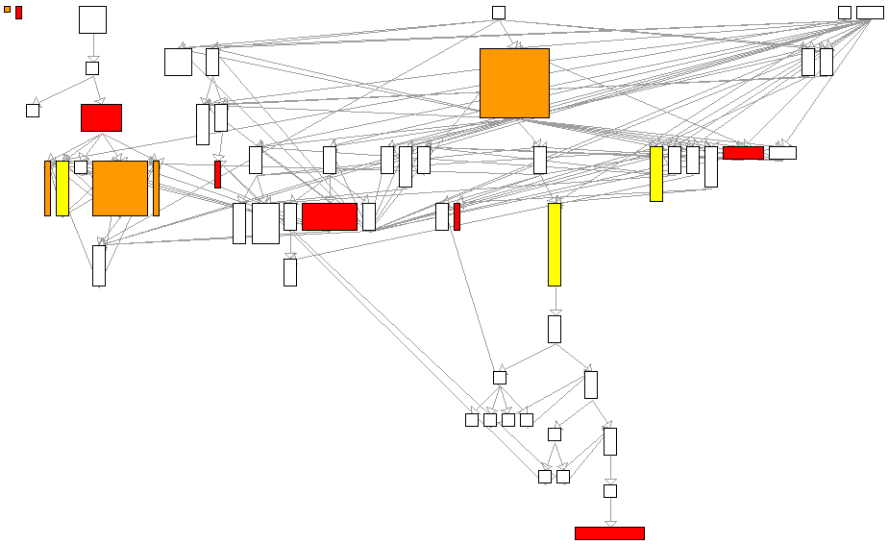

Figure 11. TASK BLUEPRINT **for localizing tasks involved with too many work products in DTS.**

The classical domain for software visualization is software source code. There has been a great deal of work on visualizing classes and methods [11], software architecture [13], 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 evaluation of process models obtained similar benefits to those achieved with software source code.

We have already presented process model blueprints in [9]. There, each blueprint is built following a reverse engineering approach as a model-driven strategy where the process model is the input to the script that acts as a transformation so that a new output model can be built presenting a partial view that may be more illustrative for finding errors. However, we realized that usability of process model blueprints was threatened when dealing with large and complex process models. AVISPA has improved usability by identifying a set of common error patterns, and highlighting them.

6. Conclusions

We have presented AVISPA, a tool for process model evaluation that follows a reverse engineering approach for localizing a set of identified potential errors. These errors may come either from process conceptualization or from misspecifications. We show how each of the error patterns identified can be seen in the appropriate process blueprint, and we made AVISPA highlight them. Some of the process models we were working with for the last years have been evaluated using AVISPA, and some errors were found, as well as some improvement opportunities, showing the effectiveness of the approach and the tool.

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. 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 in the process of 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.

Acknowledgments

The work of Julio A. Hurtado has been partially funded by a scholarship of NIC Chile.

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. Lagos, A. Bergel, and M. C. Bastarrica. Software Process Model Blueprints. In *Proceedings of the International Conference on Software Process, ICSP'2010*, volume 6195 of *LNCS*, pages 273–284. Springer-Verlag, July 2010.

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

[11] M. Lanza and S. Ducasse. Polymetric views—a lightweight visual approach to reverse engineering. *Transactions on Software Engineering (TSE)*, 29(9):782–795, Sept. 2003.

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

[13] M. Lungu and M. Lanza. Softwarenaut: Exploring hierarchical system decompositions. In *Proceedings of CSMR 2006 (10th European Conference on Software Maintenance and Reengineering)*, pages 351–354, Los Alamitos CA, 2006. IEEE Computer Society Press.

[14] 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.

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

[16] G. Pérez, K. El Emam, and N. Madhavji. Evaluating the congruence of a software process model in a given environment. In *Fourth International Conference on the Software Process*, pages 49–62, 1996.

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

[18] G. Valdés, H. Astudillo, M. Visconti, and C. López. The Tutelkán SPI Framework for Small Settings: A Methodology Transfer Vehicle. In *Proceedings of the 17^{th} European System & Software Process Improvement and Innovation*, Grenoble, France, September 2010. To Appear.

[19] 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.