JOURNAL OF SOFTWARE MAINTENANCE AND EVOLUTION: RESEARCH AND PRACTICE
J. Softw. Maint. Evol.: Res. Pract. 0000; **00**:1–16
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/smr

AVISPA: A Tool for Analyzing Software Process Models

Julio A. Hurtado Alegría*[1][2], María Cecilia Bastarrica[1], Alexandre Bergel[1]

[1]*Department of Computer Science (DCC), University of Chile, Chile*
[2]*IDIS Research Group, University of Cauca, Colombia*

SUMMARY

Defining and formalizing the software development process is a common means for improving it. Software process modeling is often a challenging and expensive endeavor, since a well specified process may still include inefficiencies that are hardly detected before enacting it. Thus, assessing process quality is a relevant concern in order to improve several aspects such as conceptual integrity, correctness, usability, maintainability, and performance, among others. This paper describes AVISPA, a graphical tool that allows analyzing the quality of SPEM 2.0 software processes models. AVISPA identifies a series of error patterns and highlights them in different blueprints. A detailed description of the internals of AVISPA is provided to show both its structure and its extensibility mechanisms. We also present an interactive mechanism to define new analysis scripts and to implement new patterns and blueprints. This paper illustrates the application of AVISPA in an industrial case study where process engineers are assisted to analyze the quality of their process.
Copyright © 0000 John Wiley & Sons, Ltd.

Received ...

1. INTRODUCTION

A software process is a set of partially ordered process steps, with related artifacts, human and computerized resources, organizational structures and constraints, intended to produce and maintain the requested software deliverables [18]. A well defined software process model is a determinant factor for achieving quality products and productive projects [10]. However, defining a software process model demands an enormous effort for making explicit common practices and defining practices that may not yet exist within the company. Standards such as ISO/IEC15504 and maturity models such as CMMI are commonly used as guidelines for defining this process. But there is still no standard wide-spread mechanism for determining the quality of a defined process, and thus the return-of-investment of software process definition is not always apparent.

We have previously proposed process model blueprints [13] for visualizing and analyzing different perspectives of a software process model. The three blueprints (ROLE BLUEPRINT, TASK BLUEPRINT, and WORK PRODUCT BLUEPRINT) are applied to software process models defined using SPEM 2.0 [20]. These blueprints enable the identification of *exceptional entities* [6], *i.e.,* exceptions in the quantitative data collected. Blueprints are successfully employed to identify flaws in industrial process models, but the process engineer knowledge still plays a key role in the analysis. We have assessed several industrial process models and we discovered a set of *recurrent errors* [14].

For the last six years we have worked with a number of small software companies in Chile to define their development processes in an effort to improve national industry standards. As part of this practical experience, we have identified some recurrent errors in software processes. Some of

*Correspondence to: E-mail: ahurtado@unicauca.edu.co

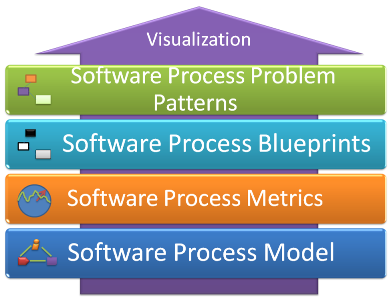

Visualization
Software Process Problem Patterns
Software Process Blueprints
Software Process Metrics
Software Process Model

Figure 1. Technology pile implementing AVISPA

them are due to conceptual errors in the process design and others are introduced during process specification. These errors are not easily identified and localized using traditional process modeling tools because of a possibly large amount of process elements involved, multiple views, and informal notations that may sometimes introduce ambiguity.

We have built AVISPA (Analysis and VIsualization for Software Process Assessment)[†], a tool that builds blueprints and highlights error patterns for a given process model. Error patterns are identified with process elements that are graphically "abnormally different" from the remaining elements [6]. Counting on this tool, the process engineer only needs to analyze highlighted elements, demanding little experience and also little previous knowledge for effective process model analysis, adding usability as well. Visualization is a well known approach to identify anomalies and errors that cannot be determined otherwise by letting software engineers use their ability to recognize cognitive patterns [16]. A large body of research use visual patterns to identify positive or negative properties of software systems [15, 23, 28]. However, none of the related work we are aware of elaborates on visual patterns for identifying problems in software process models. Figure 1 depicts the pile of technologies involved in our approach. Software process models specified in SPEM 2.0 are assumed to exist. On top of them we define a series of metrics that are used for identifying errors and improvement opportunities. Software process blueprints are visual representations of these metrics.

This paper presents a description of AVISPA detailing its blueprints and error patterns. It also provides a description of its internal structure and the means for extending it in terms of new blueprints or new error patterns. A discussion about the appropriate statistical precision of the patterns is included. We have analyzed the process model of a Chilean software company using AVISPA. In addition to our previous case studies, we have found several of the defined error patterns, and most of them resulted in actual errors. AVISPA has been highly welcomed in the company 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. Similar results were obtained in three other companies [14].

This article extends our previous work [14] by relating our experience, lesson learnt and providing some critical detail of our tool.

The rest of the paper is structured as follows. Section 2 presents a description of empirically found error patterns, their implications, and how they look in AVISPA. A detailed description of the structure of AVISPA is included in Section 3 including extension scenarios. Its application for localizing error patterns in an industrial software process model is reported in Section 4. Related work is discussed in Section 5, and some conclusions and further work are presented in Section 6.

[†]`http://www.moosetechnology.org/tools/ProcessModel`. AVISPA is free under the MIT license.

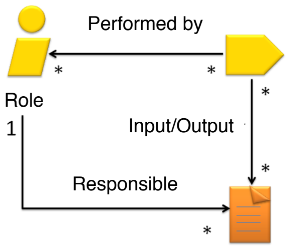


Figure 2. Conceptual Model of SPEM

2. SOFTWARE PROCESS MODELS, BLUEPRINTS AND ERROR PATTERNS

For the past six years we have conducted applied research in the area of software process models in small software companies in Chile [27] and Iberoamerica [24]. Along this work we have identified a number of common errors and problematic situations in software process model specifications, either due to misconceptions or misspecification. In this section we report a series of these patterns and how they may be identified with AVISPA. 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 is not a deliverable and neither a task nor a role needs.

2.1. Software Process Models

Modeling a software process refers to the definition of the process as a model [1]. A process model provides definitions of the process to be used, instantiated, enacted or executed. So, a process model can be analyzed, validated, simulated or executed if it is appropriately defined according to these goals. Some benefits of software process modeling are [5]: ease of understanding and communication, process management support and control, provision for automated orientation and process performance, provision for automated execution support and process improvement support.

SPEM 2.0 (Software and Systems Process Engineering Metamodel) [20] is a standard proposed by the OMG. It is based on MOF (Meta Object Facility) and it is the most popular language used to specify software processes. SPEM conceptual model, as depicted in Figure 2, allows identifying roles that represent a set of related skills, competencies, and responsibilities in the development team. Roles are responsible of specific types of work products. For creating and modifying work products, roles are assigned as performers in tasks where specific types of work products are consumed (inputs) and produced (outputs). Although the SPEM metamodel provides several other features, these three concepts together with guidances, are the key elements used by AVISPA.

2.2. Process Model Blueprints

ROLE BLUEPRINT, TASK BLUEPRINT and WORK PRODUCT BLUEPRINT are three graphical views of a software process model provided by AVISPA. Each of them focuses on a particular aspect of the process model, namely roles, tasks and work products, respectively. Each blueprint is depicted as a polymetric view [15].

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 between roles. Figure 3 shows an example 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 4 depicts a TASK BLUEPRINT. The color indicates the number of roles involved.

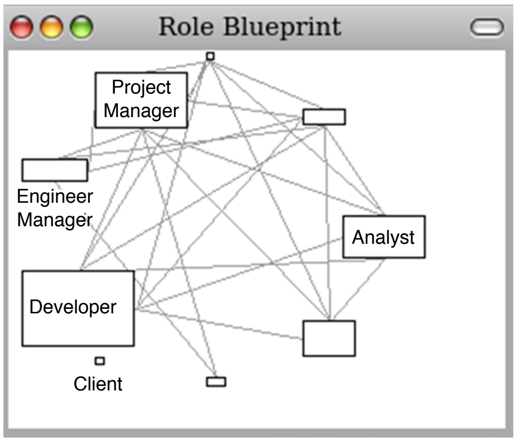


Figure 3. ROLE BLUEPRINT

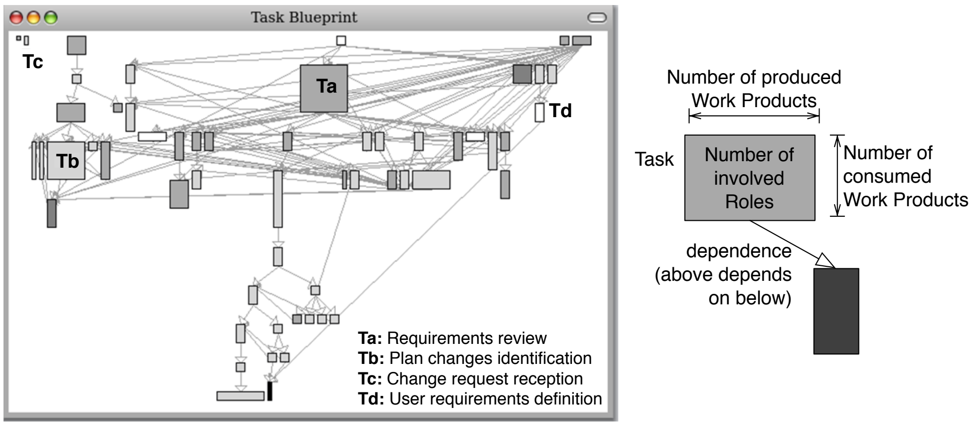

Task Blueprint
Tc
Ta
Td
Tb
Ta: Requirements review
Tb: Plan changes identification
Tc: Change request reception
Td: User requirements definition
Number of produced Work Products
Task
Number of involved Roles
Number of consumed Work Products
dependence (above depends on below)

Figure 4. TASK BLUEPRINT where potential errors are suggested but not localized

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.

2.3. *Error Patterns*

There is a number of anomalies in software process specifications we have realized that are recurrent. We call them error patterns. 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, in order to automate their location.

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 in certain task but appears to be left apart. This error pattern is apparent in the ROLE BLUEPRINT.

Independent subprojects. In a TASK BLUEPRINT and a WORK PRODUCT BLUEPRINT, tasks and work products are related with edges indicating precedence. The process model specifies the way to proceed when working on one unique project, so it is conceptually odd to have disconnected subgraphs in both blueprints. In general, these situations arise due to underspecifications, when work products have not been specified as input or output of certain tasks when they should have.

Table I. Error patterns identified by AVISPA

Error pattern	Description	Localization	Identification
Isolated role	A role that does not collaborate.	ROLE BLUEPRINT	A node that is not connected with an edge.
Independent subprojects	Independent subgraphs.	TASK BLUEPRINT or WORK PRODUCT BLUEPRINT	Subgraphs that are not connected with edges.
Overloaded role	A role involved in too many tasks.	ROLE BLUEPRINT	Nodes over one deviation larger than the mean.
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.
No guidance associated	An element with no guidance associated.	any blueprint	
Waste Work products	Independent subgraphs. Useless work products.	WORK PRODUCT BLUEPRINT	Work products that are neither deliverables nor input for any task.

Overloaded roles. If a role is involved in a large number of tasks, it becomes a risk: if it fails, all the associated tasks will fail as well. This is a possible error 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 shown in the ROLE BLUEPRINT.

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. 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. We consider a task to be too complex if it is wider than one standard deviation from the average task width.

Demanded work products. Work products required for a high number of tasks may cause serious bottlenecks when they are not available, and thus this pattern could reveal a misconception. This situation is seen in the WORK PRODUCT BLUEPRINT where we highlight nodes whose width is more than one standard deviation from the average.

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 executed. 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.

Waste Work products. Deliverables are those artifacts that need to be handed to the customer as part of the final product. Work products may be either deliverables or intermediate work products needed for coordinating successive tasks. However, if there are work products that are neither deliverables nor input for any 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 all those leaves that are not deliverables. The process engineer 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, or if they are actually waste in the process.

Table I summarizes the error patterns that have been identified so far. Isolated roles and Independent subprojects are always underspecifications. Overloaded roles, Multiple purpose tasks and Demanded work products are exceptional elements in the blueprints and they suggest possible conceptual errors. Roles, tasks and work products without guidance and Waste work products are not apparent in blueprints but still reveal improvement opportunities.

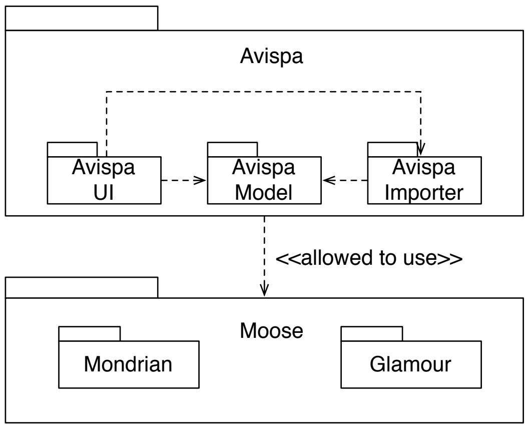


Figure 5. AVISPA Module View Type

3. THE AVISPA TOOL

This section describes the implementation of AVISPA including the technologies involved, its structure, the scripts for implementing two of the error patterns, and a scenario for extending the tool.

3.1. AVISPA Technologies

The expressiveness of AVISPA stems from combining meta-modelisation and visualization techniques. A process model, described in an XML file, is parsed and each structural element is reified to form an instance of our SPEM meta-model. We provide a set of metrics and navigation queries that may be computed on any SPEM model. The process structure and the result of the metrics are then graphically rendered. AVISPA is built at the top of the Moose software analysis platform[‡]. We have defined the SPEM metamodel as an extension of the FAMIX metamodel family[§]. Mondrian[¶] is an agile visualization engine to easily build polymetric views [15]. Mondrian operates on any arbitrary set of values and relations to visually render graphs. As exemplified in the coming sections, visualizations are specified with the Mondrian domain specific language.

AVISPA has become a reliable tool to import and visualize SPEM 2.0 based process models. It is built on top of Moose and the Pharo programming language[‖], and so it benefits from a large toolset for navigation and visualization. Its navigation panel shows four entry points to begin an analysis: activities, artifacts, roles and tasks. Navigation is realized through the information available in the metamodel. Metrics and other specific information (*e.g.,* descriptions and annotations) are also available.

3.2. AVISPA Architecture

From an architectural point of view, AVISPA follows a layered pattern, as shown in Figure 5. The *AvispaImporter* module has the responsibility of creating an AVISPA model from an XML model provided by EPF. The *AvispaModel* module has the responsibility of representing process blueprints and calculating programmed metrics. *AvispaUI* includes navigation and visualization scripts. These functionalities are illustrated in Figure 6. Importers, models, model elements, navigation and visualization modules have been implemented and integrated using abstract solutions of the Moose technology.

[‡]`http://www.moosetechnology.org`
[§]`http://www.moosetechnology.org/docs/famix`
[¶]`http://www.moosetechnology.org/tools/mondrian`
[‖]`http://www.pharo-project.org`

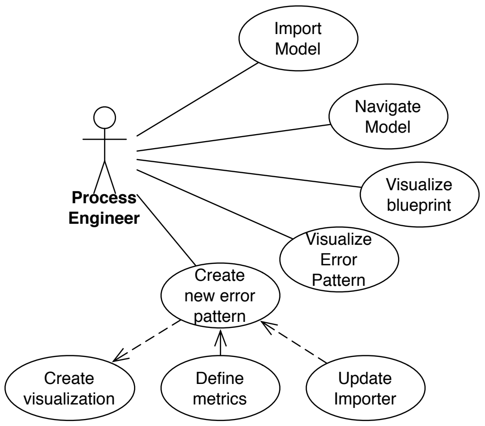


Figure 6. AVISPA Use Case Diagram

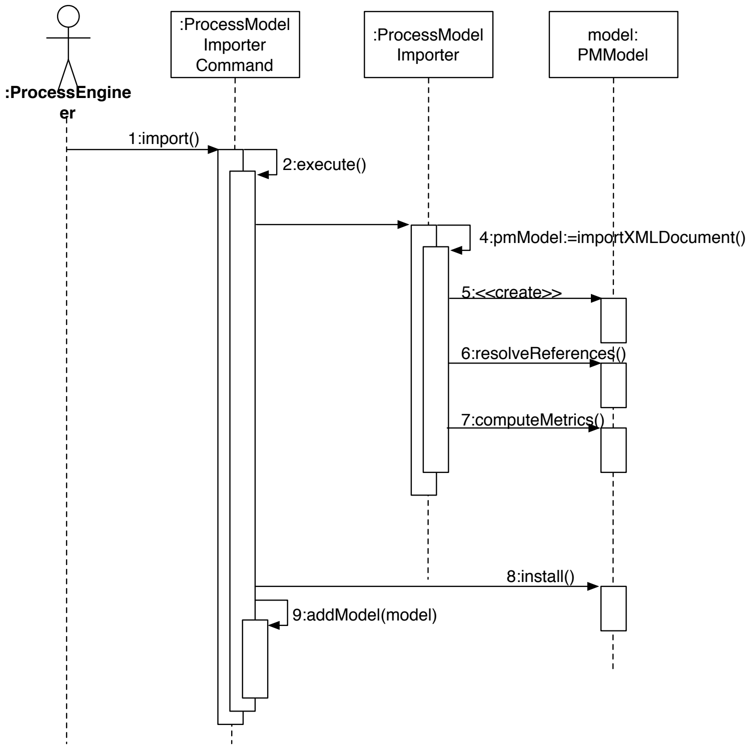
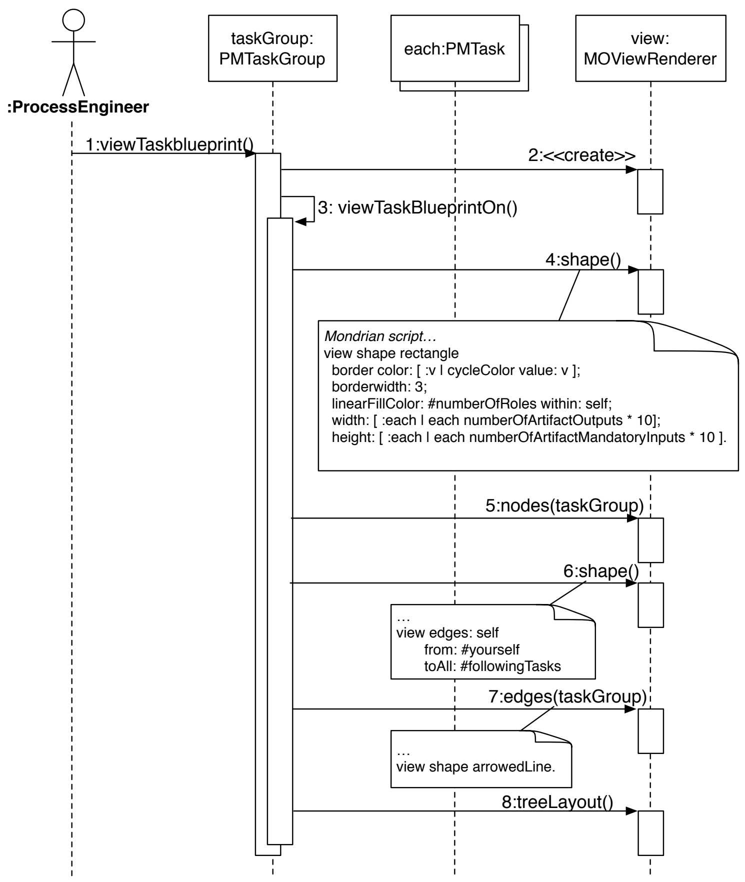


Figure 7. Importing an EPF Process in AVISPA

Figure 7 shows how a process model is imported. It starts from a *MoosePanel* object, where the import event is caught by a *ProcessModelImporterCommand* object and it gets the process model sending the *importXML* message to the a *ProcessImporter* object. In the *importXML* of the *ProcessImporter* object, each process element is imported by using the class method *fromXMLDescription* and stored in collections which are used for building the process model. Then the process model internal relationships are resolved using the *resolveReferences* method and the metrics are initialized via *initializeMetrics* method. Metrics have been programmed in each process model element. Finally, the model is installed in the Moose platform and the model is added to the *MoosePanel* using the *ProcessModelImporterCommand* object.



```
Mondrian script...
view shape rectangle
  border color: [ :v | cycleColor value: v ];
  borderwidth: 3;
  linearFillColor: #numberOfRoles within: self;
  width: [ :each | each numberOfArtifactOutputs * 10];
  height: [ :each | each numberOfArtifactMandatoryInputs * 10 ].
```

```
...
view edges: self
      from: #yourself
      toAll: #followingTasks
```

```
...
view shape arrowedLine.
```

Figure 8. Visualization of the Independent Error Pattern in AVISPA

Figure 8 shows how a visualization is performed. In this case, the visualization is about the independent project error pattern applied to the TASK BLUEPRINT. The sequence starts when the user, located on the PMTaskGroup selects the visualize option and selects *viewTaskBlueprint*. The PMTaskGroup object creates a view (of type MOViewRenderer) and performs the *viewTaskBlueprintOn* method. This method uses the view previously created to generate the blueprint and to highlight the problematic cases using a Mondrian script.

The partial class UML diagram in Figure 9 summarizes and localizes the behavior in the respective classes according to the described scenarios. It also defines the main class relationships in AVISPA. Gray classes in Figure 9 belong to the Moose platform. The class *ProcessModelImporter* is an extension of *ProcessModelImporterCommand* class and includes the import methods; the class *PMModel* is an extension of the *ProcessModel* class and it contains collections of PMTasks, PMArtefacts and PMRoles. The class PMObject and its derived *PMTask*, *PMArtefact* and *PMRole* classes include the respective element information and they are responsible for calculating the respective metrics and for implementing the navigation features. These elements are grouped by the respective *MooseGroup*. This grouping facilitates visualizing features.

3.3. Error Pattern Implementation in AVISPA

We illustrate the implementation of two error patterns: independent projects and multiple purpose tasks. 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.

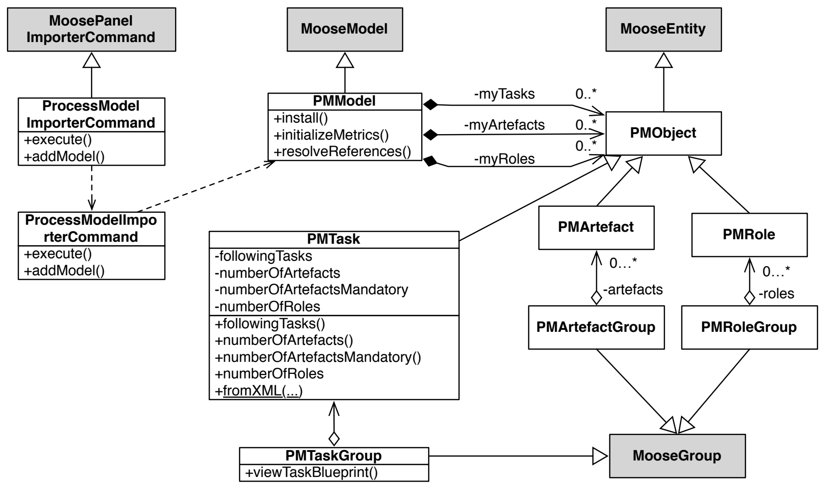


Figure 9. Partial Class Diagram of AVISPA AVISPA

Independent subprojects. In this kind of error each independent subgraph is colored differently in AVISPA, 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 means that there are no independent subprojects, and the process is fine with respect to this error pattern. The following script builds a colored TASK BLUEPRINT where independent subprojects are identified. Independent subprojects always denote errors in the process model specification.

```
PMTaskGroup>> viewTaskBlueprintOn: view
  | ds components orderedComponents normalizer cycleColor |
  "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: [:tasks :index |roles do: [:task |orderedComponents at: task 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
```

Firstly, a cycle is computed so that edges of connected subgraphs are painted with the same color. Secondly, individual nodes are built assigning them a size and a color. Tasks are represented as rectangular nodes whose color is that of the subgraph they belong to. Finally, the blueprint is displayed.

Multiple Purpose Tasks. Nodes in the TASK BLUEPRINT that are wider than one standard deviation from the average number of output work products are highlighted to indicate a potential error. One standard deviation in a normal distribution function was the empirical value calibrated from the F-measure (Section 4.3). A series 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, we can compute the mean number of output work products for the whole process, the standard deviation and the distance to the mean as follows:

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

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

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

These metrics are used as part of the script for identifying multiple purpose tasks in order to determine the color of each node in the TASK BLUEPRINT.

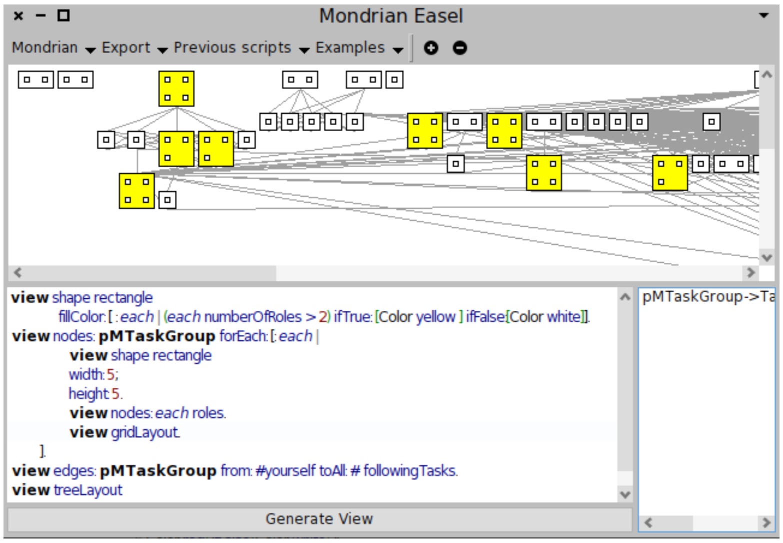
```
"Assign a color to each node"
viewTaskWarningBlueprintOn: view
view shape rectangle
  fillColor: [:each | (each distToMeanOutWP > self myModel sigmaOut)
        ifTrue: [ Color red ]
        ifFalse:[ Color white ] ];
  "Build the TASK BLUEPRINT"
  borderColor: Color black;
  width: [ :each | each numberOutputs * 10];
  height: [ :each | each numberInputs * 10].
view nodes: self.
"Display the blueprint"
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. If the distance from the number of output work products to the mean is larger than one standard deviation, then the node is painted in red and otherwise white. Obtaining a TASK BLUEPRINT with several red tasks clearly suggests a poor design.

3.4. *Extending* AVISPA

AVISPA is extensible in various ways. The easiest way is to use the interactive views of Mondrian Easel, where the group of elements is opened and a scripting window is displayed, new scripts are written and a view is generated using only the script and the visualized group. The process engineer can explore other polymetric views using new layouts, nodes, metric combinations and edges. More complex extensions can be introduced to AVISPA's object-oriented code considering simple hot spots provided by Moose. A new visualization previously tested using Mondrian Easel can be implemented adding some methods in the respective *MooseGroup*; if the new visualization requires a new metric, attributes and methods must be added to the respective *PMObject*. The most complicated scenario occurs when the visualization requires new process elements not currently included in AVISPA. In this case, a new class derived from *PMObject* must be defined including a *fromXML* method, and the *resolveReferences* method of *PMModel* must be updated.

3.4.1. Defining a new visualization. Defining a new visualization is done in three steps: a Mondrian Easel window is displayed, a view is programmed and generated using the Mondrian script language, and the visualization is integrated into AVISPA's code. The following scenario illustrates it: *the process engineer needs to define some technical requirements to support collaboration*. Therefore, it is useful to know which tasks require more collaboration support. The number of roles involved in each task is used to identify the tasks with more collaboration. If the process engineer considers that



```
view shape rectangle
        fillColor: [ :each | (each numberOfRoles > 2) ifTrue: [Color yellow ] ifFalse[Color white]].
view nodes: pMTaskGroup forEach: [:each |
            view shape rectangle
            width: 5;
            height 5.
            view nodes: each roles.
            view gridLayout.
    ].
view edges: pMTaskGroup from: #yourself toAll: # followingTasks.
view treeLayout
```

Figure 10. Extending AVISPA interactively with Mondrian Easel

a task with more than two roles involved should be identified, then this reference point could be used to define a visualization script in Mondrian Easel. In the script presented and visualized in Figure 10 collaborative tasks have been highlighted in yellow, and the subview shows the roles involved.

Once the new pattern script has been tested using the generate view button, its code has to be added to the AVISPA tool code as a new pattern. In this case the pattern does not refer to an error, instead the pattern refers to useful information to introduce collaboration practices. To include in the AVISPA code this functionality two new methods must be added: *viewCollaborativeTaskPatternOn* and *viewCollaborativeTaskPattern*. The former is the same Mondrian Easel script, but replacing the reference to the PMTaskGroup object by self reference because the message is received by the *PMTaskGroup* in the navigator.

```
viewCollaborativeTaskPattern
  <menuItem: 'Collaborative Task Pattern' category: 'Visualize'>
  | view |
  view := MOViewRenderer title: 'Collaborative Task Pattern'.
  self viewCollaborativeTaskPatternOn: view.
  view open
```

3.4.2. Importing a new Process Element. AVISPA uses a small subset of SPEM 2.0. If the process engineer wants to add another concept in AVISPA, a class of this new concept must be created inheriting the class *PMObject* implementing the *fromXMLDescription* class method. Additionally a group must be created inheriting the class *MooseGroup*. For instance, to add the SPEM Activity to AVISPA, a *PMActivity* class must be added as follows:

```
  PMModelObject subclass: #PMActivity
        instanceVariableNames: 'task predecessors'
        classVariableNames: ''
        poolDictionaries: ''
        category: 'ProcessModel-Core'
```

The *fromXMLDescription* method can be programmed reusing the general importing functionality and adding new attributes, in this case the performer primary id:

```
  fromXMLDescription: xmlElement
        | answer |
        answer := super fromXMLDescription: xmlElement.
        answer performerPrimaryId: (xmlElement attributeAt:#PerformedPrimaryBy).
        ˆ answer
```

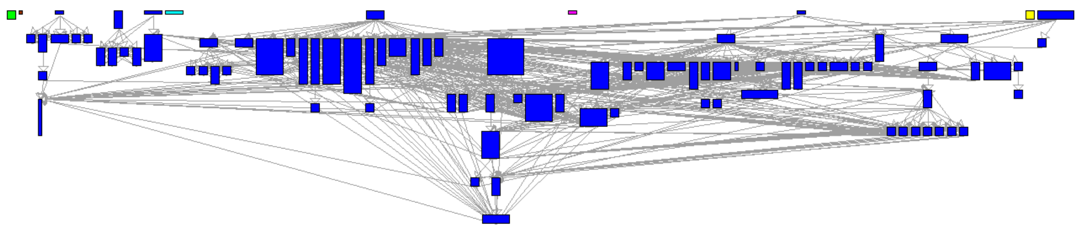


Figure 11. TASK BLUEPRINT for localizing disconnected subgraphs in Amisoft.

If the new element includes references, then the method that resolves references of the process model must be modified adding these references. Additionally, navigation methods are created in the method navigation category as follows:

```
predecessorsTasks
        < navigation: 'Predecessors Tasks' >
        ˆ PMActivityGroup withAll: self predecessors
```

3.4.3. Defining new Metrics. New metrics are obtained from numerical attributes defined in different process elements, e.g., the number of predecessor elements of an activity is available to the different blueprints. This metric is added to the class PMActivity in the metric method category as follows:

```
numberOfOutputTasks
   <MSEProperty: #NOTK type: #Number>
   <property: #NOPre longName: 'Number of predecessors activities'>
   ˆ self predecessors size
```

4. APPLYING AVISPA IN A REAL WORLD SOFTWARE PROCESS MODEL

AVISPA was applied on Amisoft, a Chilean software company. First, we briefly present the context of the company and then we describe the results of applying AVISPA to analyze its process model. We illustrate the application of AVISPA with two error patterns: independent projects in the TASK BLUEPRINT and tasks with too many output work products. Independent projects is also analyzed in the WORK PRODUCT BLUEPRINT. The process model used in this research was developed in the last two years and was obtained from the company's library using the exporter feature of EPF. The process model was analyzed and then the results were discussed with the process engineers.

4.1. Application Scenario

Amisoft is around ten years old and it employs thirty qualified employees. Its main goal is 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 has been inspired by OpenUP.

4.2. Process Model Analysis

AVISPA helped to identifying 5 instances of the pattern *independent subprojects* (in this case disconnected tasks) corresponding to the nodes with a color different from blue in Figure 11. 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 represent a high risk because the configuration management process could be chaotic and the testing of interfaces and communications could be forgotten just when it is required the most.

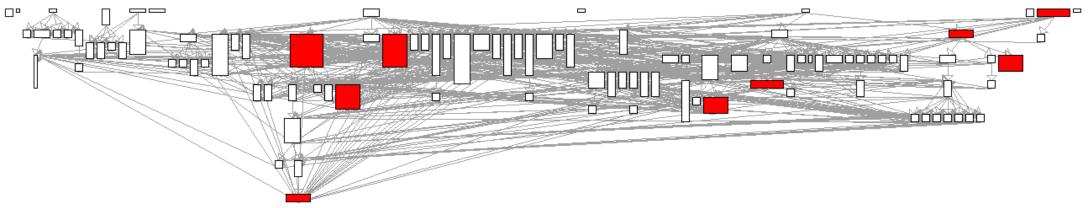


Figure 12. TASK BLUEPRINT for localizing multiple purpose tasks in Amisoft.

The *multiple purpose task* pattern was also applied (red nodes in Figure 12). 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 actually required to evaluate the project advance or to make some decisions, and as a 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 it really is, because the measurement results are not available directly in a unique work product but in many work products.

4.3. Pattern Tuning

The F-Measure [25] has been used to evaluate the effectiveness of the patterns because AVISPA recovers problematic elements in software process models but not all are actual errors. The F-Measure is obtained identifying two types of sets: recovered potential problems (*retrievedElements*) and the actual relevant problems (*relevantElements*). The recall (R - fraction of the process elements that are relevant to the query that are successfully retrieved) and precision (P - the fraction of retrieved process elements that are relevant to the search) metrics are calculated according to the formulas:

$$R = \frac{\|\{relevantElements\} \cap \{retrievedElements\}\|}{\|\{relevantElements\}\|} \tag{4}$$

$$P = \frac{\|\{relevantElements\} \cap \{retrievedElements\}\|}{\|\{retrievedElements\}\|} \tag{5}$$

$$FMeasure = 2 * \frac{P * R}{P + R} \tag{6}$$

The analysis is focused on the applied patterns: independent projects and multipurpose tasks. The first pattern is very effective because most of the identified problems were actual problems, so its F-Measure is of 0.9. The second pattern is not as effective as the first one. So, it was tuned to achieve the largest benefit using as reference unit a standard deviation (sigma) and three cases were analyzed: using 1-sigma, 2-sigma and 3-sigma as problem identification filters. The values obtained were: 0.50 for 1-sigma, 0.17 for 2-sigma, and 0.33 for 3-sigma. So, the multipurpose task pattern was tuned to 1-sigma: each task positively deviated 1-sigma from the average number of outputs will be identified as a potential error of multipurpose task.

5. RELATED WORK

Software process model quality can be addressed from different approaches: metrics [3], testing, simulation [9], or formal verification [8]. 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 [4] 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 [29]. 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 stage, based on reviewing the architectural views of a software process model defined as Software Process Blueprints [13]. 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 to 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 specialized knowledge of an expert software process engineer for identifying improvement opportunities and thus requiring less experienced process engineers.

As stated by Osterweil [21], 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 [7, 17], 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 [15], software architecture [19], and even source code annotations [2]. 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 [22] 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 [12] 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. Soto et. al. [26] 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.

6. CONCLUSIONS

In this paper we have presented AVISPA, a tool for process model analysis that localizes a set of identified error patterns within a process model specified in SPEM 2.0. These errors may come either from process conceptualization or from misspecification. We described how each of the error patterns identified are found in the appropriate blueprint, and we made AVISPA highlight them. Also some extension scenarios have been described showing the extension capability of AVISPA.

The process model of a Chilean company we have been working with for two years has 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 were real improvement opportunities.

As part of the application experience we have realized that 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 are effective in finding improvement opportunities, there is some room for fine tuning them. The F-Measure was applied to show that one standard deviation was effectively the best option for the multipurpose task error pattern.

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.

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 NIC Chile. This work has been partly funded by Program U-INICIA 11/06 VID 2011, grant U-INICIA 11/06, University of Chile, and FONDECYT 1120094.

REFERENCES

1. S. T. Acuña and X. Ferré. Software process modelling. In *World Multiconference on Systemics, Cybernetics and Informatics, ISAS-SCIs 2001, July 22-25, 2001, Orlando, Florida, USA, Proceedings, Volume I: Information Systems Development*, pages 237–242, 2001.
2. 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.
3. 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.
4. 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.
5. B. Curtis, M. I. Kellner, and J. Over. Process modeling. *Communications of ACM*, 35(9):75–90, 1992.
6. S. Demeyer, S. Ducasse, and O. Nierstrasz. *Object-Oriented Reengineering Patterns*. Square Bracket Associates, 2008.
7. 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.
8. J. Ge, H. Hu, Q. Gu, and J. Lu. Modeling Multi-View Software Process with Object Petri Nets. *ICSEA 2006*, 0:41, 2006.
9. V. Gruhn. Validation and verification of software process models. In *Proc. of the Software development environments and CASE technology*, pages 271–286, 1991.
10. W. S. Humphrey. *Managing the software process*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.
11. J. A. Hurtado. *A Meta-process for Defining Adaptable Software Processes*. PhD thesis, Computer Science Department, Universidad de Chile, 2012.
12. 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.
13. 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.
14. J. A. Hurtado Alegría, M. C. Bastarrica, and A. Bergel. Analyzing software process models with avispa. In *Proceedings of the 2011 International Conference on Software and Systems Process*, ICSSP '11, pages 23–32, New York, NY, USA, 2011. ACM.
15. M. Lanza and S. Ducasse. Polymetric Views-A Lightweight Visual Approach to Reverse Engineering. *Transactions on Software Engineering*, 29(9):782–795, Sept. 2003.
16. H. Lieberman and C. Fry. ZStep 95: A reversible, animated source code stepper. In J. Stasko, J. Domingue, M. H. Brown, and B. A. Price, editors, *Software Visualization — Programming as a Multimedia Experience*, pages 277–292, Cambridge, MA-London, 1998. The MIT Press.
17. B. Livshits and T. Zimmermann. Dynamine: finding common error patterns by mining software revision histories. *SIGSOFT Softw. Eng. Notes*, 30(5):296–305, 2005.
18. J. Lonchamp. A Structured Conceptual and Terminological Framework for Software Process Engineering. *Second International Conference on the Continuous Software Process Improvement*, pages 41–53, 1993.
19. M. Lungu and M. Lanza. Softwarenaut: Exploring Hierarchical System Decompositions. In *Proceedings of the 10^{th} European Conference on Software Maintenance and Reengineering)*, pages 351–354, Los Alamitos CA, 2006. IEEE Computer Society Press.
20. OMG. Software Process Engineering Metamodel SPEM 2.0 OMG Specification. Technical Report ptc/07-11-01, OMG, 2008.
21. L. J. Osterweil. Software Processes Are Software Too. In *International Conference on Software Engineering*, pages 2–13, 1987.

22. 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.
23. 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.
24. 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.
25. C. J. V. Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, Newton, MA, USA, 2nd edition, 1979.
26. 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.
27. 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 European System & Software Process Improvement and Innovation*, Grenoble, France, September 2010.
28. R. Wettel, M. Lanza, and R. Robbes. Software systems as cities: a controlled experiment. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 551–560, New York, NY, USA, 2011. ACM.
29. 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.