Software Process Model Blueprints*

Julio Ariel Hurtado Alegría[1,2], Alejandro Lagos[1], Alexandre Bergel[1], and María Cecilia Bastarrica[1]

[1] Computer Science Department, Universidad de Chile, Chile
[2] IDIS Research Group, University of Cauca, Colombia
{jhurtado,alagos,abergel,cecilia}@dcc.uchile.cl

Abstract. Explicitly defining a software process model is widely recognized as a good software engineering practice. However, having a defined process does not necessarily mean that this process is good, sound and/or useful. There have been several approaches for software process evaluation including testing, simulation and metrics; the first one requires software process enactment, i.e., an expensive, risky and long process, and the others require high expertise for correctly interpreting their meaning. In this paper we propose a visual approach for software process model evaluation based on three architectural view types, each one focusing on basic process elements: Role Blueprint, Task Blueprint and Work Product Blueprint. They enable visual evaluation of different perspectives of a software process, each being relevant for a particular stakeholder. We illustrate the proposed approach by applying it to the software process defined for a real world company that develops software for retail. We show how design errors were identified.

1 Introduction

There is a generalized agreement among software practitioners about the relevance of counting on a well defined process model for systematizing development. There have been several efforts in aiding organizations toward this goal: maturity models and standards that describe which elements should be part of a software process, notations that allow rigorous specification, and tools that support these notations. However, having a well defined software process does not necessarily mean having a good process.

It is not apparent how to determine if a software process is good, or if it can be improved in any sense before it is enacted [16]. The software process metamodel SPEM 2.0 [15] proposes wellformedness rules for software process definition but their scope is limited. For example, SPEM 2.0 does not determine if for a given process some of the roles are overloaded, if there are work products that are bottlenecks, or if there is no clear termination condition for a task cycle. What is even more difficult about these issues is that they do not always constitute errors, but they are indicators that something may not be right [10]. Even though there are some metrics defined for measuring software processes [2], they provide

* The work of Julio Hurtado Alegría has been partly funded by NIC Chile.

little intuition about what may be wrong, where the error is, and how to find opportunities for improvement when there is no apparent error. How can we present a rich software process on mere flat diagrams? Software processes should represent a complex, dynamic and multidimensional world. Moreover, available tools are usually intended only for software process visual specification, and not for evaluation, let alone visual evaluation.

The convenience of specifying the software architecture with multiple views has been agreed upon [3]. Different stakeholders require different information for evaluations, thus it seems natural to deal with multiple process architectural views [11]. We specify our process models using the SPEM 2.0 standard. Process architectural views are built following an architectural recovery approach such that they allow us to evaluate different process characteristics. We propose three process architectural view types: Role Blueprint, Task Blueprint and Work Product Blueprint. These blueprints are process views that make use of metrics computed from a process model for easing understanding and evaluating different aspects not available in SPEM 2.0 and its associated tools. These visualizations have been implemented in the ProcessModel tool[3] based on Moose technology and the Mondrian tool. Visualization aids in data analysis and problem detection [14]. Using different concerns about process models in a graphical manner, we make use of the human visual capacity for interpreting the process in context and to evaluate its coherence and suitability [13].

We have applied our approach to the process model specification of DTS, a real world small company that develops software for retail. We have been able to identify several problems in the process and the company's stakeholders agreed that they were actual problems. The process has been redefined and the changes are currently being applied.

2 Related Work

Software quality is a key issue in software engineering, and it highly depends on the process used for developing it [8]. Software process quality assurance can be addressed in different ways: metrics, testing, simulation, or formal reviews.

Canfora et. al. [2] define some ratio metrics for measuring overall software processes, but 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. However, these metrics are a key starting point. We use them as a basis over which we build a visual layer to evaluate more specific aspects of different process model elements. We define a evaluation approach based on visualization whereas Canfora et. al. propose and validate some general metrics. Our approach is based on the evidence about the visualization aids in data analysis and problem detection [14]. Further we visualize different concerns about process models using the human visual capacity for interpreting the process in context and to evaluate its coherence and suitability [13].

[3] Freely available at `http://www.moosetechnology.org/tools/ProcessModel`.

In Process Model Testing, process models are checked against their specifications [16]. 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 approach is present in CMMI [19] and ISO/IEC15504 [9]. This kind of testing activities can only be carried out once the process model has already been implemented, tailored and enacted. It checks for adherence to a standard but it does not evaluate the appropriateness of the process for the organizational or project needs. Process testing requires very long time cycles. 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 suitable for the environment, but if the model is incomplete, underspecified or it is not suitable for the organization, the process model simulation will not yield the expected results.

Cook and Wolf [4] present a formal verification and validation technique for identifying and measuring the discrepancies between process models and actual executions. They do not address suitability, completeness or consistency of the process model. Pérez et al. [17] 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 [5], but formal checking has semantic limitations. Our process model blueprints use SPEM 2.0 models to recover and visualize them independently using some basic metrics, and the evaluation can be delegated to process designers in an informal but structured and practical way.

Software inspections, walkthroughs, and technical reviews, are widely used to improve software quality, so they are promising techniques for software processes too. The key idea is that a group of experienced software engineers, meeting together to review a software product, can improve its quality by detecting defects that the product designer or developer could have overseen. Our proposal consists of a tool that supports the software process model review based on visual mechanisms. According to [18], the review process has three main tasks: defect discovery, defect collection, and defect discrimination. The defect discovery is the one that demands the highest expertise. The approach and tool presented in this paper is intended to facilitate the process model defect discovery in a visual way inside a planned verification and validation context. Also, the specific solution presented in this paper can be useful for a process designer to gain some insights about the process model design and potentially find some problems.

3 Problems in Software Process Model Evaluation

Defining a software development process is an effort to systematize and improve software development. However, defining a software process does not necessarily imply that the process is complete, sound and/or well defined.

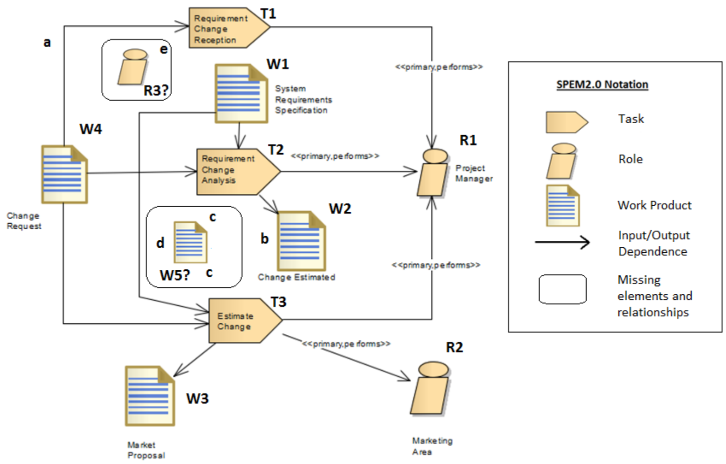

Fig. 1. SPEM Model Fragment for Requirements Change Management

SPEM is a specific standard language used to define software process models. Enterprise Architect (EA)[4] and Eclipse Process Framework (EPF)[5] are two popular integrated development environments (IDE) used to describe and visualize SPEM processes. We argue that the representations provided by these tools are not enough for easily assessing model quality. As an illustration, we will use the process model used in DTS, a small software company.

The software process model used by DTS is written in SPEM and it is composed of 57 tasks, 66 work products and 9 roles. We consider the complete process, including project management, testing, requirements management and technical development. Figure 1 is a screenshot of a piece of the DTS process model visualized using EA. This model excerpt focuses on part of the requirements change management area and represents the main dependencies between tasks and their related roles and work products. The process model adopted at DTS received a careful attention in its conception by dedicated engineers. However, a number of problems were recently detected but not easily identified.

Incorrect relationships (a,b): W4 (Change Request) should be an output of T1 (Requirement Change Reception) instead of being an input, and W2 (Change Estimated) should be an output of T3 (Estimate Change) instead of T2 (Requirement Change Analysis). The change request is received by task *Requirement Change Reception* and is used by all other tasks. The *Change Estimated* work product is the result of the *Estimate Change* task.

Missing elements (d,e): W5 (Requirement Change Analysis) is required as output of T2 (Requirement Change Analysis) and input of T3 (Estimate Change). A new role R3 (*Client*) is required because it is involved in T1 and T2 (Re-

[4] `http://www.sparxsystems.com.au`
[5] `http://www.eclipse.org/epf`

quirements Change Reception and Analysis). W5 and R3 are missing because the process has been inadequately designed.

Missing relationships (c): control links from work product W5 (Requirement Change Analysis) to task T2 (Requirement Change Analysis) as output and to task T3 (Estimate Change) as input are missing. Without W5 between T2 and T3, the information flow remains underspecified.

These anomalies in the DTS process were not easily identified mainly due to scalability, complexity and availability issues. Figure 1 depicts only a small portion of the complete process. The complete process is multi screen which makes it difficult to analyze. Many different sources of information are gathered in the picture obtained from EA. As a consequence, checking the validity of the model is often perceived as a tedious activity. A global picture can be obtained through a simple composition within the environment, but this composition is not automatically built in a global fashion by SPEM modeling tools.

Although we conducted our analysis on EA and EPF, the same actions could have been made in other tools (including SPEM as UML Profile). To tackle these issues, we propose to complement the process designer's activities with a number of extra visualizations. Simulation is not a practical option because SPEM 2.0 neither provides concepts nor formalisms for executing process models [1].

4 Multiple Software Process Model Blueprints

There is no unique perfect view to visually render a software process model [10]. As for most engineering activities, defining a robust, efficient and multi-view software process is a non-trivial activity that requires flexible and expressive tools. As a complement to software process design tools, we propose a visual approach to evaluate some quality criteria. We introduce the notion of blueprint as a partial but focused graphical representation of a software process model.

4.1 Process Model Blueprints in a Nutshell

Process model blueprints are graphical representations meant to help software process designers to (i) assess the quality of software process models and (ii) identify anomalies in a model and provide hints on how to fix them. The essence of these blueprints is to facilitate the comparison between elements for a given selected domain using a graph metaphor, composed of nodes and edges.

The size of a node or an edge tells us about their relative importance. In the case that a node is “much larger” than others, this should draw the attention of the process designer because it may reveal an anomaly or a misconception. We identified three blueprints that help identifying opportunities for improving software process models. Each of them focuses on a particular domain, namely *roles*, *tasks* and *work products*. Table 1 summarizes the meaning of boxes, edges and dimensions for each of these blueprints.

The visualizations we propose are based on *polymetric views* [12]. A *polymetric view* is a lightweight software visualization technique enriched with software

	Role Blueprint	***Task Blueprint***	***Work Product Blueprint***
Layout	Circle layout	Tree ordered layout	Tree ordered layout
Node	Role	Task	Work product
Edge	Role collaboration	Task order dependence	Work product production dependence
Scope	Full Process Model	Full Process Model	Full Process Model
Node color	Associated guidances	Associated roles	Associated guidances
Node height	Required and produced work products	Required work products	Tasks where it is an input
Node width	Tasks where it participates	Produced work products	Tasks where the it is an output

Table 1. Blueprints Details

metrics information. It has been successfully used to provide "software maps" intended to help comprehension and visualization.

Given two-dimensional nodes representing entities we follow the intuitive notion that the wider and the higher the node is, the bigger the measurements its size is telling. The color interval between white and black may render another measurement. The convention that is usually adopted [6] is that the higher the measurement the darker the node is. Thus light gray represents a smaller metric measurement than dark gray. An edge between two nodes n and m may be directed representing asymmetry. Direction is usually graphically represented with an arrow. We are not making use of other edge measurements.

4.2 Role Blueprint

Description and Specification: this blueprint shows a role-oriented perspective of a process model. A role (called *Role Definition* in SPEM 2.0) defines a set of skills, competences and responsibilities required for performing it. A role is responsible of some tasks and could participate in others; also a role is responsible of some work products. In general, the more associated to tasks, work products and guidances a role is, the heavier the load the role will have. In this blueprint a role is represented as a node. The width represents the quantity of tasks and the height the quantity of work products this role is in charge of. An edge between two roles represents the collaboration between them, i.e., they work on the same task. Collaboration is not explicit in a SPEM 2.0 model, but it can be deduced.

Example: the Role Blueprint of DTS is presented in Fig. 2. It shows collaboration between roles, so we can see that the *Client* appears isolated. This situation suggests that this role is either not involved in the process (and it is a possible conceptual problem), or that the process has been badly specified. On the other hand, the *Analyst* role should have been related to *Engineer Manager* and it is not, so this is an underspecification of the process model. In this blueprint two roles are much larger than the others: they have more work products assigned and they participate in more tasks. Also the role *Client* is small, and this issue suggests that it is not an active participant in the software process.

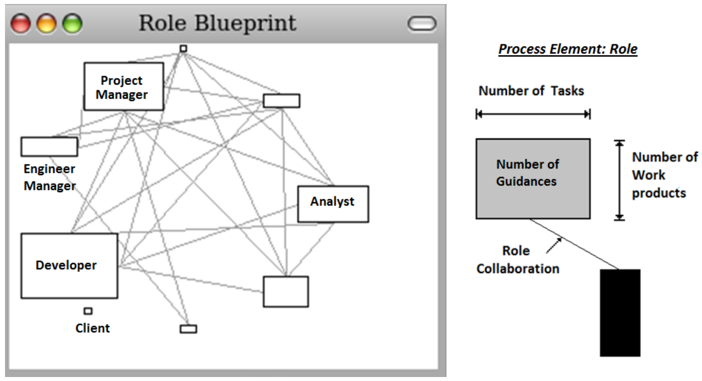

Fig. 2. Role Blueprint of the DTS Process Model.

Interpretation: this blueprint allows us to evaluate if the assigned responsibility is suitable for process requirements. We can also discover overloaded roles when lightweight roles are expected or vice versa, an isolated role when collaboration is required, and roles without training material when it is necessary for a good performance. Therefore, a complex role could be decomposed, simple roles could be integrated, training material could be added, collaborative techniques and enactment constraints could be added.

4.3 Task Blueprint

Description and Specification: this blueprint shows a task-oriented perspective of the process model. A task (*Task Definition* in SPEM 2.0) is the basic element for defining a process. It defines a work unit assignable to roles. A task is associated to input and output work products. A task has a clear purpose: it provides a complete explanation step by step of the associated work. In this blueprint a node is a task. The width and hight of a task node represent the quantity of work products that are required and produced, respectively, and the color represents the number of roles involved. A directed edge between two tasks states an dependency between them: T1 depends on T2 if T1 uses work products produced by T2. The organization of the tasks uses a TreeOrderLayout, so they are placed in a top-down fashion to reflect the dependency order. As a natural and intuitive good practice, each task should be connected to a previous and a next task, except for the first and the last.

Example: the Task Blueprint of DTS is presented in Fig. 3. Node size shows their complexity. In the example, the *Requirements Review* task is more complex than all others, so decomposition may be needed, e.g. *Requirements Review* could be decomposed into *Software Requirements Review* and *User Requirements Review*. Color shows participant roles, so *User Requirements Definition* does not have associated roles. This is a specification error unless the task is au-

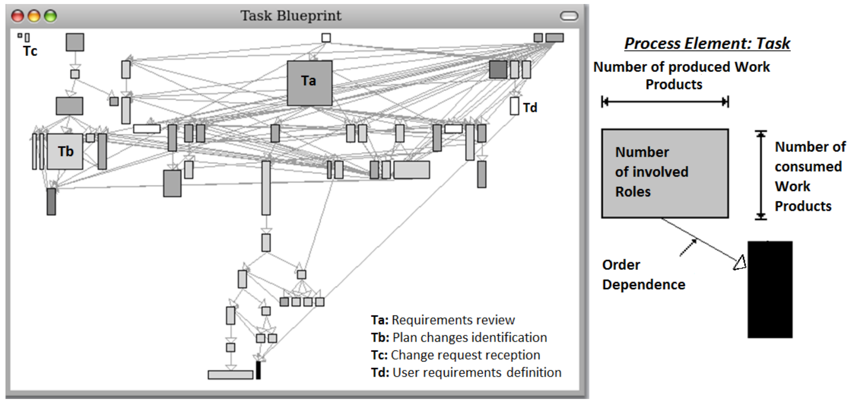

Fig. 3. Task Blueprint of the DTS Process Model.

tomatic. Other problems may be identified in the model such as the existence of many initial and final tasks: tasks without a possible next or previous task. Another problem arises due to the multiple links between tasks of different process areas, indicating a high coupling between main process components. The tasks on the upper left represent the management process area, those on the upper right correspond to engineering process area and those on the bottom belong to testing. These process areas, interpreted as process components, should be connected with as few dependencies as possible using process ports as SPEM2.0 suggests; they can be used with composed work products and reports generated from other work products between main or facade tasks inside each process component. Related small tasks can be redefined as steps of a larger task.

Interpretation: this blueprint allows us to evaluate if task granularity is appropriate for process requirements. It enables the discovery of a complex task, or a disconnected task or task sub graph. A complex task could be decomposed and simple tasks could be integrated. Similarly, this blueprint can be used to find SPI opportunities and misspecifications such as high coupling, final or initial tasks with next or previous tasks, and tasks without assigned roles.

4.4 WorkProduct Blueprint

Description and Specification: this blueprint shows a work product-oriented perspective of the process model. A work product (*Work Product Definition* in SPEM 2.0) is used, modified or produced by tasks. Roles use work products to perform tasks and produce other work products. Roles are responsible for work products, so they aid in identifying skills required for producing them. In this blueprint each node is a work product. The width and height of the node represent the quantity of tasks where the work product is an input or an output, respectively. The color refers to the quantity of associated guidances. An edge

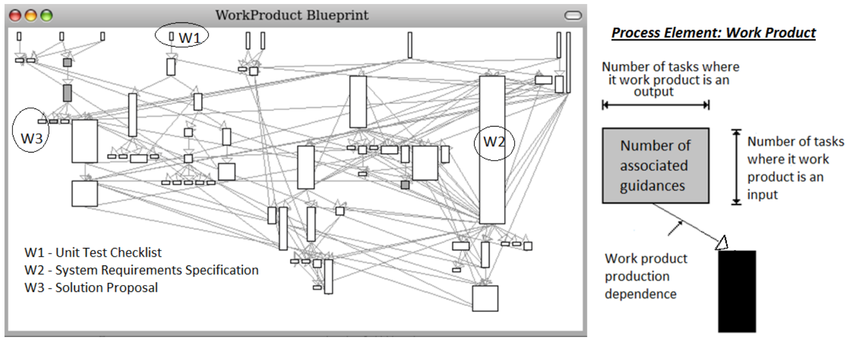

Fig. 4. Work Product Blueprint of the DTS Process Model.

between two work products designates a dependency: W1 depends on W2, if W1 is an output of a task where W2 is input. Any work product should be connected to previous and next work products except for the first and the last.

Example: the WORK PRODUCT BLUEPRINT of DTS is presented in Fig. 4. This blueprint shows a process with small work products except for a few of them. *System Requirements Specification* is big and it can probably be divided into simpler work products. Very few work products appear colored, so this process has not defined a homogeneous guidance for executing the defined process, i.e. the process may not be repeatable because each performer may choose a different the representation. The work product *Unit Test Checklist* is produced during the development, but it is not represented in the model, so it is incorrectly modeled as an initial work product. The same situation occurs with *Solution Proposal* because even if it is not a final product, no task consumes it. It is either a waste work product, so process grease can be eliminated (process improvement), or it was wrongly specified (process model specification improvement).

Interpretation: this blueprint allows us to evaluate if a work product is a bottleneck risk, to discover a disconnected work product, or to identify isolated descriptions in the process model. It can also be used to find SPI opportunities such as complex, bottleneck or waste work products. Guidances are very important in this blueprint because the more formal a process model is the more reference material it needs, so the darker the nodes will be. For example, Unified Process defines templates for each work product, whereas Extreme Programming does not. Similar to tasks, work products must be separated by process areas; connections between them are first candidates to be component ports.

5 Process Model Blueprints Application: A study case

DTS's development process has been defined as a SPEM 2.0 model and implemented in EPF. Even though some evaluations such as CMMI appraisals were included, there is still no information about the quality of the process model itself

Blueprint	***Identified Problems***	***Quantity***	***Improvements***
WorkProduct Blueprint	Few guidelines, some are very long, some are disconnected, many initial and final artifacts	31	23
Task Blueprint	Tasks without roles, disconnected tasks, many initial tasks, many final tasks, many relationships big tasks, deformed tasks	22	19
Role Blueprint	Roles with light load, roles with weight load, roles isolated, there are few guidelines	7	5
Total		60	47

Table 2. Process Model Problems Identified with PMBlueprints

or the adequacy with the company's goals. As part of its SPI project DTS wants to assess the quality of its process, so they used the Process Model Blueprints. The problems found were used as input for the following improvement cycle. The evaluation process was carried out by two process engineers, one from DTS and another one from the University of Chile, and two software engineers for process definition. The steps followed were:

(1) Visualization: the process engineers imported the model from EPF and they realized a short review. Some errors in the tool were identified and some aspects were tuned. A first visualization of the blueprints was generated.

(2) Potential problem identification: for each blueprint problems were identified. Each problem was recorded and a trivial analysis was realized in this step.

(3) Problem analysis: each blueprint was used to find a symptom of a problem and the related elements. For example, *Analysis* was not related to *Client* and *Chief Engineer* roles, so it is possible that some tasks where these roles participated were not adequately specified. Some problems required more analysis and discussion with users of the software process.

(4) Data collection: the potential and actual problems were collected, and also some solutions were proposed.

(5) Data analysis: the data collected in this study case was analyzed and the main results are presented below.

Table 2 summarizes the main problems found during the evaluation. For a total of 132 process elements, 47 actual problems were found (false positives have been discarded). So, the process model error rate was of 0.356 errors per model element. In general, the review found that there were few guidances associated to model elements. This situation suggests low formality in software process specification. Disconnected elements cause modeling problems: some connections were omitted at modeling time and others caused serious conceptual problems in the software process. For example, the isolated *Client* role was a symptom of a problem, and actually a big problem: the client was not involved in many tasks where his/her participation is required, specially in requirements tasks. We suggest two kinds of improvements:

Software Process Improvements: the main suggested improvements are: increase guidance for roles (guide of work) and for work products (templates,

Blueprint	*Improvements to Specification*	*Improvements to Process*
WorkProduct Blueprint	19	4
Task Blueprint	16	3
Role Blueprint	2	3
Total	37	10

Table 3. Improvements suggested with PMBlueprints

examples and concepts); task re-design for getting tasks of medium size (neither too large nor too small), such as *Requirements Review*; big work products decomposition such as *System Requirements Specification*; decrease coupling between process areas, with few relationships between work products and tasks; increase the *client* participation and improve the collaboration required between the *Analyst* and *Engineer Manager* roles. The bottleneck risks identified on the *System Requirement Specification*, *Alpha Component* and *Alpha System Cycle n* work products are normal bottlenecks because of the incremental nature of the software process, so these work products are completed and refined in each cycle of the software process. However, the *Architecture* work product and its associated tasks are not suitable to this incremental development.

Software Process Model Specification Improvement: the main improvements to software model specification are about missing nodes, edges and guidances. Several problems were identified directly, others required more analysis as collaboration between specific roles, and others were generally due to conceptual problems of the software process.

6 Conclusion and Further Work

As a complement to process model design, it is also necessary to evaluate software processes. This paper presents a mechanism for recovering software process architectural views based on blueprint visualization. Many process model blueprints can be defined, but three essential ones have been described here: Task Blueprint, Role Blueprint and Work Product Blueprint. The approach has been complemented with ProcessModel, the visualization tool we built (all screenshot used in this paper were obtained from ProcessModel), based on Mondrian and Moose. This tool imports process models defined in SPEM 2.0 from EPF, and automatically produces the blueprints described in this paper. These blueprints are used to find errors and problems in the specification and improvement phases of software process model. The paper presents an experience with a process being implemented in a small company using EPF. The assessment we conducted identified a number of important problems. This experience lets us conclude that our approach is useful for the evaluation of software process models within a SPI project. However, more experience is required for more conclusive results; metrics and blueprints should also be extended, formalized and validated. Some limitations of our approach occur because different visualizations can be obtained from different metrics, so there is no unique way of measuring and the interpretation must be put in context. The approach can be extended using goal oriented metrics. Currently, the approach is been used for evaluating the open source process models defined in the EPF website, and it is being applied in other small companies in Chile. Also the tool is being improved, and other blueprints can be defined based on other concepts and elements defined in SPEM 2.0, or based on other metrics.

References

1. Reda Bendraou, Jean-Marc Jezéquél, and Franck Fleurey. Combining Aspect and Model-Driven Engineering Approaches for Software Process Modeling and Execution. In *ICSP 2009*, pages 148–160, Berlin, Heidelberg, 2009. Springer-Verlag.
2. Gerardo Canfora, Félix García, Mario Piattini, Francisco Ruiz, and Corrado Aaron Visaggio. A family of experiments to validate metrics for software process models. *Journal of Systems and Software*, 77(2):113–129, 2005.
3. Paul Clements, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Robert Nord, and Judith Stafford. *Documenting Software Architectures*. Addison-Wesley, 2002.
4. Jonathan E. Cook and Alexander L. Wolf. Software process validation: quantitatively measuring the correspondence of a process to a model. *ACM TOSEM*, 8(2):147–176, 1999.
5. Jidong Ge, Hao Hu, Qing Gu, and Jian Lu. Modeling Multi-View Software Process with Object Petri Nets. *ICSEA 2006*, 0:41, 2006.
6. Tudor Gîrba and Michele Lanza. Visualizing and characterizing the evolution of class hierarchies. In *WOOR 2004 (5th ECOOP Workshop on Object-Oriented Reengineering)*, 2004.
7. Volker Gruhn. Validation and verification of software process models. In *Proceedings of the European symposium on Software development environments and CASE technology*, pages 271–286, New York, NY, USA, 1991. Springer-Verlag New York, Inc.
8. Watts S. Humphrey. *Managing the Software Process*. Addison-Wesley, Reading, MA, 1989.
9. ISO. ISO/IEC 15504 : Information technology - software process assessment and improvement. Technical report, 1998.
10. David Jacobs and Chris Marlin. Multiple view software process support using the MultiView architecture. In *ISAW-2 and Viewpoints '96*, pages 217–221. ACM, 1996.
11. David Jacobs and Chris D. Marlin. Software process representation to support multiple views. *IJSEKE 1995*, 5(4):585–597, 1995.
12. Michele Lanza and Stéphane Ducasse. Polymetric Views-A Lightweight Visual Approach to Reverse Engineering. *TSE 2003*, 29(9):782–795, September 2003.
13. Jill H. Larkin and Herbert A. Simon. Why a Diagram is (Sometimes) Worth Ten Thousand Words. *Cognitive Science*, 11(1):65–100, 1987.
14. Michael D. Lee, Marcus A. Butavicius, and Rachel E. Reilly. An Empirical Evaluation of Chernoff Faces, Star Glyphs, and Spatial Visualizations for Binary Data. In *APVis 03*, pages 1–10. Australian Computer Society, Inc, 2003.
15. OMG. Software Process Engineering Metamodel SPEM 2.0 OMG. Technical Report ptc/08-04-01, 2008. Object Managemente Group.

16. Leon Osterweil. Software processes are software too. In *ICSE 1987*, pages 2–13, Los Alamitos, CA, USA, 1987. IEEE Computer Society Press.
17. G. Perez, K. El Emam, and N. H. Madhavji. Evaluating the congruence of a software process model in a given environment. In *ICSP 1996*, page 49, Washington, DC, USA, 1996. IEEE Computer Society.
18. C. Sauer, D.R. Jeffery, L. Land, and P. Yetton. The effectiveness of software development technical reviews: a behaviorally motivated program of research. *Software Engineering, IEEE Transactions on*, 26(1):1–14, Jan 2000.
19. SEI. CMMI for Development, Version 1.2. Technical Report CMU/SEI-2006-TR-008, Software Engineering Institute, 2006.
20. Edward Tufte. *Envisioning Information.* Graphics Press, Cheshire, CT, 1990.