Software Process Definition: a Reuse-based Approach

Ahilton Silva Barreto[1], Leonardo Gresta Paulino Murta[2], Ana Regina Rocha[1]

[1] COPPE/UFRJ - Universidade Federal do Rio de Janeiro
Caixa Postal 68511 – CEP 21945-970 – Rio de Janeiro, Brazil
{ahilton, darocha}@cos.ufrj.br

[2] Instituto de Computação – Universidade Federal Fluminense (UFF)
Niterói, RJ, Brazil
leomurta@ic.uff.br

Abstract. This paper presents a software process definition approach based on reuse techniques, which aims at making this activity easier, providing relevant knowledge, avoiding rework, and improving the adequacy of the defined processes. Stability and performance data from the subprocesses that can compose a process are also considered. We describe the different scenarios to process reuse (software process implementing organizations, software development organizations, and software projects), the structure and concepts to be used in the approach, and the strategies to define processes for reuse and with reuse. Concepts such as process components, architectures, lines and features are described and used.

Keywords: Software Process Definition, Software Process Reuse, Software Process Components, Software Process Lines

1 Introduction

Software process definition is not a simple task; it demands experience and involves knowledge from several aspects of software engineering. There are many factors to consider, such as: organization's or projects' needs and characteristics, techniques and methods to use, adherence to standards and reference models, business constraints (schedule, costs, etc), among others. Therefore, this task usually requires a highly skilled professional which is able to reconcile all these factors. Software organizations often do not have this kind of professional available, and usually need help from more experienced consultants. However, if knowledge possessed by experienced process engineers could be made explicit, formalized, and made available to other professionals, it would probably be possible to reuse this knowledge in an effective way.

Software processes can be reused in different scenarios. Organizations that help other organizations to define and deploy software processes (Software Process Implementing Organizations - SPIO) usually need to define processes to several different organizations. Although each organization has its own characteristics and peculiarities (that will probably lead to unique processes), many characteristics of the processes are similar to previously defined processes by the same SPIO to other organizations, which is a good process reuse opportunity. Software organizations can

also reuse previously defined processes in different projects with similar characteristics. Software projects can also contribute to reusable process definitions through the collection of data and lessons learned from the enactment of processes.

To accomplish the reuse of software process related knowledge, reuse techniques have been adapted from traditional software product development to the context of software process definition [1-3]. Concepts like components, architectures, product lines, and patterns have been used to define and improve software processes.

This paper presents a software process definition approach that aims at making this activity easier and more effective, catching the reuse opportunities that exist in the different contexts in which processes need to be defined (software processes implementation organizations, software development organizations, and software projects), through the use of reuse techniques. In our approach, knowledge acquired from more experienced software process engineers and from processes and process components enactment can be made available for reuse in order to define more adequate processes. We present: (i) the main goals and characteristics of our approach; (ii) the proposed structure and concepts to represent and reuse processes; (iii) the strategy to be adopted at each reuse context; and (iv) the strategy to define processes for and with reuse. This paper is divided into four sections, including this introduction. Section 2 presents some process definition and process reuse concepts, and related work. In Section 3, we present the proposed approach. Finally, Section 4 presents some conclusions and future work.

2 Definition and Reuse of Software Processes

The software process is a critical factor for delivering quality software systems, as it aims to manage and transform the user need into a software product that meets this need [4]. It defines how the development is organized, managed, measured, supported and improved, irrespective of the techniques and methods used for development [5].

Software process definition can take place in different levels of abstraction. First, a standard software process is defined for the organization. Based on this organizational process, specialized processes can be defined considering aspects like development paradigm, application domains, and others. Finally, project defined processes can be instantiated from standard processes (specialized or not) [6].

According to ISO/IEC 15504 [7], a standard process is the set of definitions of the basic processes that guide all processes in an organization. It describes the fundamental process elements that will be part of the projects' defined processes. It also describes the relationships (for example, ordering and interfaces) between these process elements. A defined process, on the other hand, is a process that is managed (planned, monitored and adjusted), and tailored from the organization's set of standard processes according to the organization's tailoring guidelines. A project's defined process provides a basis for planning, performing, and improving the project's tasks and activities of the project [7].

Reference models and standards, like CMMI-DEV [8], MPS.BR [9] and ISO/15504 [7] establish that, in high maturity organizations, processes need to be defined based on smaller process units, usually called subprocesses, process elements

or process components. They also establish that processes have to be defined based on the selection of the more adequate subprocesses to compose the process, considering their historical stability, capacity and performance data. The stable process can be defined as a predictable process, whose performance and variability are known, allowing the elaboration of estimates based on its own past performance. To verify if a process is stable, it is necessary to enact it, collect a significant amount of performance measurements and check if the process is under statistical control [10]. Therefore, to select subprocesses to compose a process considering their known past stability and capacity data (i.e. the way it is able to perform) tends to produce more adequate processes.

Several works described in the literature argue that software processes have strong similarities with software products. Osterweil [11], for instance, established that software processes are software too, and just like software, could have its requirements specified, could be modeled, developed, tested and reused. There are also described similarities between software process reuse and software product reuse. Kellner [1] argues that knowledge of software product reuse techniques, such as: architectures populated with reusable components; repository management to store, catalog, search, access, etc. the reusable assets; configuration management of reusable assets; and others; could be applied to software processes as well.

To achieve effective process reuse, it is necessary to develop effective methods to capture the common and variant elements of specific processes and create process definitions that can be applied in a variety of situations, i.e., that are reusable [12].

One of the ways software processes can be reused is through the use of process components. A process component can be considered an encapsulation of process information and behaviors at a given level of granularity [13]. So, a process can be treated as the integration of process components in different levels of granularity [14]. These components could provide strong and efficient support for process engineers and project developers to build process models and perform process dynamic adjustment, which brings great value to the process optimization and control [15].

The concept of architecture is also used in the context of software processes, since the use of software components alone still seems to be insufficient, due to the fine granularity. According to Chrissis [8], a process architecture involves the ordering, interfaces, interdependencies and other relationships among the elements of a process in a standard process. In a simplified way, we can consider that process architecture defines the “skeleton” that a process must have, establishing the main elements and how they relate to each other, not necessarily defining the details of these elements. A similar concept is process template, which can be defined as a generic and reusable process model that establishes a starting point to the definition of a new process model [2]. A process template can be customized to address the requirements of a particular context (methodological, organizational or technological), or also combined with other templates or instantiated process models [16]. These concepts are closely related to the concept of framework, usually applied in software product development.

Sofware product lines can also be adapted to be used in software processes. A product line works like a factory, that instantiates similar products, each one of them with a set of characteristics or features, through the arrangement of existent components. Likewise, it is possible to consider that processes can be defined using the same ideas, in other words, processes can be instantiated from previously defined process

components, in a way that each instance satisfies a set of chosen process characteristics or features. So, software process lines are product lines whose products are software processes [3, 17].

3 An Approach to Software Process Definition based on Reuse

To define the main general aspects of our approach we have considered: (i) a literature search considering the most relevant aspects of process reuse, and how the topic has been addressed by the academy and industry; (ii) the experience of many years of COPPE/UFRJ as a software process implementing organization. This kind of experience was very important to identify improvement opportunities in process definition activities, considering the real needs of software process implementing organizations and software development organizations aiming at defining software processes.

So, the main aspects of our approach are: (i) the definition of the main concepts and infrastructure to make software process reuse possible; (ii) a strategy to define software processes reusing previously defined process elements; (iii) a strategy to define software process elements for later reuse; (iv) a strategy to reuse processes in the different contexts reuse can take place; (v) a measurement strategy considering reusable software process elements, to allow their stability and capability analysis; (vi) a configuration management strategy to control the reusable elements and their evolution; and (vii) supporting tools.

3.1 Main concepts and infrastructure

To allow software process reuse to happen, first it is necessary to define how software process related knowledge will be structured and represented in a way that it can be later reused. In this approach, a software process component can be defined in any level of detail, i.e. from a single activity to a whole process. However, a process component always has to be composed by at least one activity, since we consider that a finer reuse unit would probably introduce unnecessary complexity and make reuse activities more difficult. Fig. 1 shows a high level and simplified model of the core concepts of our approach and how they relate to each other.

A process element is a decomposition of a process in some level of granularity. We define two kinds of process elements: process components and activities. These concepts are closely related, but the main difference is that a component is defined for reuse. Thus, a component may have an internal architecture, composed by other components that may be not completely defined, to allow some kind of customization (variability). Moreover, a component is considered something that is worth reusing, analyzing its stability or performance, versioning, establishing traceability, and so on. Activities, on the other hand, are process elements that are completely defined, and do not exhibit any kind of variability.

The process architecture is similar to a workflow, and may be composed of process components, activities or any combination of them. The process architecture defines the main structure the process has to have, determining the main elements and how they relate to each other, not necessarily defining all their details. An example of

process architecture could be a lifecycle model. In this case, we know the main phases of the process, how they are related and their goals. However, several different instantiations are possible using the same structure, i.e. different components or activities that conform to the established structure can be used, depending on the situation. Thus, the process architecture aims to guide and make it easier to define processes, determining a basic structure to follow during the definition. The architecture needs to be flexible enough so that it can be adapted to the several different situations that can happen on organizations or projects. The architecture can also define which components are optional, i.e. can be present or not in process instantiations.

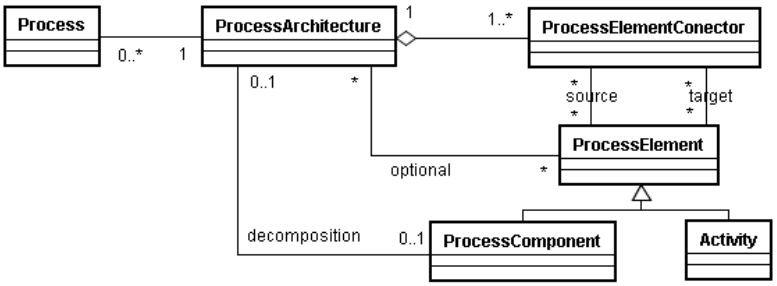

Fig. 1. Core concepts and their relationships

A process element connector is used to connect several process elements (components and/or activities). A connector has a list of source elements, a list of target elements and some rule to associate them (e.g. start-end; end-end; etc).

If a component does not have an internal architecture, it means it is not yet defined how the component has to be accomplished. For instance, if a component like "Establish Estimates" is not completely defined, it means any component that can accomplish this goal could be "plugged" and instantiated in the process. So, in our example, "Establish Estimates" is considered a variation point in the process architecture, while any component that could be plugged into this variation point is called a variant. Examples of variant components are "Establish Estimates using Use Case Points", "Establish Estimates using Function Points", and so on. A process architecture that has variation points or optional elements is called a software process line, since like a product line does with software products, it is able to instantiate different software processes, depending on some characteristics, or process features.

A process component is considered a concrete component when it does not allow any kind of variability. In other words, the component is completely defined and will be instantiated and enacted in the process exactly the way it is. In a concrete component, there are no remaining decisions to make, and it can be directly enacted, measured and controlled in a software project. A process component is said to be an abstract component when it is not yet related to a single enactment possibility, i.e. there are still variation points to treat. In this kind of component, the definition is not complete, and the component cannot be enacted in a project. Fig.2 defines these two kinds of components more formally. We can see that "x", to be a concrete component, has to be a process component, possess an internal architecture ("y"), and each process element in "y", or is an activity or is a concrete process component. This recursive

definition establishes that in any level of decomposition of the concrete component, there is no remaining variability. An abstract component, on the other hand, is simply a process component that is not concrete.

$$(\forall x)(ConcreteComponent(x) \rightarrow (\Pr ocessComponent(x) \wedge ((\exists y)(\Pr ocessArchitecture(y) \wedge InternalArchitecture(x, y) \wedge ((\forall z)(\Pr ocessElement(z) \wedge ArquitectureElement(z, y) \wedge (Activity(z) \vee ConcreteComponent(z)))))))$$

$$(\forall x)(AbstractComponent(x) \rightarrow \Pr ocessComponent(x) \wedge \neg ConcreteComponent(x)$$

Fig. 2. Formally defining concrete and abstract components

Fig.3 shows a process line example. In this process line, the purpose is to plan the process of a project, plan the project and assess the quality of the produced documents. First component, "Plan the Process" is a concrete one and does not admit any variability, and will be included in the defined process exactly as is. The component "Plan the Project" had its internal architecture depicted in the figure. We can see that "Establish Estimates" is an abstract component, becoming a variation point. Therefore, this component can be instantiated in a process in three different ways, through the choice of one of its three variants ("Experience-Based", "Function Points" and "Use Case Points"). So, when defining the process, the responsible can choose which one of them will be included in the process. The last component, "Assess Work Product Quality" is optional, and can be included or not in the defined process.

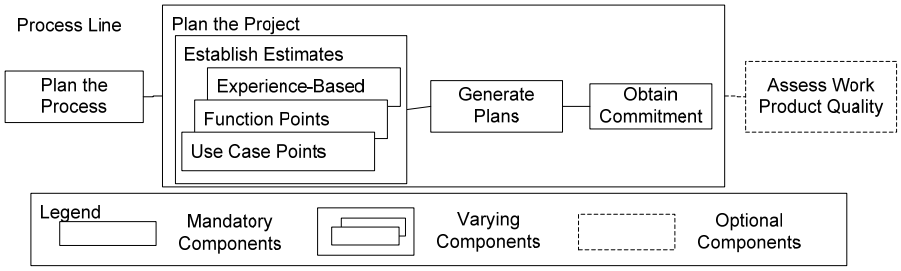

Fig. 3. Process line example

3.2 Software Process Reuse Scenarios

The existence of one or more reusable software process components repository is necessary to allow the effective reuse of software processes. These repositories store process components, process lines, knowledge related to the enactment of components, components measurements, among others. The repositories can be used when defining processes for organizations or projects (definition with reuse), where components and other reusable items can be searched and used to compose processes. Components repository need to be fed and constantly evolved, in order to offer to projects and organizations a useful and comprehensive set of reusable items (definition for reuse). In this approach, we consider that process reuse can take place in different contexts, such as: (i) Software Process Implementing Organizations (SPIOs) – may need to define processes, usually similar to several different organizations; (ii) Software Development Organizations – may need to define their standard processes, or

specialize these processes to situations which are common in the organization; (iii) Software Projects – contribute to the components repository with data from its enactment, such as measures, improvement requests, etc.

As shown in Fig.4, the SPIO establishes and feeds its repository with process components, process lines, measures, data from the enactment of components, among others (definition for reuse), considering general software process requirements, previously defined processes, maturity models, and other process requirements. When it is necessary to define a standard process for an organization or group of organizations (definition with reuse), the items from the repository can be (re)used, based on the particular needs, to compose new processes. When there are no available items in the repository to fulfill a specific need, a call to the process of definition for reuse can be made. Therefore, it is possible to constantly enrich the repository. Together with the process or set of processes resulting from the definition, it is also possible to provide the organization with a subset of the SPIO´s components repository, in order to allow the organization to modify and evolve its processes and keep the culture of process reuse. In this repository may exist partial solved process lines, i.e., with some variants already selected, but with some variation points still open. The standard process itself can be deployed this way, with some variability to solve. The reuse in the organizational context happens in a similar way, but the repository is the own organizational repository, and the defined processes are project defined processes.

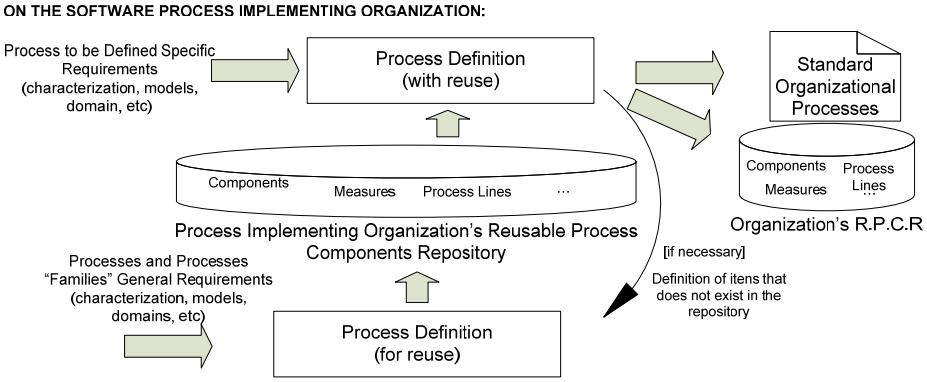

Fig. 4. Software Process Reuse on Software Processes Implementing Organizations

In Fig. 5 (left hand side) we can see how process enactment happens on projects. Project's defined processes are enacted and contribute with data to organization´s reusable process components repository. On-the-fly improvements can be made on the project's process, if needed, through the replacement or maintenance of components in use. On the right hand side of Fig. 5 we can see the feedback between the organizational repository and the repository of the SPIO that originated it. Organizations can contribute to the SPIO sending data related to the enactment of reusable items. This information can be useful for the SPIO to assess the adequacy of each component in the context they were used and also determine process components stability and performance data in that context. The SPIO can analyze the gathered information from organizations and provide back to the organizations benchmarking data, allowing them to compare the performance of the components they use. SPIOs

can also provide updated data from its repository, such as new process components, new process lines, and so on.

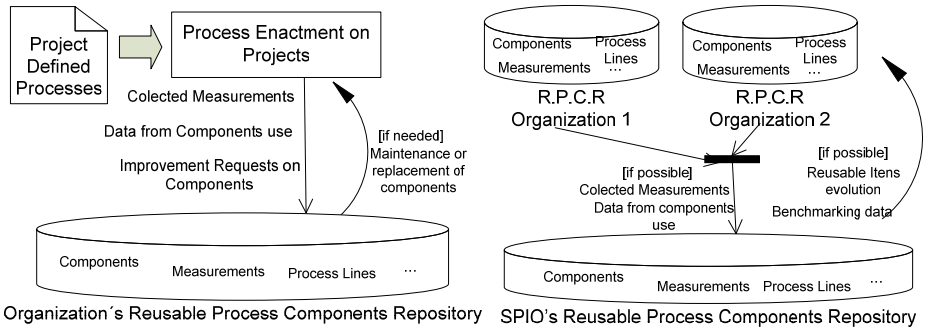

Fig. 5. Software Process Reuse on Projects and Feedback loop

Consider the example shown in Fig.3, and assume that the presented process line was defined by the SPIO, which made it available to several organizations. Some organizations, however, are more mature than others and never use estimates based only on manager's experience. Moreover, they could always perform quality assessments. Due to that, they could create a specialized process line, more adequate for them, as shown in Fig.6. Projects in that organization have to choose one among the available variants, considering project characteristics and data from the enactment of each variant. It is important to say that SPIOs and organizations may have several different process lines, Fig.3 and Fig.6 just present intentionally simple examples for didactic reasons.

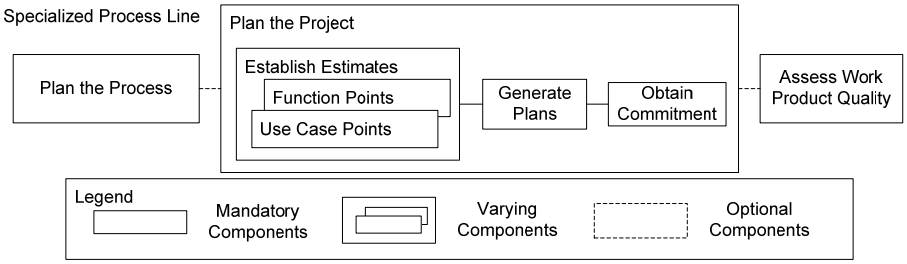

Fig. 6. Specialized Process Line Example

3.3 Defining Software Process for and with Reuse

After some time using and evolving the repository, the number of items to choose can grow dramatically, and it could complicate the search for adequate components. We define the concept of process feature to alleviate this problem and also guide process definition based on process requirements. A process feature is the translation of the concept of feature used in the traditional product lines domain to the context of processes. A feature can be seen as a kind of functionality or classification that the process has to be compliant. Examples are: Adherence to CMMI level 2, Object

Oriented Paradigm, High Reliability, and others. A feature constrains the use of components, establishing a set of components that can or cannot be used. Thus, it consists in a set of rules applied to process components. These rules allow us to establish several kinds of traceability.

Fig. 7 illustrates the use of features in process definition, showing that the number of choices can diminish a lot, and that focus is on process requirements. Process features may contribute to allow less experienced professionals to define processes focusing on process requirements, since the knowledge on which components to choose is already modeled on features.

During process definition with reuse, it is important to consider the historical stability and performance of the components being selected. This analysis can be performed, since we can associate measures to all components. Therefore, it is possible to check if a component is adequate for a certain context, based on its stability and performance in other similar contexts it has been enacted, since we can compare its past performance with the required performance. Variant selection for a variation point can be done based on this analysis.

The process definition for reuse (which is responsible for maintaining the components repository) can be done in at least three ways. The first is based on any requirements, considering any references or experiences, in other words, it is a "freestyle" definition. The second option consists in the analysis of several previously defined processes. Based on these processes, components and processes are derived; i.e. parts of processes that are always present will be mandatory elements of process lines, while the parts that were sometimes present, but not always, will be considered variation points. The last situation is an incremental approach to componentize processes. The idea is that "legacy" processes can have some of its parts componentized throughout time. It is a kind of process refactoring, and will allow organizations to continually migrate their processes to the new structure so that they can be more easily reused. The three ways are not mutually exclusive, and can be combined as needed.

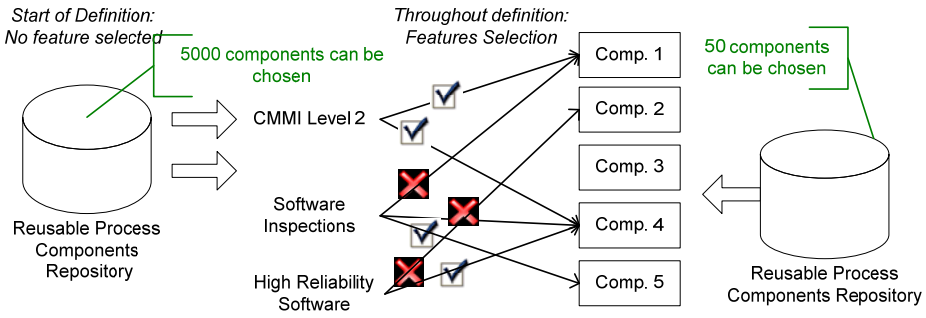

Fig. 7. Using Process Features

4 Conclusions

In this paper we presented a software process definition approach based on reuse. This approach intends to make process definition easier in different scenarios, making this activity less expensive and improving the quality of the processes defined,

through the reuse of knowledge from more experienced process engineers and also historical stability and performance analysis of subprocesses.

This approach is still being developed, and several aspects (e.g. components measurement, reusable items configuration management, process ontologies, integration into software development environments, supporting tools) could not be described here due to the limited space available. In the present moment, we are in the final stage of the definition of the reuse structure, also adapting a previously defined software process ontology to include the proposed concepts.

In the near future, we will start the development of tools to support our approach, what will allow us to start collecting information from its use both from academic and industrial environments.

References

1. Kellner, M.I.: Connecting Reusable Software Process Elements and Components. In *10th International Software Process Workshop*, pp.8-11. Dijon, France (1996).
2. Reis, R.Q.: APSEE-Reuse: Um Meta-Modelo para Apoiar a Reutilização de Processos de Software. 2002. Tese de Doutorado. UFRGS: Porto Alegre, Brasil.
3. Washizaki, H.: Building Software Process Line Architectures from Bottom Up. In *Product-Focused Software Process Improvement*, pp.415-421. Amsterdam, Netherlands (2006).
4. Acuña, S.T., et al., *The Software Process: Modelling, Evaluation and Improvement*, in *Handbook of Software Engineering and Knowledge Engineering*, S.K. Chang, Editor. 2000, World Scientific Publishing Company: Singapore. p. 193-237.
5. Derniame, J.-C., B.A. Kaba, and D. Wastell: Software Process: Principles, Methodology and Technology. 1999, Berlin: Springer-Verlag.
6. Rocha, A.R.C.d., J.C. Maldonado, and K.C. Weber: Qualidade de Software: Teoria e Prática. 2001, São Paulo, Brasil: Prentice Hall.
7. ISO/IEC-15504: Information Technology – Software Process Assessment. 2003, Parts 1-9.
8. Chrissis, M.B., M. Konrad, and S. Shrum: CMMI: Guidelines for Process Integration and Product Improvement. 2nd ed. 2006, Nova York, Estados Unidos: Addison-Wesley.
9. SOFTEX: MPS.BR - Melhoria de Processo do Software Brasileiro, Guia Geral (v1.2). 2007, SOFTEX - Associação para Promoção da Excelência do Software Brasileiro.
10. Wheeler, D.J. and D.S. Chambers: Understanding Statistical Process Control. 1992, Knoxville, Estados Unidos: SPC Press.
11. Osterweil, L.: Software Processes Are Software Too. In *International Conference on Software Engineering*, pp.2-13. Monterey, Estados Unidos (1987).
12. Hollenbach, C. and W. Frakes: Software Process Reuse in an Industrial Setting. In *4th International Conference on Software Reuse*, pp.22-30. Orlando, Estados Unidos (1996).
13. Gary, K.A. and T.E. Lindquist: Cooperating Process Components. In *COMPSAC99*, pp.218-223. Phoenix, United States (1999).
14. Fusaro, P., G. Visaggio, and M. Tortorella: REP - ChaRacterizing and Exploiting Process Components: Results of Experimentation. In *Working Conference on Reverse Engineering*, pp.20-29. Honolulu, United States (1998).
15. Ru-Zhi, X., et al.: Reuse-Oriented Process Component Representation and Retrieval. In *ICCIT'2005*, pp.911-315. Shangai, China (2005).
16. Franch, X. and J. Ribó: Supporting Process Reuse in PROMENADE. 2002, Departament de Llenguatges i Sistemes Informàtics, Universitat Politécnica de Catalunya.
17. Rombach, H.D.: Integrated Software Process and Product Lines. In *International Software Process Workshop*, pp.83-90. Beijing, China (2005).