Journal of Universal Computer Science, vol. 17, no. 13 (2011), 1765-1799
submitted: 8/2/10, accepted: 29/8/11, appeared: 1/9/11 © J.UCS

Software Process Definition: a Reuse-based Approach

Ahilton Silva Barreto
(COPPE/UFRJ - Universidade Federal do Rio de Janeiro
Caixa Postal 68511 – CEP 21945-970 – Rio de Janeiro, Brazil
ahilton@cos.ufrj.br)

Leonardo Gresta Paulino Murta
(Instituto de Computação – Universidade Federal Fluminense (UFF)
Niterói, RJ, Brazil
leomurta@ic.uff.br)

Ana Regina Cavalcanti da Rocha
(COPPE/UFRJ - Universidade Federal do Rio de Janeiro
Caixa Postal 68511 – CEP 21945-970 – Rio de Janeiro, Brazil
darocha@cos.ufrj.br)

Abstract: Software product development has been taking advantage of reuse techniques for some decades. Concepts like software components, architectures, and product lines have been successfully applied in several contexts to develop software products, although some difficulties are still faced. Software processes have strong similarities with software products, and some researchers argue that they are software too. Therefore, we believe that software processes may take advantage of some benefits expected by the use of existing software products reuse techniques, adapted to software processes. It is also possible that similar difficulties are faced. This paper presents a software process definition approach based on reuse techniques, which aims at making some of the benefits expected by software product reuse available to software process definition activities. Concepts such as process components, architectures, process lines and features are described and used. We describe the proposed approach, tools developed to support it, and also results of a survey on the expected benefits and difficulties on software process reuse in the point of view of experienced software process engineers.

Keywords: Software Process Definition, Software Process Lines, Software Process Reuse, Software Process Components
Categories: D.2.0, D.9, D.2.13, K.6.3

1 Introduction

Several works described in the literature argue that software processes have strong similarities with software products. Osterweil [1987], 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 works in the literature that describe similarities between software process reuse and software product reuse. Kellner [1996] argues that knowledge of software product reuse techniques could be applied to software processes as well. Some examples of such knowledge are: architectures populated with reusable process elements, to represent the main structure of software processes and make their definition and tailoring easier; repository management infrastructure to store, catalog, search, retrieve, and access reusable process elements; configuration management of reusable process elements, including their version control and change management; etc.

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: needs and characteristics of the organization or project, 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. Often, software organizations do not have this kind of professional available, and usually need help from more experienced consultants.

An even more complex scenario is when processes need to be defined not for one, but for several different organizations. This is a common requirement for software process implementing organizations (hereafter SPIOs). A SPIO is a consultant organization that is hired by other organizations intending to define, deploy, or improve their software processes. In Brazil, the concept of SPIO has gained importance with the publication of MPS.BR [SOFTEX, 2009], a Brazilian reference model in which this kind of organization is very important. SPIOs are authorized by the MPS.BR coordinators to support software process improvement on other organizations. Most of the improvement initiatives aiming to achieve MPS.BR maturity levels were supported by SPIOs, and today there are about 200 organizations successfully appraised in this model. It can give us an idea of the relevance of the SPIOs and the need to properly support them.

However, in spite of these complex scenarios, if knowledge belonging to experienced process engineers could be made explicit, formalized, and available to other professionals, it would probably be possible to reuse this knowledge in an effective way. The interplay of software processes and software reuse is wide. SPIOs 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 development organizations can also reuse previously defined processes in different projects with similar characteristics. Finally, software projects can also contribute to reusable process definitions through the collection of data and lessons learned from the enactment of processes. Although SPIOs and organizations may always perform some kind of ad-hoc reuse (e.g., they may adapt and reuse process descriptions or templates, performing a kind of oportunistic reuse, similarly to what happens in product development, in which developers try to copy-and-paste existing code), it is expected that a defined reuse approach will leverage process reuse in these contexts, optimizing the gains of each reuse oportunity.

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 [Kellner, 1996; Washizaki, 2006b; Reis et al., 2001]. Concepts like components, architectures, product lines, and patterns have been adapted to the context of software processes, as we describe in Section 2.

In this paper we present a software process definition approach that aims at catching the reuse opportunities that exist in the different contexts in which processes need to be defined (software processes implementing organizations, software development organizations, and software projects), through the use of reuse techniques. We adapt some concepts from software product reuse expecting that some of their expected benefits also apply to software processes reuse, such as less rework, increased quality, and less time required to perform the activities. However, we also consider that some of the difficulties faced by software product reuse initiatives may also apply to processes, such as lack of quality of the reusable elements, lack of adequate supporting tools, and high costs to deploy the reuse initiatives. In our approach, knowledge acquired from more experienced software process engineers and from processes enactment can be made available for reuse in order to define more adequate processes, as described in the next Sections. Furthermore, we intend to make process definition an easier activity, even for less experienced professionals. Thus, reusable process elements have to be simple enough to allow a professional trying to define processes be able to do it, defining intelligible processes without having to worry with excessive formalisms or complexity. In order to define the main aspects of our approach, some additional requirements were also considered:

1. The need to consider legacy processes – The existence of legacy software processes, i.e., software processes that were already defined, that would have to be made reusable should be considered. Therefore, the definition of the concepts used by our approach and their interrelationships had to be performed in a way that legacy processes could be modelled without modification and could be refactored to support reuse in the future. In other words, we chose an evolutionary approach over a revolutionary approach, aiming at not losing what was already available and also at not forcing an extreme and probably very expansive change. Moreover, since the existence of legacy processes is common to several other organizations, it was important to describe how to make them reusable.
2. The need to consider multi-organizational contexts – COPPE/UFRJ is an important SPIO in Brazil and has already supported more than 40 organizations to define, enact, and improve their processes. It usually has to define processes to several different organizations. We believe this context is common to many other organizations, and therefore it was important to properly address the reuse opportunities of this scenario.
3. The need to consider practices from high process maturity levels – Reference models and standards, like CMMI-DEV [Chrissis et al., 2006], MPS.BR [SOFTEX, 2009] and ISO/15504 [ISO/IEC-15504, 2003] establish that, in high maturity organizations, processes need to be defined based on smaller process units. 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. Therefore, the selection of 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. Thus, the existence of subprocesses and ways to reuse such subprocesses is fundamental to support the needs of high-maturity organizations.

This paper is a major evolution of a previous conference paper [Barreto et al., 2008] that described the main aspects of our software process reuse approach. The new content material comprises the focus and a broader discussion on process reuse itself, implemented tools that support the approach, and an overall update on our reuse approach. We also describe results of a survey that aimed at capturing the expected level of benefits and difficulties of each process reuse approach used (process components, process lines, and process features) in the point of view of experienced process engineers. We also present some usage results of parts of the approach, as described in [Barreto et al., 2010].

This paper is organized into eight sections, including this introduction. Section 2 presents some background concepts and related work on software process definition and reuse. In Section 3 we explain how the main reuse concepts were adapted to our approach. Section 4 presents the considered process reuse scenarios, while Section 5 shows how to define reusable process elements. In Section 6 we present and discuss the results of the performed survey. Section 7 presents an environment to support process reuse that has been developed. Finally, Section 8 presents some conclusions and future work.

2 Background and Related Work

A software process can be defined as the coherent set of policies, organizational structures, technologies, procedures, and artefacts that are needed to conceive, develop, deploy, and maintain a software product [Fuggetta, 2000]. The essence of the process paradigm seems to be that humans solve problems by creating process descriptions and then instantiating them to reach the solution. Rather than repetitively and directly solving individual instances of problems, humans prefer to create generalized solution specifications and make them available for instantiation (often by others) to solve individual problems directly [Osterweil, 1987].

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 [Acuña et al., 2000]. The following facts are among the key benefits of a defined process [Madhavji, 1991]: (i) it can be measured against itself or other processes and any differences can be brought to reason; (ii) it can be used to promote process understanding and standards among software developers; (iii) it can be reused; (iv) it can simplify process management, control, and automation; (v) it can help identifying process measurements points; (vi) it can become the basis for the next level of process improvement; (vii) it may enable effective communication about software processes; (viii) it may support co-operative work among human beings.

According to ISO/IEC 15504, 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 set of standard processes of the organization according to its tailoring guidelines. A defined process of a project provides the basis for planning, performing, and improving the tasks and activities of the project [ISO/IEC-15504, 2003].

As stated earlier, there are several reuse opportunities on software process definition activities. In the last years, some works [Rombach, 2005; Armbrust et al., 2009; Washizaki, 2006a; Ru-Zhi et al., 2005] have suggested different approaches to try to accomplish process reuse, as we describe in the following paragraphs.

Software reuse can be achieved in different deep levels. In a shallow level, it is possible to visualize reuse as copy-and-paste activities or at most as individual component reuse. In a deeper level, it is possible to envision reuse of not only individual components, but of a set of collaborating components and the relationships among them (i.e., reuse of the whole architecture) [Garg et al., 2003]. We believe the same principle also applies to process reuse.

Process components are described and structured in different ways by different authors. Fusaro et al. [1998] argue that a process component can indicate a technique, a method, or a process. They present a framework to support the comprehension and evaluation of process components candidates to be used in software processes. Ru-Zhu et al. [2005] present a reuse-oriented process component representation framework for the description and classification of process components. They suggest a process component should be described in terms of a general description (users view, goal oriented), a specification description (process engineers view, implementation oriented), and a data description (project manager view, optimization-oriented). Gary and Lindquist [1999] present the Open Process Components (OPC) approach, in which process information is expressed in any formalism or language that defines the semantics of the component. OPC separates this information in three ways: process schema, process states and transitions, and process implementation. Fadila and Mohamed [2009] define process component as a software process model fragment, useable independently from the original software process model. The authors briefly present an approach to support the definition of process components derived from process models that were not defined for reuse or that were defined using different reuse approaches. SPEM [OMG, 2008], states that a process component contains exactly one process represented by an activity, and defines a set of work product ports that define the inputs and outputs for a process component.

Thus, there is not a general agreement on which information has to exist in a process component or which level of detail it must have. These decisions are usually based on the intended use for a component in each approach. All the described approaches provide some valuable concepts and are important to create a critical mass on the subject of process reuse. However, they do not consider other kinds of reusable elements, such as process features or process lines, which are described in the following paragraphs. Moreover, these papers do not present supporting tools and the description of the approaches is usually very brief.

Although acknowledging that using components to compose processes could bring benefits, starting a process definition using such small parts to compose large processes can still be difficult, error prone, and counterproductive. If we observe software product reuse, it is possible to note that one of the lessons learned from the efforts to achieve reuse on the last decades was that bottom-up reuse (i.e. using arbitrary components to build systems) usually does not work in practice [Bosch, 2000]. Well succeeded reuse programs should employ also a top-down approach, i.e. components are created in a way that they can be placed into a high-level structure defined by a software architecture [Bosch, 2000]. Therefore, combining the two approaches seems to be a better choice, i.e., to start from a higher abstraction level (top-down) and throughout the composition consider also the individual components (bottom-up).

We may consider the same principle applies to software process reuse. In other words, it is possible to argue that although process components play a central role on process definition, an approach that begins in a top-down way, with process components being put into larger reusable structures (e.g. lifecycle models, process architectures, process lines, or process templates) to be used in specific situations, is expected to leverage process definition activities. These larger reusable structures could be used to guide the definition of more complex process structures, composed of several basic elements. For example, higher-level components (e.g., composed activities) could be defined through the use of an internal architecture, establishing the way that basic elements (e.g., activities) could be placed into them. Similarly, even larger process structures (e.g., subprocesses, process phases, and defined processes) could be detailed based on a pre-defined architecture.

Thus, the concept of architecture is also being used in the context of software processes. According to Chrissis et al. [2006], 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 and complementary concept that can be considered a specialization of the process architecture concept is process template. It can be defined as a generic and reusable process model that establishes a starting point to the definition of a new process model [Reis et al., 2001]. 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 [Franch and Ribó, 2002]. These concepts are closely related to the concept of framework [Johnson, 1997], usually applied in software product development.

Sofware product lines [Clements and Northrop, 2001] 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 or features, through the arrangement of existent components. Features play an important role in this context. A feature can be defined as a logical unit of behavior that is specified by a set of functional and non-functional (i.e., quality attributes) requirements [Bosch, 2000]. A feature generally captures a considerable set of requirements and is, as such, used to group requirements, which simplifies requirements handling [Bosch, 2000]. Likewise, it is possible to consider that processes can be defined using similar ideas, i.e. processes can be instantiated from previously defined process components, in a way that each instance satisfies a set of chosen process features. So, software process lines (SPLs) are product lines whose products are software processes [Washizaki, 2006b; Rombach, 2005].

Rombach [2005] uses the term “SPL engineering” and states that the objectives of using it are – as in the case of all reuse approaches – increasing predictability, reducing cost and time, and reducing risk. According to him, the main characteristics of SPL engineering include: (i) Two separate development processes: the domain engineering process, which creates processes for reuse, and the application engineering process, which develop project-specific processes; (ii) A process repository: reusable processes at all abstraction levels are made available; (iii) A systematic reuse process: for each predefined choice of variabilities, the choice of process components is pre-defined (e.g., via empirically justified 'project maps'); (iv) A systematic process management process: for each exception (e.g. an unexpected behavior of the process occurs), it will be decided whether this exception will be factored into the generic process or not.

Washizaki [2006a] also proposed a 4-step bottom-up approach to establish SPLs that include similarities and variabilities. Likewise, Simidchieva et al. [2007] suggest a characterization of what might comprise a process family and introduces a formal approach to defining families based upon this characterization, where a process family is very closely related to a SPL. Ambrust et al. [2009] describe "SPL engineering" in more detail. Their approach establishes the three main steps to be followed to define a SPL in the context of an organization. The steps are: (i) Process Line Scoping, which identifies the range of characteristics that processes in the process line should cover; (ii) Process Line Modeling; which defines generic work products, commonalities, and variabilities; and (iii) Process Line Architecting, which defines a reference process architecture, based on the process line model. The authors also describe a 5-steps approach to deal with the first step of SPL engineering (i.e., Process Line Scoping). There are also other works that deal with this subject, reinforcing the described concepts [Hollenbach and Frakes, 1996; Lobsitz, 1996; Perry, 1996; Sutton and Osterweil, 1996; Jørgensen, 2000; Jaufman and Münch, 2005; Ternité, 2009].

It is possible to note that some works were published in the last years on software process reuse. This can be considered an indication that traditional product reuse techniques could really be successfully applied to software processes and also contribute with the creation of a critical mass about process reuse. Therefore, the described approaches were very important as a basis for the work being described in this paper. Some of the concepts described by them were used in our approach; although some works were concomitant with initial versions of our approach [Armbrust et al., 2009; Ternité, 2009; Barreto et al., 2008]. However, these works do not properly deal with the requirements of our approach, as described in Section 1. In other words, these works do not precisely describe how process reuse can happen in the multi-organizational context (i.e., SPIOs); they also do not consider legacy processes, detailing the steps to make them reusable and what information should be considered; and they also do not consider measures as an important part of the reusable elements, in order to properly select process components. Moreover, all these works focus on particular aspects of process reuse, such as SPL scoping, components classification, or the steps to create a SPL, and supporting tools are often very limited. Furthermore, none of these works provide evidence on the expected benefits and difficulties related to process reuse. At last, while there are many works on how to define languages and mechanisms to automate processes, there are less works aiming at providing knowledge and guidance on how to actually define processes, describing what to consider, which elements to choose, regardless of representation details.

3 An Adaptation of Software Product Reuse Concepts to Software Process Reuse

As described in Section 2, there are several different ways to describe and use the main software process reuse concepts. Thus, as a first step in the development of our process reuse approach, considering the goals described in Section 1, we have defined how to use the main concepts and their interrelationships, as shown on Figure 1.

To define these concepts, we have performed a literature search considering the most relevant aspects of process reuse, and how the topic has been addressed by the academia and industry, including the works described in Section 2. We have also considered the experience of many years of COPPE/UFRJ as a software process implementing organization [Montoni et al., 2008; Ferreira et al., 2008; Santos et al., 2007]. 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.

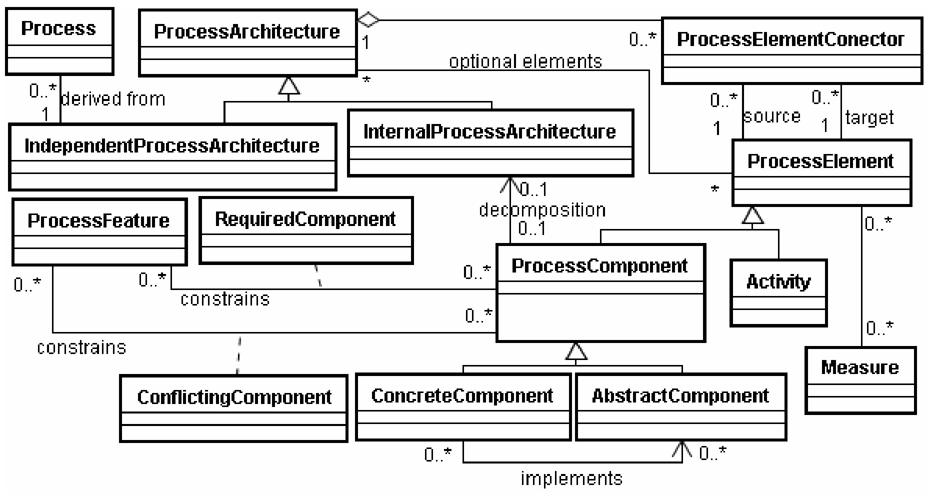


Figure 1: Simplified version of the core concepts and their relationships

In this approach, 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 [Florac and Carleton, 1999], 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 [Barreto et al., 2008]. *Activities* were used this way in our legacy processes, and due to this legacy demand we chose to keep the concept, maintaining its non-varying characteristic, as justified by the additional requirement 1 in Section 1. Thus, *activities* are usually used as building blocks for software *process components*, and can be reused only indirectly when they belong to a *process components*. A useful analogy is software components and software classes. Software components, like process components, are available in software reuse libraries for download and are internally composed by classes (assuming that they are implemented according to object-oriented paradigm). Classes, like process activities, are also reused indirectly, but it is uncommon to directly reuse an individual class due to its fine granularity, lack of encapsulation, and high coupling with other classes. Both *components* and *activities* encapsulate other kinds of process information, e.g., required or produced work products, roles, supporting tools to be used, supporting knowledge to aid their enactment, etc. A *measure* is used to measure *process components* and makes it possible to select *process components* based on their past behavior in a similar context. 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. To exemplify these concepts, we could have a *process component* “Elicit Customer Requirements” responsible for describing how requirements should be elicited. We could also have an *activity* “Establish Traceability Matrix” that could be considered not relevant for being directly reused, since it would always be performed the same way, and could be made part of the larger *process component*. Of course, if different ways of establishing traceability matrix become a reality in the future, this *activity* can be refactored and promoted to a *process component*.

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. However, the determination of how much flexibility is enough is a responsibility of the process engineer in charge of defining it, and will be very influenced by the scope defined for it. Section 5 provides more details on the steps to define reusable elements, including the definition of the scope and the verification of the reusable items, which tend to help in this determination.

We can consider a *software process line* (SPL) as a kind of software *process architecture*. However, since a SPL is a reusable asset, which aims at generating different software processes based on the same structure, a software *process architecture* is named as SPL only if it is able to generate different process definitions. The SPL concept is not shown in Figure 1 since it can be considered a role that process architectures may assume. A SPL may have *mandatory components*, which have to be present on every process instantiated from it. It may also have *optional components*, which may be present or not on the derivations of the SPL. At last, a SPL may have *variation points*, which are process components that may be instantiated in different ways, depending on the specific situation. For each variation point, there are some *variant components*, which are *process components* allowed to fit that variation point. It is important to note that even variation points can be optional. So, it is possible to have *mandatory variation points*, where at least one of its variant elements has to be present in the derived process; or *optional variation points*, in which it is possible to select none of the variant components, i.e., that component will not be included in the derived process.

A *process architecture* can be *internal* to a software *process component* (allowing its decomposition) or *independent* (representing a whole process), i.e. it can be used directly to derive defined *processes* or simplified *process lines*. A *process element connector* is used to connect several *process elements*. A *connector* has a source element, a target element, and some rule to associate them (e.g., start-end, end-end, etc). To illustrate, we could have a process architecture internal to the process component “Specify Requirements”. In this architecture, we could have the process components “Specify System Requirements”, “Specify Software Requirements”, and “Verify Requirements Specified” connected using “end-start” connectors, i.e., when the enactment of one component ends, the enactment of the following can start.

To make it possible to describe variabilities in a SPL, we have defined two kinds of process components: concrete and abstract. A *concrete component* does not allow any kind of variability. In other words, there are no remaining decisions to make, and it can be directly enacted, measured, and controlled in a software project. An *abstract component*, on the other hand, is a way to represent variabilities. It can have no internal *architecture*, in such a way that it can be directly substituted by *concrete components* (variants); or it can have an *internal architecture* that includes other *abstract components* or optional elements. In this kind of component, the definition is incomplete, and the component cannot be directly enacted. To allow the substitution of *abstract components* by adequate *concrete components* during processes definition, *concrete components* may determine which *abstract components* they implement. Only *process architectures* that have *abstract components* or optional elements are called *software process lines*. To exemplify these concepts, we could have a SPL to describe processes adherent to CMMI-DEV maturity level 2. This SPL would describe different ways to assure the quality of the work products. Thus, there could be an abstract component “Review Project Plan”, which would be a variation point in the SPL. To resolve the variation point, concrete components such as “Inspect Project Plan” or “Perform a Simple Review of the Project Plan” could be used.

A *process feature* in our approach is an adaptation of the concept of feature used in the traditional product lines domain to the context of processes. A *feature* can be seen as an aspect, quality, or characteristic that the *process* has to be compliant with.

It constrains the use of *components*, establishing a set of *components* that can (*required components*) or cannot (*conflicting components*) be used. Thus, it consists of a set of rules applied to *process components* that guides process definition based on process requirements. It is important to mention that in the current version of our approach, the semantics of a *feature* is defined by the user. In other words, it is used mainly as a high level mechanism for *component* selection. Therefore, the granularity of a *feature* will depend on its intended use, and the process engineer will have to determine the adequate level. Section 5 will discuss the definition of *features* in more detail, showing that reviews should be performed to ensure that they are adequately defined for each particular situation.

4 Software Process Reuse Scenarios

In this approach, we consider that process reuse can take place in different contexts, such as [Barreto et al., 2008]: (i) Software Process Implementing Organizations (SPIOs) – may need to define relatively similar processes to several different organizations; (ii) Software Development Organizations – may need to define their standard processes, or specialize these processes to situations that are common to the organization; and (iii) Software Projects – contribute to the components repository with data from process enactment, such as measures, improvement requests, etc.

Aligned with the ideas of Rombach [2005] regarding SPL engineering, in our reuse approach the existence of one or more reusable software process repository is needed 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 <u>with</u> reuse, i.e., the application engineering process), where components and other reusable items can be searched and used to compose processes. The repository needs to be fed and constantly evolved to offer to projects and organizations a useful and comprehensive set of reusable items (definition <u>for</u> reuse, i.e., domain engineering process).

Considering the context of SPIOs, a common situation faced by them is when a group of similar client organizations have a common goal regarding software processes improvement. For instance, they all may want to achieve a certain level of a maturity reference model such as MPS.BR or CMMI-DEV. This situation is common in Brazil, especially when organizations are starting software process improvement initiatives due to financial support provided by the Inter-American Development Bank (IDB) and by the Brazilian Ministry of Science and Technology. For these organizations, a software process improvement initiative in group tends to decrease costs and make it easier to share experiences. Since organizations in a group tend to be similar and are usually trying to achieve similar goals, the software processes under definition tend to be similar as well. Therefore, there is a great process reuse opportunity.

As shown in Figure 2, the SPIO establishes and feeds its repository with process components, process lines, measures, data from the enactment of components, among others (definition <u>for</u> 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 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 a subset of the SPIO´s components repository and also the supporting tools, to allow the organization to modify and evolve its processes and keep the culture of process reuse. In the provided repository there may be partially 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.

A software development organization may take advantage of the reusable processes defined by SPIOs and maintain its own reusable processes repository. Similarly to the scenario described in Figure 2, based on the specific needs of a process to be defined for a project, the organizational repository is used to compose project defined processes. Correspondingly to the SPIO context, organizations should also constantly feed and evolve their repository, aiming at making them progressively useful and complete in regard to the needs of the organization. It is important to note that organizations can define new standard processes or specializations of these processes to later reuse.

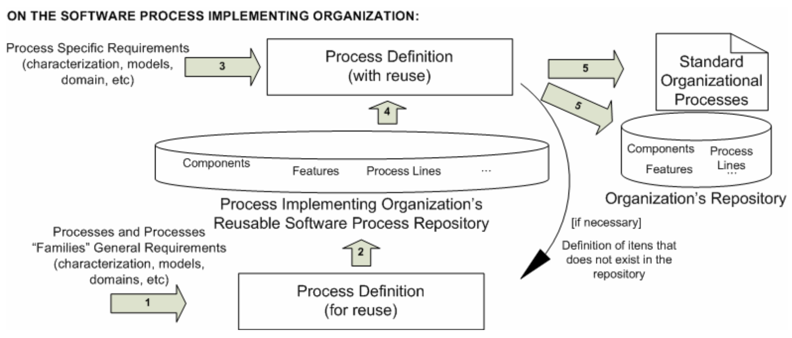

ON THE SOFTWARE PROCESS IMPLEMENTING ORGANIZATION:
Process Specific Requirements (characterization, models, domain, etc)
3
Process Definition (with reuse)
5
Standard Organizational Processes
5
4
Components
Features
Process Lines
...
Components
Process Lines
Features
...
Organization's Repository
Process Implementing Organization's Reusable Software Process Repository
Processes and Processes "Families" General Requirements (characterization, models, domains, etc)
[if necessary]
Definition of itens that does not exist in the repository
2
1
Process Definition (for reuse)

Figure 2: Software Process Reuse on SPIOs

It is also possible to establish a feedback between the organizational repository and the repository of the SPIO that originated it, as shown in Figure 3 (left hand side). 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 behavior (stability and performance) in that context (this is related to sub-objective 3 listed in Section 1). The SPIO can analyze the gathered information from organizations and provide back to the organizations benchmarking data. This allows them to compare the performance of their components with a baseline (i.e., mean and standard deviation) that comprises prior performances in similar contexts of other organizations. It is important to note that this strategy avoids privacy issues because no actual enactment data is provided. SPIOs can also provide from time to time updated data from its repository, such as new process components, new process lines, and so on. These communication requirements are being addressed by our supporting tools, specifically by Process Broker, which is described in Section 7. However, organizations do not necessarily need to be associated to a SPIO. If this is the case, they need to define their own reusable process elements.

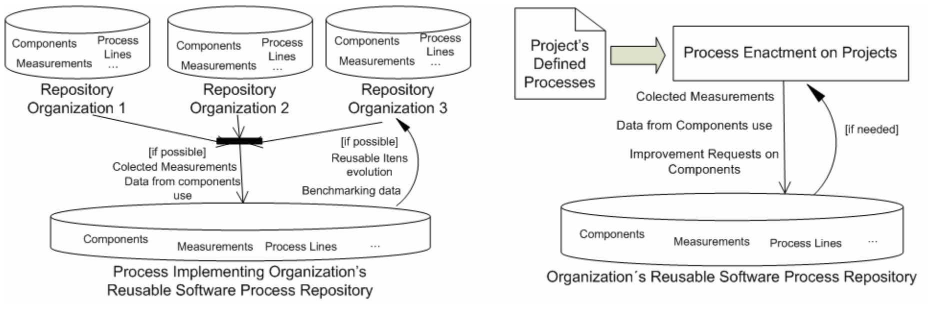

Components
Process Lines
Measurements
...
Repository Organization 1
Repository Organization 2
Repository Organization 3
[if possible]
Colected Measurements
Data from components use
[if possible]
Reusable Itens evolution
Benchmarking data
Process Implementing Organization's Reusable Software Process Repository
Project's Defined Processes
Process Enactment on Projects
Colected Measurements
Data from Components use
Improvement Requests on Components
[if needed]
Organization's Reusable Software Process Repository

Figure 3 Feedback loop and Project's defined processes enactment

On projects, process components are finally enacted. Projects' defined processes are enacted and contribute with data to the reusable process repository of the organization, such as measurements, lessons learned, and improvement requests. On-the-fly improvements can be made on the processes of the projects, if needed, through the replacement or maintenance of components in use. Figure 3 (right hand side) illustrates this context. After some time collecting measures from the enactment of components, it will be possible to consider this data to choose components during process definition. For instance, if a process being defined has a behavior requirement regarding costs, we could choose components that behaved in similar contexts in accordance to the requirements of the process being defined (related to sub-objective 3 in Section 1).

5 Defining Reusable Process Elements

In order to define reusable process elements (i.e. define processes for reuse), an SPIO or software development organization may adopt a bottom-up or a top-down approach (or a mix of them).

The top-down approach begins by defining process features and deriving the other reusable elements from them. In the bottom-up approach, we start defining process components based on previously defined processes, connecting and characterizing them until we have all the reusable elements. We used the top-down approach when there was a well defined set of requirements for the processes to be defined and the organization or SPIO did not have legacy processes (i.e., previously defined processes that were not defined for reuse and needed to be made reusable). In this situation it seems to be easier to define the reusable elements that are necessary and sufficient to fulfill the requirements. On the other hand, in some situations an organization or SPIO may want to make its legacy processes reusable and define reusable elements from them, to capture past situations and be prepared for new occurrences. In this case, we used a bottom-up approach (like a reengineering), which seemed to be more adequate, since the set of requirements that would be fulfilled by the reusable elements was not very clear at the beginning.

5.1 Defining Reusable Process Elements from Legacy Processes (Bottom-up Approach)

To perform the bottom-up approach we suggest the use of four main steps, as shown in Figure 4: (1) Define Process Components; (2) Define Process Features; (3) Define Process Lines; (4) Approve the Inclusion of the Defined Reusable Elements into the Repository. This approach was already used to define reusable elements from processes defined to the Software Engineering Laboratory at COPPE/UFRJ (hereafter *LENS*, which is the acronym used in Portuguese to refer to this laboratory), which is a software development organization. LENS has already experienced a successful MPS.BR appraisal in 2008, and its processes were considered adherent to MPS.BR Level E (which corresponds to an intermediate state between CMMI-DEV Levels 2 and 3) and aims at achieving MPS.BR Level A (which is similar to CMMI-DEV Level 5) in the near future. Projects at LENS are usually of two kinds: the development of academic software and the development of software products hired by customers from the industry. Project teams are usually small (not bigger than 10 people) and are mainly composed by M.Sc. and D.Sc. students.

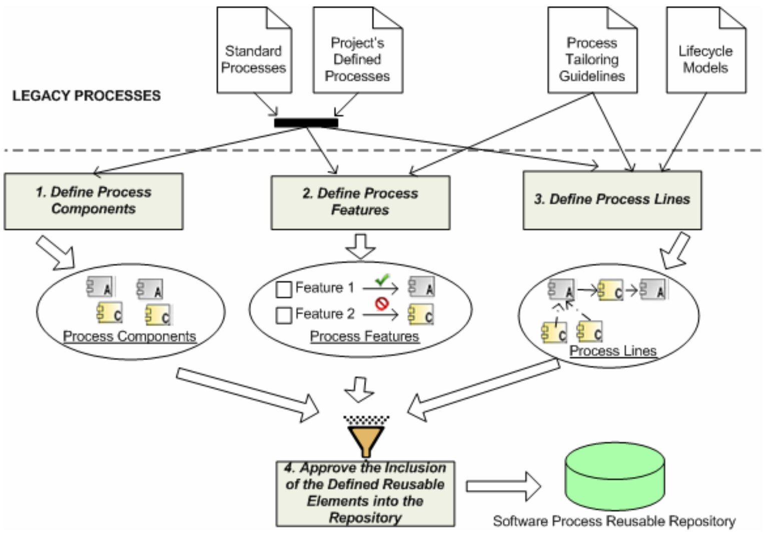

Standard Processes
Project's Defined Processes
Process Tailoring Guidelines
Lifecycle Models
LEGACY PROCESSES
1. Define Process Components
2. Define Process Features
3. Define Process Lines
Feature 1
Feature 2
Process Components
Process Features
Process Lines
4. Approve the Inclusion of the Defined Reusable Elements into the Repository
Software Process Reusable Repository

Figure 4 Bottom-up approach to define reusable elements from legacy processes

The goal of the first step of the bottom-up approach is to define process components that represent subprocesses that are potentially relevant for reuse. During this step we need to consider organization's standard process and also all processes that have already been defined to projects of the organization, since they were tailored from the standard processes, and these customizations are a good source of information. It is necessary to identify the parts of the standard process that were always present on projects and the ones that suffered variation. The parts that varied indicate that these are variation points of the process, since they could be performed in different ways. All these varying parts have to be considered abstract process components, which indicate the possibility to enact a subprocess in different ways. We also need to define concrete components to each possible way to enact the defined abstract components. Parts of the standard process that have never been changed can be modelled as concrete process components.

Considering the processes of LENS, we analyzed the documents that contained the rationale for adapting the standard processes (document produced during project process planning) of twelve projects. We detected some common processes customizations (which originated process components) such as: (i) the exclusion of some modeling activities, such as the elaboration of sequence diagrams, complementary diagrams or database project diagrams; (ii) inclusion of project monitoring and action plans management activities; (iii) inclusion of prototyping activities; and (iv) inclusion of activities to adapt C++ classes into JAVA. The fact that some groups of activities were removed or added in the processes of the projects indicates that these groups represent variations on the legacy standard processes, which could be represented as process components to foster process reuse. Thus, we created abstract components to indicate the variation possibilities and concrete components to represent each possible customization.

It is important to mention that although the definition of software process components is part of the supporting tools of our approach, as we describe in Section 7, we do not provide tools to support the identification of the common and varying parts of the legacy processes. This support is provided only through knowledge that is available to be consulted during the enactment of process components definition. In the context of LENS it was easy to perform this step, since the differences among the processes from the projects and the standard processes were described and justified, as required by the adopted maturity model. If this kind of information is not available, however, the use of tools that detect the differences between documents could help. Nevertheless, it would only be helpful if the notation and structure of the processes and their descriptions were uniform. Otherwise, the comprehension of the differences could require as much effort as detecting these differences without tool support. Thus, in this case a manual effort is necessary, equivalent to that existent in Domain Engineering, when there are legacy software products using different modeling notations and programming languages.

In addition, we can identify groups of activities that: (i) are in a level of detail (granularity) that makes it easy for them to be used in future definitions; (ii) are potentially relevant for collecting measures and having their behaviour analysed; (iii) already have data from their enactment collected, so that is possible to use this data to analyse their behaviour (for guiding their selection). Groups of activities with the described characteristics could become process components as well. Once again, considering the definition performed at LENS, all macro-activities originated process components. It was done because those elements were considered to be at a good granularity level, and also because of the existence of measurement data associated to them. This step was performed recursively, and when necessary more specific components were also created.

As an alternative to the approach just described, which can be considered reactive since it is based on past events, a proactive approach can also be adopted, aiming at anticipating future needs. Thus, we have to search for process elements from the legacy processes that have good reuse potential, even if they have not yet suffered variation. It is possible that variation has not happened yet due to the fact that the required definitions and guidance information were not available. At LENS we identified some of these opportunities. For instance, activities related to calculating the size of the software led to abstract components, with at least two related concrete components: one to use Function Points and another one to use Use Case Points.

It is important to highlight that information from legacy activities or from other kinds of legacy process elements that originated process components must be considered on its definition. Thus, information such as measures, entry and exit criteria, required and produced work products, among others, have to be adapted to become suitable not only to a single activity, but to the defined component as a whole.

Moreover, it is expected that within some time, actions that were once performed in a certain way (and that probably originated a concrete component) may be modified. Thus, a more general abstract process component can be created and the previous concrete component can become one of the variants of the newly created abstract component. Therefore, the first componentization can continuously evolve from constant software process improvement.

To summarize, we suggest some questions that, when positively answered, can make a legacy process element (e.g. activity, macro-activity, subprocess, etc) from the organizational standard process to become a process component: (i) Is it in such a level of detail (granularity) that it may be easily reused later? (ii) Is it potentially relevant for collecting measures and having its behaviour analysed? (iii) Does it already have data collected from its enactment, so that it is possible to use this data to analyse its behaviour (to guide its selection)? (iv) Has it already been modified (tailored) to be used on projects? (i) Even if not already modified to be used on projects, does it have a good potential to be performed in different ways? It is important to mention that these questions are guidelines that have to be interpreted by process engineers. Since the answers can be influenced by each particular situation in which the approach is used, the supporting tools do not support this analysis. Once these decisions are made, the resulting process components can be defined using the supporting tools, as described in Section 7. These guidelines are provided as knowledge related to the enactment of the "Define Process Components" task.

The next step aims at characterizing the defined process components using process features. Through these features it is possible to establish several kinds of traceability to process components. Process features constrain which components can be chosen throughout process definition.

One of the main sources of information to define process features is obtained from the standard processes tailoring guidelines. These guidelines establish in which scenarios each adaptation (tailoring) may be performed. Thus, it is possible to define process features that represent the different ways a process can be tailored and associate them to the process components that should or should not be selected if the feature is selected. In Figure 5 it is possible to see some tailoring guidelines, some process features that could be originated by them, and the relations among features and process components. The analysis of the tailoring guidelines and their mapping into process features is supported only through guiding information that is available during the enactment of "Define Process Features", since this interpretation is essentially human. Once again, our supporting tools provide the means to define process features, but the way they are modeled is dependent on the users of the approach, although we provide guidance on how it could be done.

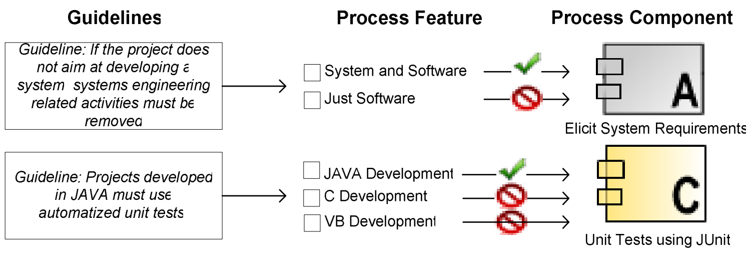

Guidelines
Process Feature
Process Component
Guideline: If the project does not aim at developing a system systems engineering related activities must be removed
System and Software
Just Software
A
Elicit System Requirements
Guideline: Projects developed in JAVA must use automatized unit tests
JAVA Development
C Development
VB Development
C
Unit Tests using JUnit

Figure 5: An example of the use of standard processes tailoring guidelines to define process features

It is also important to consider the main characteristics possessed by the legacy processes and make this knowledge explicit through process features as well. For instance, the processes of LENS are adherent to Level E of MPS.BR. Therefore, we can associate process components originated from these processes to process features such as: Adherence to MPS.BR Level E, Adherence to MPS.BR Level G (a lower level), Adherence to CMMI-DEV Level 2 (which is a subset of MPS.BR Level E), etc. Thus, it is important to register the features the legacy process already had. Depending on the way features are used, it is possible to associate process components to other kinds of process features, such as: disciplines (e.g. project planning, requirements), development techniques and methods (e.g. function point analysis, inspections), and development paradigms (e.g. structured, object oriented). A checklist with a set of possible process features and their respective kind based on common process characteristics can be used to support feature identification, so that no characteristic from legacy process is forgotten.

Some organizations, such as LENS, already have a set of factors that has to be considered while characterizing a software project and this information is used to tailor its processes. Examples of these factors are: development paradigm being used, application domain, and also factors related to the problem, product, or project. This set of factors is also a great source of information to define process features.

Once process components are defined and classified to the organization, it is important to consider different alternatives to structure, relate, and order these components, so that they can be used to derive processes. To do so, software process lines can be used. To define a process line, a top-down approach is suggested, beginning from the establishment of the skeleton that the processes derived from it will have. Good starting points are lifecycle models used in the organization. Each different model can be considered a different process line (or at least a first step to define a more complex process line), in which the relationships among components and the order of their execution will be already defined.

During software process line definition it is also important to consider the standard processes tailoring guidelines. These guidelines can support the determination of the optional elements and variation points of the process line. The different adaptations to the standard processes performed so far can also be used. It is important to note that process features definition and process line definition can be performed concurrently, since one step can complement the execution of the other.

The last step comprises a review performed by the relevant stakeholders. Candidate process components, features and process lines are reviewed before being added to the reusable process repository. Guidelines to accept reusable elements may be defined and used. The reviewers must agree that the candidate items are relevant and adequate for being reused. The approved items may finally be included into the repository and then become available to be used by the organization to define processes from them.

As described throughout the text, the bottom-up approach described here has been already successfully used to define reusable processes at LENS. Moreover, it was also used to make the processes of a CMMI-DEV Level 3 Brazilian organization reusable, which was considered by the organization a first step to properly select subprocesses (process components) that would be statistically controlled [Florac and Carleton, 1999]. Therefore, we have already some indication that this approach can be successfully used to define processes for reuse, considering existing legacy processes. However, we still intend to use it in different contexts (e.g., to make the processes defined by COPPE/UFRJ SPIO reusable) in order to better evaluate it. The reusable process elements defined at LENS will be important in its initiative to achieve MPS.BR Level A, since higher maturity levels have specific requirements related to process definition, as described by sub-objective 3, in Section 1.

5.2 Top-Down Approach to Define Reusable Process Elements

As mentioned earlier, sometimes an organization or SPIO need to define reusable processes, but does not have legacy processes available to address the specific situation, but has a well defined set of requirements for the processes to be defined. In this context, a top-down approach is suggested. Figure 6 shows the main steps of this approach. We consider its main and final goal is to define software process lines (SPLs). However, throughout SPL definition, other reusable elements may need to be defined as well.

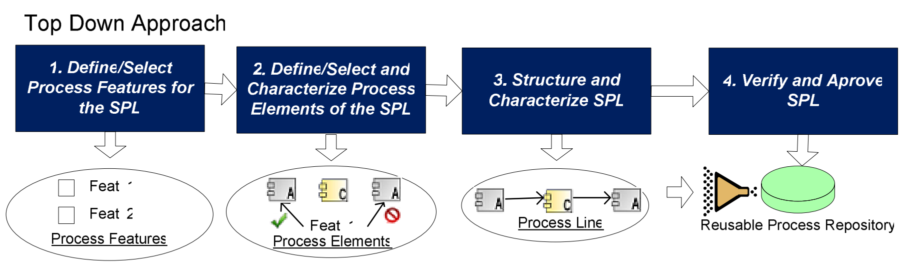

Top Down Approach
1. Define/Select Process Features for the SPL
2. Define/Select and Characterize Process Elements of the SPL
3. Structure and Characterize SPL
4. Verify and Aprove SPL
Feat 1
Feat 2
Process Features
A
C
A
Feat 1
Process Elements
A
C
A
Process Line
Reusable Process Repository

Figure 6 Top-Down Approach to Define Reusable Process Elements

This top-down approach was also already used to define a SPL focused on acquisition processes, as described in [Barreto et al., 2010]. As mentioned in Section 1, COPPE/UFRJ is a SPIO in Brazil. Several organizations have already requested COPPE/UFRJ the definition of software acquisition processes. These organizations usually have difficulties to map the possible acquisition scenarios to software processes. Thus, COPPE/UFRJ has decided to create a SPL focusing on these processes, to model their commonalities and variabilities and also take advantage of the reuse opportunities. The requirements of the SPL were defined by a software acquisition specialist, based on COPPE/UFRJ previous experiences, on a literature search focusing on acquisition processes, and also on maturity models and standards [Barreto et al., 2010].

First, considering that the set of requirements that has to be addressed is available, we need to map them to process features. These features will guide the definition or selection of the other elements. In other words, we have to define the scope of the SPL, similarly to "Process Line Scoping" of Ambrust et al. [2009], establishing which situations are going to be treated with the future SPL and to model these situations as process features. Figure 7 exemplifies this step. As mentioned earlier, this kind of interpretation is essentially human, and therefore we do not provide a tool to perform this mapping between requirements and features. However, this knowledge can be stored as descriptive information of the feature being created.

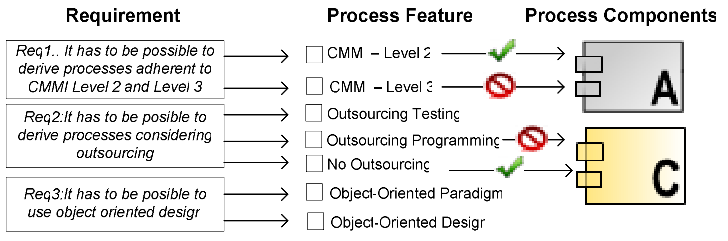

Requirement
Process Feature
Process Components
Req1.. It has to be possible to derive processes adherent to CMMI Level 2 and Level 3
Req2:It has to be posible to derive processes considering outsourcing
Req3:It has to be posible to use object oriented design
CMM – Level 2
CMM – Level 3
Outsourcing Testing
Outsourcing Programming
No Outsourcing
Object-Oriented Paradigm
Object-Oriented Design
A
C

Figure 7 Defining Process Features

Considering the SPL defined by COPPE/UFRJ, 37 process features were created and grouped in process feature kinds. Features definition was based on the requirements defined for the SPL, considering maturity models, characteristics of the target organizations, and several specific needs of acquisition processes. For instance, "Compliance with maturity models" was considered a process feature kind, with the features "CMMI – Level 2" and "MPS.BR – Level F" associated. "Suppliers Selection" was also considered a feature kind, with the features "Suppliers chosen from a list of previously authorized suppliers" and "Suppliers chosen considering technical and commercial criteria" associated.

Once the scope is established, it is necessary to define (or select, if already available) the process elements that will be used to provide the features that were selected in the first step. Thus, we need to identify what will be common to each process derived from the process line and what will be the varying parts. The common parts can be modeled as concrete components, while to deal with the varying parts we define abstract components describing the generalized expected behavior. Each abstract component that is put into the process line become a variation point, and needs variant components to realize them. Therefore, we need to define or select concrete components to represent all the necessary process variants. Once the required process components are available, we need to map them to the process features selected on the first step, so that it will be possible to know which variant to use depending on the selected feature, during SPL derivation. It is important to guarantee that there are variants to be associated with each one of the possible defined features. In the SPL defined by COPPE/UFRJ, 28 process components were defined in this step to comply with the selected process features. This step was performed in several iterations, together with the next step *Structure and Characterize SPL*.

The next step is to actually structure and characterize the SPL. We have to define interactions among the selected components and also choose what components will be optional in the SPL. The structure of the generated processes, including the connections among the elements has to be defined. We can also map process features directly to the SPL, to explicit in which situation this process line can be selected or not. If there is already a SPL similar to the one that is needed, instead of starting a new SPL from scratch, we can derive the existing SPL to define the new one. At the end of this step, a first version of the SPL is ready.

A last step was also introduced to check if the defined SPL seems to be adequate to fulfill its requirements. During this review, the relevant stakeholders of the organization will use their experience and tacit knowledge to verify the components used, the set of process features, the sequencing of the components, the available variants, among others, trying to agree on the best possible solution. To perform this step we provide a checklist that can be used to guide the review. The checklist is comprised of 24 questions about the process features, components, and the process line defined, such as: Were the defined process features adequately grouped into process feature kinds? Are the defined process features in an adequate level of detail? Are there sufficient variant concrete components for each abstract component? Are the interfaces among process components being respected in the SPL? At COPPE/UFRJ, a senior process engineer that coordinates the SPIO performed the review. Some change requests were made, comprising inconsistencies among components, inadequate granularity of some components, and also the description of some elements that was not clear. This step was fundamental to try to guarantee the quality and adequacy of the defined SPL. During the definition of the SPL, some of the discussed steps can be performed more than once, since the process engineer can find out better ways of modeling something that has already been done.

As described earlier, the top-down approach has already been successfully used in a real scenario [Barreto et al., 2010]. Although processes have not yet been derived from the SPL, we hope this will happen soon. Aiming at learning from the use of the approach, we asked the participants (the process engineer responsible for defining the SPL and the one responsible for reviewing it) to fill in a form about their impressions on the benefits and difficulties of using the proposed approach. The participants mentioned that the approach was suitable to model the several variabilities that exist on software acquisition processes and also to explicit knowledge, including lessons learned from domain experts. Moreover, the use of the defined concepts was considered adequate to clearly model processes and their variability. Some improvement opportunities were also suggested, such as the realization of a small training session on the concepts of the approach prior to its first use (due to some initial difficulties noticed) and some suggestions regarding the addition of more information to describe process components. These suggestions are being analyzed and will be important to improve our approach.

It is important to mention that the top-down approach may be expensive, since it may demand a great effort in order to consider the needs of not one, but of several processes that could be defined. The bottom-up approach tends to be cheaper, since most of the processes are already defined, and only need to be adapted. Therefore, it is necessary to carrefully define the scope of the reusable elements to be defined, so that the initial effort is compensated through their reuse. Although SPIOs seem to have a greater reuse potential than development organizations, and therefore they may experience more benefits, it is also expected that development organizations can take advantage of this kind of approach, as also shown by [Armbrust et al., 2009] using a similar top-down approach.

Our research group is still working on the definition of SPLs for particular situations. Therefore, we believe we are going to have more feedback from the use of this approach in the near future.

6 A Survey on the Expected Benefits and Difficulties on Software Process Reuse

After the definition of the main concepts of our approach and also the usage of some parts of it, as described in Section 5, we needed additional information to better choose our next steps. Since the main potential users of our approach are process engineers, i.e., professionals that are responsible for defining software processes, we wanted to capture their expectation regarding the benefits and difficulties related to the main software process reuse concepts being used in this work (i.e., software process components, software process lines, and process features). It was necessary to investigate which are the expected benefits of each parts of our approach. For instance, if process engineers expect to improve the quality of their defined processes, do all the reuse approaches (process components, process lines, and features) have similar significance? In addition, it was also necessary to determine the way to proceed with our research that better addresses the most relevant concerns of our potential users. For instance, we could develop supporting tools, certify process components, or investigate psychological aspects, among others, in order to properly address the expected difficulties mentioned by our potential users.

Furthermore, if software processes are software too, as stated by Osterweil [1987], it is expected that software processes reuse and software products reuse have similar expected benefits and difficulties. For instance, if “diminish rework” is an expected benefit for software products reuse, it may also be expected for software processes reuse. The same may apply to difficulties, such as “inadequate supporting tools”. So, in order to capture the opinions of process engineers, a subset of the main expected benefits and difficulties for product reuse initiatives were adapted to the context of software processes and considered. In this context, in order to get the opinions that we wanted and also check the similarities between software process reuse and software product reuse, we planned and executed a survey with the following goal:

To analyze *the answers of the survey* **with the purpose of** *characterizing the expected level of benefits and difficulties expected from process reuse approaches* **with respect to** *similarities with common software product reuse expected benefits and difficulties* **from the point of view of the** *researcher* **in the context of** *experienced process engineers.*

The selection of subjects was based on their experience on software process definition. In order to gain more confidence on the results, we chose to have a reduced number of specialists rather than hundreds of inexperienced participants. Thus, the set of subjects was composed by 23 experienced process engineers from both industry and academia. Considering academic degree, 8 (35%) participants hold Ph.D. degree, 10 (44%) hold M.Sc. degree, and 5 (21%) hold B.Sc. degree. 9 (39%) worked only in industry, 7 (30%) worked only in academia, and 7 (30%) worked in both environments. Moreover, there were participants from 7 different academic institutions and from 12 different organizations from the industry. There were participants from 7 different states of Brazil, comprising all its regions. Considering all these aspects, we believe that our set of subjects can be considered comprehensive, even with only 23 participants.

The survey asked subjects to rank the expected benefits and difficulties of software reuse approaches according to the following Likert interval scale levels: high, medium, low, and none, and mapped respectively to 3, 2, 1, and 0. Participants also rated their experience on software process definition, software process implementation (deployment), software products reuse, and software process reuse according to this Likert scale. Table 1 summarizes the experience levels of the subjects. It is possible to note that the participants are more experienced on processes than on reuse (which represents more adequately the potential users of our approach), and that even experienced process engineers have less experience on process reuse, which means process reuse has not been accomplished very often even by experienced process engineers. We also collected their experience in years. As an example, considering the professionals with high level of experience on software process definition, the average experience in years was approximately 10 years, while the professionals with medium level of experience on software process definition, the average experience in years was around 4 years.

Level of Experience	**Software Process Definition**	**Software Process Implementation**	**Software Product Reuse**	**Software Process Reuse**
High	13	13	00	01
Medium	10	08	11	11
Low	00	02	11	11
None	00	00	01	00

Table 1 Histogram of the survey participants experience level

Table 2 summarizes the expected benefits from the adoption of each process reuse approach considering all the participants. The results presented in this table are:

mode (in bold) followed by mean (μ) and standard deviation (σ). The benefits listed were adapted from some of the most common software product reuse expected benefits [Lim, 1994; Mili et al., 1995; Prieto-Diaz, 1993; Pfleeger, 2001], but considering software process definition (e.g. diminish rework *on process definition*).

It is possible to note, after analyzing Table 2 that a high level of benefits is expected from the use of each process reuse approach in regard to the great majority of the listed common software product reuse benefits. Moreover, we can see that the highest benefits are expected from the use of software process lines, considering it had the highest mode and mean values together with the lowest standard deviation. Among the expected benefits listed, "Improve quality" had the lower level of expectation, although it was also high. It is probably due o the fact that only experienced process engineers were considered. The results show that there is an indication that these common expected benefits from product reuse are really also expected from process reuse. Moreover, there is an indication that process engineers really consider that using the described approaches can bring great benefits for process definition activities. In addition, since process lines were the most promising approach, we need to focus even more on the support and guidance to use software process lines. Furthermore, although results obtained are close to each other, Table 2 gives us some indication on what are the most relevant approaches to be used in order to achieve each of the expected benefits. For instance, if the main goal of the adoption of process reuse is to diminish rework, according to these process engineering specialists, software process lines are more useful than process components alone, which are more useful than software process features. Therefore, it is possible to prioritize the approaches to be used, depending on the expected benefits.

Expected Benefit / Process Reuse Approach	**Software Process Components**	**Software Process Lines**	**Software Process Features**
Increase productivity	**High** μ = 2.61 σ = 0.58	**High** μ = 2.78 σ = 0.52	**High** μ = 2.61 σ = 0.66
Diminish rework	**High** μ = 2.57 σ = 0.59	**High** μ = 2.74 σ = 0.62	**High** μ = 2.52 σ = 0.73
Improve quality	**Medium** μ = 2.39 σ = 0.58	**High** μ = 2.48 σ = 0.79	**High** μ = 2.48 σ = 0.73
Diminish costs/effort	**High** μ = 2.48 σ = 0.67	**High** μ = 2.65 σ = 0.57	**High** μ = 2.48 σ = 0.67
Diminish time required to perform the activity	**High** μ = 2.43 σ = 0.66	**High** μ = 2.65 σ = 0.57	**High** μ = 2.48 σ = 0.67

Table 2 Expected benefits from software process reuse

Table 3 summarizes the expected difficulties from the adoption of each process reuse approach. The results presented in this table are also: mode (in bold) followed by mean (μ) and standard deviation (σ). The difficulties listed were adapted from some of the most common software product reuse expected difficulties [Sherif and Vinze, 2003; Mili et al., 1995; Prieto-Diaz, 1993; Pfleeger, 2001], but considering software process definition.

Expected Difficulty / Process Reuse Approach	**Software Process Components**	**Software Process Lines**	**Software Process Features**
Difficulty to identify, retrieve and modify reusable elements	**Medium** $\mu = 2.04$ $\sigma = 0.71$	**Medium** $\mu = 1.91$ $\sigma = 0.67$	**Low** $\mu = 1.52$ $\sigma = 0.67$
Insufficient quality of the reusable elements	**Medium** $\mu = 1.82$ $\sigma = 0.73$	**Low** $\mu = 1.73$ $\sigma = 0.83$	**Medium** $\mu = 1.59$ $\sigma = 0.73$
Existence of psychological, legal or economic barriers	**Medium** $\mu = 1.83$ $\sigma = 0.78$	**Medium** $\mu = 1.78$ $\sigma = 0.80$	**Medium** $\mu = 1.74$ $\sigma = 0.81$
Necessity to create reuse incentives	**High** $\mu = 2.09$ $\sigma = 0.85$	**Medium** $\mu = 1{,}96$ $\sigma = 0.77$	**Low** $\mu = 1.83$ $\sigma = 0.83$
High adoption (deployment) cost	**Medium** $\mu = 1.91$ $\sigma = 0.73$	**Medium** $\mu = 2{,}13$ $\sigma = 0.63$	**Medium** $\mu = 1.96$ $\sigma = 0.64$
Inadequate supporting tools	**Medium** $\mu = 2.26$ $\sigma = 0.69$	**High** $\mu = 2.52$ $\sigma = 0.59$	**High** $\mu = 2.48$ $\sigma = 0.67$

Table 3 Expected difficulties from software process reuse

It is possible to note, after analyzing Table 3, that the level of difficulties expected from the use of each process reuse approach is lower than the level of benefits expected from them. However, the difficulties from product reuse also seem to apply to software process reuse. We can see that the "Inadequate supporting tools" is the biggest concern of the participants. This result shows us we really have to provide adequate supporting tools; otherwise process reuse is expected to be seriously threatened. It is also possible to see that the use of process features seems to have a smaller potential to bring difficulties to process definition. Maybe, this is due to the fact that process features goal is exactly to simplify and provide high level selection of process components. These results were very important to give us more confidence on the next steps of our research. For instance, if insufficient quality of the reusable elements were the most relevant difficulty, we would have to work more on the quality of the reusable elements, providing some kind of certification for them, for example. However, according to the obtained results, we decided to focus on the development of the supporting environment, described in Section 7. This is worth to notice that this technical aspect was considered more relevant than other non-technical aspects by the process engineering specialists. This may be due to the fact that process reuse is a young discipline if compared to product reuse, which dates from 1968 [MCIlroy, 1968]. Probably, process reuse still needs comprehensive tool support to be able to face other non-technical issues in the future.

Considering the results presented on Table 2 and Table 3 together, if software process lines are the most promising approach and inadequate support tools is the great concern, software process line supporting tools deserve special attention.

In order to learn more from our results, we also grouped the results in three different groups: (i) participants from the industry; (ii) participants from the academia; and (iii) participants from both industry and academia. We analyzed the obtained results when considering each group alone in comparison to the results considering all the participants, using the average. In general, in regard to the expected benefits, results in each group were very similar to the ones considering all participants. The biggest difference was found in the academia professionals group, which considered that software process components is about 10% less beneficial to "Increase productivity", "Diminish rework", and "Improve quality".

On the other hand, when analyzing the expected difficulties, there was a more significant difference. In general, the group from the industry considered the difficulties more relevant then the group from the academia, while the group from both had similar results to the ones of the group from the industry. The group from the academia considered "Insufficient quality of the reusable elements" from 18% to 35% less relevant for all reuse approaches, probably considering they would be able to create reusable elements with the adequate quality level. However, the same group considered "Existence of psychological, legal, or economic barriers" 18% and 23% more relevant respectively for process components and process features. The group from the industry considered "Difficulty to identify, retrieve and modify reusable elements" 16% more relevant considering both process lines and process features; and "Insufficient quality of the reusable elements" from 5% to 15% more relevant in all the reuse approaches.

These results can be an indication that participants from the industry are more worried about the practical aspects of process reuse, such as supporting mechanisms or the quality of the elements. This may be related to the varying level of expertise expected from the professionals that are supposed to perform these activities in the industry, where there is probably a bigger need for support and guidance. It can also be an indication that the participants from the academia are more worried about non-technical aspects of process reuse, like psychological barriers. It may be related to their understanding that in several other disciplines, such as project management or even software products reuse, usually the main problems are related to non-technical aspects.

Finally, we acknowledge surveys are a limited approach, since it considers only people's opinion on a subject and the opinions may not be necessarily correct. To try to alleviate this threat, only professionals that had already performed process definition activities on real environments and that possessed deep theoretical knowledge were asked to participate. Moreover, the wrong interpretation by the subjects about what has been asked through the survey could also threaten the validity of the study. To deal with this threat we have added explanatory tables into the survey form in order to precisely define: (i) how to rate subjects' experience, exemplifying each possible value; (ii) how to rate the expected benefits and difficulties; (iii) a description of each considered reuse approach, with examples; and (iv) a description of each benefit and difficulty listed.

We ran a pilot version of the survey with two subjects, in order to determine whether they would have difficulties, and then we sent the survey for the remaining subjects. We also considered the risk of the subjects' lack of interest in the study, since some of the participants could execute the study in an uncompromised way. They could also give false opinions, worried about how the results would be used. To deal with these risks, we clearly stated the goals of the study and explained that results would be used anonymously only in the context of an academic research, and thus honest opinions were wanted. Moreover, we also made the survey simple, so that a subject would take no longer than 15 minutes to answer it. We also avoid insisting with the subjects to return the surveys, in order to avoid uncompromised responses. With regard to external validity, i.e., the possibility to generalize these results to different contexts, we tried to consider a comprehensive sample, with participants from industry and academia, from different institutions, and from different locations. However, we are not able to generalize the results to different contexts, and further research is still necessary, for example, for different countries.

In addition to the results of the survey, we consider that results from other researches being done [Armbrust et al., 2009; Washizaki, 2006b] on similar topics are also indications that process reuse techniques bring benefits to process definition. Moreover, the survey is not the only mean to evaluate our approach. The use of the approach in real environments has already brought good benefits for users and for the improvement of the approach itself. We are still going to use the approach on several other real environments in order to constantly evolve it. We will repeat the survey in the future trying to evaluate benefits and difficulties observed from the use of our approach. Thus, we will be able to evaluate if the expected benefits are being provided specifically by our approach and whether the expected difficulties are being properly addressed.

7 An Environment to Support Software Process Reuse

To accomplish systematic process reuse without supporting tools may be a very difficult task. One of the difficulties often mentioned, related to software product reuse, is the lack of adequate support mechanisms. In the context of process reuse the same difficulties are expected. As we presented in Section 6, in the point of view of experienced software process engineers, the lack of adequate supporting tools was the difficulty with higher probability to threaten process reuse, among the difficulties considered. Thus, in order to allow process reuse to happen we developed a software process reuse environment. It supports the main steps of our approach and is detailed in Figure 8. Therefore, this environment is actually the support of the approach described in this paper and implements the concepts and activities described.

At the center of Figure 8 it is possible to see the reusable process repository, which was already described earlier in Section 4. To implement it we used a relational database that models all the concepts described in Section 3, among others.

At the left-hand side of Figure 8 it is possible to see the domain engineering activities (definition for reuse) that are accomplished by the most experienced process engineers, who make their knowledge explicit through the definition of reusable process elements, as described in Section 5. Actually, this domain engineering process corresponds to the steps described in Figure 4 and Figure 6, except for the verification activities, that are performed through checklists.

Continuing on Figure 8, at its right-hand side it is possible to see the application engineering activities (definition with reuse) that can be accomplished even by less experienced process engineers, since knowledge acquired from the most experienced process engineers is already available to be reused by them. Thus, process engineers can derive new process lines from the existing ones, so that more specific variant process lines are available. They may also define processes from the available process lines. Process definition based on a process line consists in choosing a concrete variant at each variation point and in defining if the optional elements are going to be present in the resulting process or not.

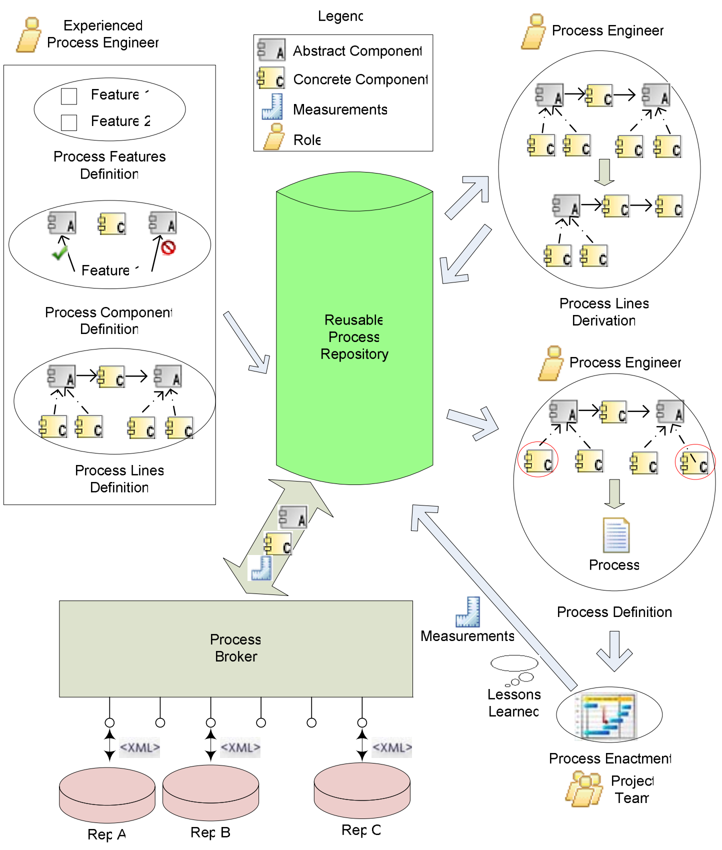


Figure 8 An Environment to Support Software Process Reuse

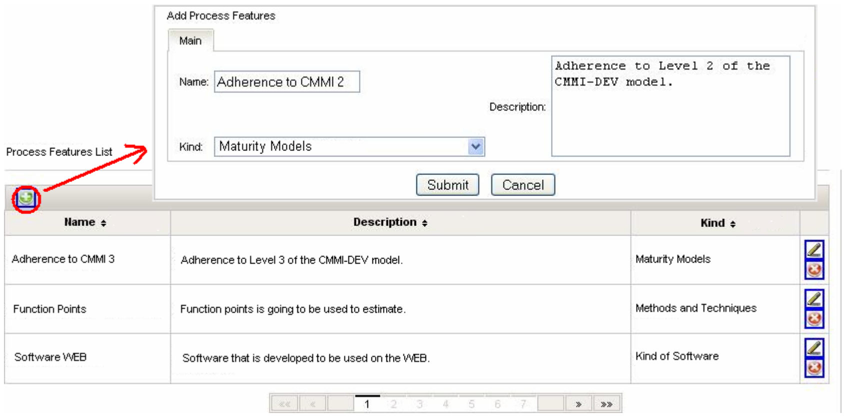
Once a process is defined, it can be enacted on projects. Projects team execute the process of the project and contribute to the reusable repository through the collection of measures, lessons learned, process improvement requests, among others, as mentioned earlier.

Other important element of our environment is Process Broker (as shown in Figure 8), which is responsible for managing the communication among reusable process repositories. This communication can happen among software process implementation organizations (SPIOs), among software development organizations, and among software development organizations and SPIOs, as show in Figure 3. Process Broker externalizes the structure of a reusable repository through the definition of a XML Schema. Thus, elements can be exported in the established format, and have to be in the same format to be imported.

Once the overall characteristics of our environment were described, we provide a guiding example of how our supporting tools could be used in order to make it easier to understand. Therefore, let us define an exemplary situation where a group of five organizations gets in touch with COPPE/UFRJ. Their goal is to achieve CMMI-DEV Level 2 as a first step in their software process improvement initiatives. They do not have previously defined processes, and do not perform several practices required by the model. So, they want COPPE/UFRJ to help them on the definition of a process, on training to correctly enact the process, and on mentoring during the process enactment. We are focusing only on the process definition related activities. Since COPPE/UFRJ will need to define similar processes, it can take advantage from software process reuse, and may define and use a SPL. Doing so, it tends to be easier to define the processes for the five organizations and, in addition, it tends to be even easier to define processes in the future for other organizations that require similar processes (which is a very likely situation). Thus, let us consider that COPPE/UFRJ, acting as a SPIO, chose to use the top-down approach, described in Section 5.2.

The first step consists in the definition of the process features to be used. Since all organizations want to comply with CMMI Level 2, this is an important feature to be considered. Moreover, let us assume also that the organizations use or will use different project estimates techniques. For instance, one organization wants to use the project manager experience to support its estimates. A second organization is having problems with its estimates, and wants to try another technique. Since use cases play a central role in the way this second organization develop software and they have already been trained on Use Case Points, they want their estimates based on this technique. The other organizations, however, have Function Points certified professionals, and thus want to use it to estimate projects. Due to that, it is possible to notice three other features, based on the way estimates are going to be performed. Figure 9 shows how process features can be defined using our supporting tools.

Once process features are defined, it is necessary to define the process components that will fulfill those features. So, let us consider a small portion of the SPL being defined: a portion that aims to plan the process of a project, plan the project itself, and assess the quality of the produced work products. Therefore, following our approach, COPPE/UFRJ experienced process engineers chose to create 8 process components. The first component, “Plan the Process”, was considered by the process engineers as a component that will not suffer variation. So, it was defined as a concrete component and will be included in the 5 derived processes exactly as defined. The component “Establish Project Estimates” was defined as an abstract component, since this component can be instantiated in a process in three different ways, through the choice of one of its three concrete varying components, which also had to be defined ("Use Case Points", "Function Points", and "Experience-based"). In order to make the example simpler, let us consider that the remaining process components were all concrete components: "Generate Project Plans", "Review and Obtain Commitment with Project Plan" and "Assess Work Product Quality". Figure 10 illustrates how process components can be defined using our tool.



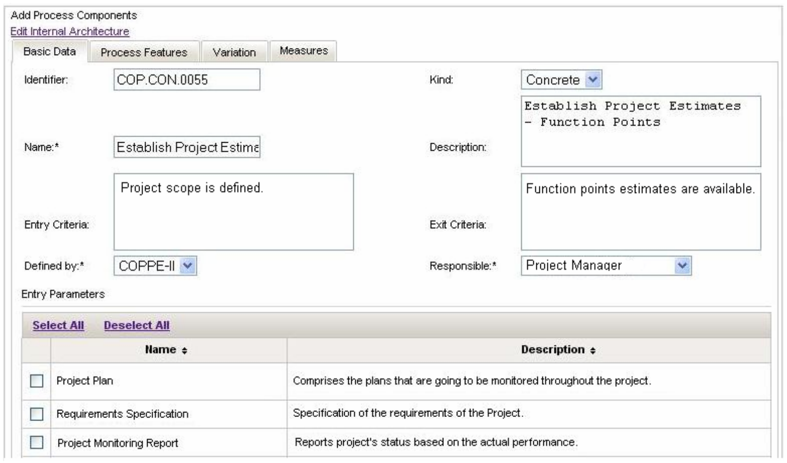
Name: Adherence to CMMI 2

Description: Adherence to Level 2 of the CMMI-DEV model.

Kind: Maturity Models

Name	Description	Kind
Adherence to CMMI 3	Adherence to Level 3 of the CMMI-DEV model.	Maturity Models
Function Points	Function points is going to be used to estimate.	Methods and Techniques
Software WEB	Software that is developed to be used on the WEB.	Kind of Software

Figure 9 Process Features Listing and Adding Using the Support Tool



Identifier: COP.CON.0055

Kind: Concrete

Name:* Establish Project Estima

Description: Establish Project Estimates - Function Points

Entry Criteria: Project scope is defined.

Exit Criteria: Function points estimates are available.

Defined by:* COPPE-II

Responsible:* Project Manager

	Name	Description
☐	Project Plan	Comprises the plans that are going to be monitored throughout the project.
☐	Requirements Specification	Specification of the requirements of the Project.
☐	Project Monitoring Report	Reports project's status based on the actual performance.

Figure 10 Adding a Process Component Using the Support Tool – Basic Data

In order to define each component, process engineers had to define: their indentifier, kind (concrete or abstract), name, description, entry and exit criteria, organization that defined it, process role responsible for its enactment (e.g., project manager), entry and exit parameters (i.e., required and produced work products), other process roles that participate in its enactment, tools that support its enactment, and measures that can be collected during its enactment. Moreover, if a component is a variant of another, the abstract component from which it is a variant has to be identified. The component has also to be mapped to the process features that it complies and also to the ones that conflict with it. Some of these fields do not appear in Figure 10, since the screen shown has to be scrolled down for some of them to appear, and some are included in the other tab panels (process features, variation and measures). Furthermore, their internal architecture can also be defined (its support is similar to the one presented in Figure 11).

The next step focus on structuring and characterizing the SPL. In our example, COPPE/UFRJ structured the SPL as shown in Figure 11 through our supporting tool. We can see that the component "Assess Work Product Quality" was considered optional, while all the others were defined as mandatory elements of the SPL. The last step includes a review of the defined reusable elements. This review is not supported by our tools, and a checklist is provided to guide the review. As a result of this review, the previously defined elements can be changed or removed, and new ones may be necessary. After that, the definition for reuse step is finished.

With this SPL, the SPIO COPPE/UFRJ would be able to derive different process, depending on the selected process features for each particular situation. Moreover, through the use of process broker, the SPIO is able to send new process elements to the organizations, and the organizations is be able to send back data related to the enactment of the components. Using this data, the SPIO could update process components performance data in order to use it during process definition, since components should be chosen based on their past behavior.

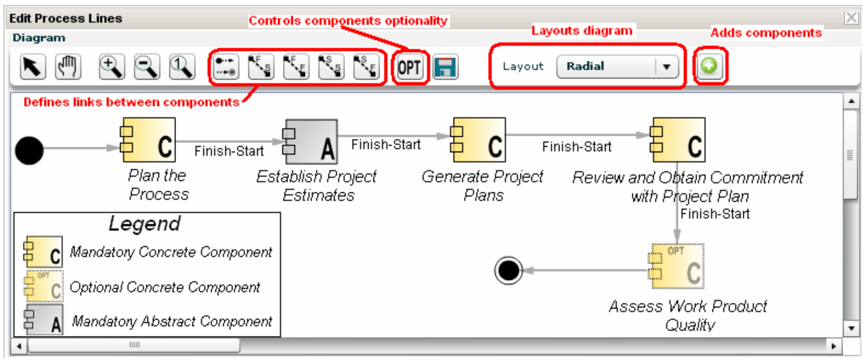

Edit Process Lines
Diagram
Controls components optionality
Layouts diagram
Adds components
Defines links between components
OPT
Layout
Radial
C
Plan the Process
Finish-Start
A
Establish Project Estimates
Finish-Start
C
Generate Project Plans
Finish-Start
C
Review and Obtain Commitment with Project Plan
Finish-Start
OPT C
Assess Work Product Quality
Legend
Mandatory Concrete Component
Optional Concrete Component
Mandatory Abstract Component

Figure 11 Structuring a Process Line Using the Support Tool

Since software process lines (SPLs) are the most promising approach and inadequate support tools is the great concern, considering the survey presented on Section 6, SPLs supporting tools deserved special attention. To address it in our approach, we are providing better supporting tools to SPLs definition and use. There are not many works on the literature focusing on tools to support SPLs. Therefore, we are focusing on providing the functionalities that are needed, but considering also usability issues and knowledge that has to be available. We are using rich internet applications [Preciado et al., 2005] technology to make it easier for the users to interact with software process lines, as shown in Figure 11.

Although there are some other tools that also aim at achieving process reuse [WebApsee, 2010; Rational, 2010; EPF, 2010] they do not use the concept of software process line or process features, which are central aspects of our tool. Therefore, a comparison with these tools would not be adequate.

The reuse environment described in this section already has its first version implemented, although it is still being improved. Process Broker component has also a functional version already. Its current version has a XML Schema defined, import, and export functionalities available. We have already used the environment in real contexts to define the reusable elements using top-down and bottom-up approaches, as described in Section 5. We also are going to use it on every process definition opportunity that we have hereafter. Furthermore, as mentioned earlier, our research group is working on the definition of SPLs for particular situations. Therefore, the environment will also be used to perform these definitions, although these definitions are yet at an early stage. We also are going to use it to define the new versions of the processes of LENS, since the organization is aiming at achieving higher maturity levels and processes need to be improved. In addition, at LENS we have already identified the need for a software process line to consider the differences among software projects requested by the industry and software projects that are developed in the context of academic research.

8 Conclusions

This paper presented a software process definition approach based on reuse techniques. Our main contributions are: (i) the adaptations made to use some reuse concepts in our approach (Section 3); (ii) the considered reuse scenarios, i.e., software process implementing organizations, software process development organizations, and software processes (Section 4); (iii) the steps that should be followed to define the reusable elements and how to perform them, including usage experiences (Section 5); (iv) the results of a survey on the expected benefits and difficulties from the adoption of the reuse approaches being used (Section 6); and (v) an environment to support process reuse, including the overall structure, some of the available tools, and the development status (Section 7).

We believe it is possible to expect similar benefits and difficulties from the adoption of software process reuse techniques, when compared to software product reuse techniques, as shown in Section 6. We also believe that these benefits may be provided by our approach, although some of these difficulties may also need to be addressed. Moreover, we believe that through our approach, organizations could take more advantage from the reuse opportunities that happen on different contexts, as described in Section 4.

As mentioned in the introduction, we believe that if knowledge belonging to experienced process engineers could be made explicit, formalized, and available to other professionals, it would probably be possible to reuse this knowledge in an effective way. Results of the presented survey indicate that experienced process engineers do consider that these goals could be achieved through the use of process components, features, and process lines. Therefore, since our approach and tools provide means to define these reusable elements and (re)use them, we believe our work is able to achieve the mentioned goals.

The main limitations of this work are related to the supporting environment, which is in its first version, and some features are still being developed, such as the software process line derivation. We intend to alleviate this problem in the next few months, since the environment is in constant evolution. Moreover, in the next versions, Process Broker component will manage which elements can be exchanged by which repositories, establishing exchange policies and access rules. Process Broker will also define some plug-ins in order to make it possible the exchange of data from repositories with different structures. We also could not use all the aspects of our approach yet, since some tools are under development.

We intend to run a complementary study of the survey described in Section 6 in order to check if the expected benefits and difficulties are going to be present in our approach specifically. Moreover, we will also perform other kinds of experiments to quantitatively evaluate the results of the use of our approach in regard to aspects such as: time spent, rework, quality of the process, etc, when compared to ad-hoc process reuse or no process reuse approaches. We will start planning these studies in the next few months. We also want to improve our support on process features, defining and being able to handle more complex rules among them to better derive SPLs, and also on configuration management of the reusable elements.

Acknowledgements

We would like to thank the students at COPPE/UFRJ and UFF, the survey participants, and the anonymous reviewers for their valuable contributions to this work. Moreover, we would like to thank CNPq and FAPERJ for their financial support.

References

[Acuña et al., 2000] Acuña, S. T., Antonio, A., Ferré, X., López, M. and Maté, L.: "The Software Process: Modelling, Evaluation and Improvement"; *In:* CHANG, S. K. (ed.) Handbook of Software Engineering and Knowledge Engineering, 1st ed, World Scientific Publishing Company, Singapore (2000).

[Armbrust et al., 2009] Armbrust, O., Katahira, M., Miyamoto, Y., Münch, J., Nakao, H. and Ocampo, A.: "Scoping Software Process Lines"; Software Process: Improvement and Practice, 14, 3 (2009), 181-197.

[Barreto et al., 2010] Barreto, A., Duarte, E., Rocha, A. R. and Murta, L.: "Supporting the Definition of Software Processes at Consulting Organizations via Software Process Lines"; 7th International Conference on the Quality of Information and Communications Technology, Porto, Portugal, (2010), Accepted for publication.

[Barreto et al., 2008] Barreto, A., Murta, L. and Rocha, A. R.: "Software Process Definition: a Reuse-based Approach"; XXXIV Conferencia Latinoamericana de Informática (CLEI'08), Santa Fe, Argentina, (2008), 409-419.

[Bosch, 2000] Bosch, J.: "Design and use of software architectures: adopting and evolving a product-line approach"; 1st, Addison Wesley Professional, New York, United States (2000).

[Chrissis et al., 2006] Chrissis, M. B., Konrad, M. and Shrum, S.: "CMMI: Guidelines for Process Integration and Product Improvement"; 2nd, Addison-Wesley, New York, USA (2006).

[Clements and Northrop, 2001] Clements, P. and Northrop, L.: "Software product lines:practices and patterns"; Addison-Wesley Professional, (2001).

[EPF, 2010] EPF: "Eclipse Process Framework" [Online], Available: http://www.eclipse.org/epf/ [Accessed 12/05/2010].

[Fadila and Mohamed, 2009] Fadila, A. and Mohamed, A. N.: "Reusing heterogeneous software process models"; 14th IEEE ISCC'2009, Sousse, Tunisia, (2009), 291-294.

[Ferreira et al., 2008] Ferreira, A. I. F., Santos, G., Cerqueira, R., Montoni, M., Barreto, A., Rocha, A. R., Barreto, A. S. and Cabral, R.: "ROI of Software Process Improvement at BL Informática – SPI is Really Worth It"; Software Process Improvement and Practice, 13, 04 (2008), 311-318.

[Florac and Carleton, 1999] Florac, W. A. and Carleton, A. D.: "Measuring the Software Process"; 1st, Addison-Wesley, Reading, United States (1999).

[Franch and Ribó, 2002] Franch, X. and Ribó, J.:"Supporting Process Reuse in PROMENADE", Departament de Llenguatges i Sistemes Informàtics, Universitat Politécnica de Catalunya, (2002).

[Fuggetta, 2000] Fuggetta, A.: "Software process: a roadmap"; Conference on The Future of Software Engineering, Limerick, Ireland, (2000), 25-34.

[Fusaro et al., 1998] Fusaro, P., Visaggio, G. and Tortorella, M.: "REP - ChaRacterizing and Exploiting Process Components: Results of Experimentation"; Working Conference on Reverse Engineering, IEEE Computer Society, Honolulu, United States, (1998), 20-29.

[Garg et al., 2003] Garg, A., Critchlow, M., Chen, P., Westhuizen, C. V. d. and Hoek, A. v. d.: "An Environment for Managing Evolving Product Line Architectures"; International Conference on Software Maintenance, Amsterdan, Netherlands, (2003), 358-367.

[Gary and Lindquist, 1999] Gary, K. A. and Lindquist, T. E.: "Cooperating Process Components"; COMPSAC'99, Phoenix, United States, (1999), 218-223.

[Hollenbach and Frakes, 1996] Hollenbach, C. and Frakes, W.: "Software Process Reuse in an Industrial Setting"; 4th ICSR, Orlando, USA, (1996), 22-30.

[ISO/IEC-15504, 2003] ISO/IEC-15504:"Information Technology – Software Process Assessment", Parts 1-9, The International Organization for Standardization and the International Electrotechnical Commission, (2003).

[Jaufman and Münch, 2005] Jaufman, O. and Münch, J.: "Acquisition of a Project-Specific Process"; Product Focused Software Process Improvement, Oulu, Finland, (2005), 328-342.

[Johnson, 1997] Johnson, R. E.: "Frameworks = (components + patterns)"; Communications of the ACM, 40, 10 (1997), 39-42.

[Jørgensen, 2000] Jørgensen, H. D.: "Software Process Model Reuse and Learning"; Process Support for Distributed Team-based Software Development, Orlando, USA, (2000), 726-731.

[Kellner, 1996] Kellner, M. I.: "Connecting Reusable Software Process Elements and Components"; 10th International Software Process Workshop, Dijon, France, (1996), 8-11.

[Lim, 1994] Lim, W.: "Effects of reuse on quality, productivity, and economics"; IEEE Software, 11, 5 (1994), 23-30.

[Lobsitz, 1996] Lobsitz, R. M.: "A Method for Assembling a Project Specific Software Process Definition"; Hawaii International Conference on System Sciences (HICSS), Hawai, United States, (1996), 722-730.

[Madhavji, 1991] Madhavji, N. H.: "The process cycle"; Software Engineering Journal, 6, 5 (1991), 234-242.

[MCIlroy, 1968] MCIlroy, M. D.: "Mass Produced Software Components"; Software Engineering, Report on a conference sponsored by the NATO Science Committee, Garmisch, Germany, (1968), 88-98.

[Mili et al., 1995] Mili, A., Mili, F. and Mili, A.: "Reusing Software: Issues And Research Directions"; IEEE Transactions on Software Engineering, 21, 6 (1995), 528-562.

[Montoni et al., 2008] Montoni, M., Santos, G., Vasconcelos, J., Figueiredo, S., Cabral, R., Cerdeiral, C., Katsurayama, A. E., Lupo, P., Zanetti, D. and Rocha, A. R.: "Application of the SPI-KM Approach to Support the Implementation of the MPS Model in Small and Medium-Sized Enterprises in Brazil"; Software Quality Professional Journal, 11, 1 (2008), 34-45.

[OMG, 2008] OMG:"SPEM - Software Process Engineering Metamodel"; v2.0 ed, Object Management Group, (2008).

[Osterweil, 1987] Osterweil, L.: "Software Processes Are Software Too"; International Conference on Software Engineering, Monterey, USA, (1987), 2-13.

[Perry, 1996] Perry, D. E.: "Practical Issues in Process Reuse"; 10th International Software Process Workshop, Dijon, France, (1996), 12-14.

[Pfleeger, 2001] Pfleeger, S.: "Software Engineering: Theory and Practice"; 2nd, Prentice Hall, (2001).

[Preciado et al., 2005] Preciado, J., Linaje, M., Sanchez, F. and Comai, S.: "Necessity of methodologies to model rich Internet applications"; Seventh IEEE International Symposium on Web Site Evolution (WSE 2005), Budapest, Hungary, (2005), 7-13.

[Prieto-Diaz, 1993] Prieto-Diaz, R.: "Status report: software reusability"; IEEE Software, 10, 03 (1993), 61-66.

[Rational, 2010] Rational: "Rational Method Composer" [Online], Available: http://www-01.ibm.com/software/awdtools/rmc/ [Accessed 12/05/2010].

[Reis et al., 2001] Reis, R. Q., Reis, C. A. L. and Nunes, D. J.: "Automated Support for Software Process Reuse Requirements and Early Experiences with the APSEE model"; International Workshop On Groupware, Darmstadt, Germany, (2001), 50-57.

[Rombach, 2005] Rombach, H. D.: "Integrated Software Process and Product Lines"; International Software Process Workshop, Beijing, China, (2005), 83-90.

[Ru-Zhi et al., 2005] Ru-Zhi, X., Tao, H., Dong-Sheng, C., Yun-Jiao, X. and Le-Qiu, Q.: "Reuse-Oriented Process Component Representation and Retrieval"; International Conference on Computer and Information Technology, Shangai, China, (2005), 911-315.

[Santos et al., 2007] Santos, G., Montoni, M., Vasconcelos, J., Figueiredo, S., Cerdeiral, C., Katsurayama, A. E., Lupo, P., Zanetti, D. and Rocha, A. R.: "Implementing Software Process Improvement Initiatives in Small and Medium-Size Enterprises in Brazil"; 6th International Conference on the Quality of Information and Communications Technology, Lisbon, Portugal, (2007), 187-198.

[Sherif and Vinze, 2003] Sherif, K. and Vinze, A.: "Barriers to adoption of software reuse A qualitative study"; Information & Management, 41, 2 (2003), 159-175.

[Simidchieva et al., 2007] Simidchieva, B. I., Clarke, L. A. and Osterweil, L.: "Representing Process Variation with a Process Family"; International Conference on Software Process, Minneapolis, Estados Unidos, (2007), 109-120.

[SOFTEX, 2009] SOFTEX: "MPS.BR - Brazilian Software Process Improvement, General Guide:2009" [Online]; SOFTEX - Association for Promoting the Brazilian Software Excellence, Available: http://www.softex.br/mpsBr/_guias/default.asp [Accessed 07/02/2009 2009].

[Sutton and Osterweil, 1996] Sutton, S. M. and Osterweil, L.: "Product families and process families"; International Software Process Workshop, Dijon, France, (1996), 109-111.

[Ternité, 2009] Ternité, T.: "Process lines: a product line approach designed for process model development"; 35th Euromicro SEAA, Patras, Greece, (2009), 173-180.

[Washizaki, 2006a] Washizaki, H.: "Building Software Process Line Architectures from Bottom Up"; PROFES'06, Amsterdam, Netherlands, (2006a), 415-421.

[Washizaki, 2006b] Washizaki, H.: "Deriving Project-Specific Processes from Process Line Architecture with Commonality and Variability"; IEEE International Conference on Industrial Informatics, Singapore, (2006b), 1301-1306.

[WebApsee, 2010] WebApsee: "WebAPSEE - Flexible Process Management" [Online], Available: http://sourceforge.net/projects/webapsee/ [Accessed 12/05/2010].