Computer Science Department
University of Valladolid
Valladolid – Spain

Software Process Specification for Product Line Approach
Version 1.0

Bruno González-Baixauli, Miguel A. Laguna

Computer Science Department, University of Valladolid, Spain
{bbaixauli, mlaguna}@infor.uva.es

Abstract. Software reuse is accepted as a source of important benefits, expressed in productivity and quality terms, when an industrial approach is introduced in the software process. However, mainstream software processes, such as Unified Process, do not include reuse techniques among the tools that software engineers must deal with. In this paper we present a proposal to introduce software reuse with minimal disturbance of established disciplines by means of the introduction of a new process for the product line engineering (in the same spirit of UP) and a UP adaptation for the specific products construction. This proposal reduces the money and time costs related to the progressive introduction of software reuse in an organization because the use of a well-known process definition style.

The basis of this work is a coarse-grained reusable component model, which presents a full project scope and supports the product line approach. Also, reusable components can easily be integrated in a repository of reusable elements.

Some tools which provide support and link to the two processes, including a requirement tool with glossary management and a repository of reusable elements, has been developed.

Keywords: software process, software reuse, product line, Unified Process

Technical Report No. DI-2003-1

1 Introduction

The assembly of new products from software pieces has been one of the main goals of the Software Engineering discipline from its beginning, with the aim of obtaining important benefits, expressed in productivity and quality terms, when an industrial reuse approach is introduced in the software process.

The basic reuse unit was initially the module, but the class readily occupied this role due to the object-oriented paradigm popularity. However, these reuse initiatives failed to establish a systematic reuse approach because these efforts only provided reuse at the small-scale level. For this reason the reuse unit has increased its size and complexity towards coarse-grained reusable software artifacts, such as frameworks or components. Nevertheless, even with these coarse-grained constructions, the expected benefits have not appeared because these large elements present a bottom-up reuse approach (i.e. the composition of arbitrary components to construct systems) that has failed in practice [4].

Finally, product lines appear as the more successful approach in the reuse field, as they combine coarse-grained components, i.e. software architectures and software components, with a top-down systematic approach, where the software components are integrated in a high-level structure. The product line concept emerged in the eighties in the business schools, aiming at achieving scope economies through synergetic development of products [15].

However, product lines is a very complex concept that requires a great effort in both technical – architecture definition, development, usage and instantiation [10, 4, 6]– and organizational – business view [2] – dimensions. In addition, the standard proposals of the software development process traditionally ignore the reuse issues, in spite of their recognized advantages [10]. These characteristics move many organizations away from software reuse, because they cannot support the effort or the investment needed to initiate a product line, changing from a standard process to a entirely new one, as proposed by reuse gurus. We aim to introduce a reuse approach based on product lines that requires less investment and presents results earlier than more traditional product line methods. This proposal incorporates the best practices in reuse approaches, mainly of the domain engineering process, into conventional disciplines of the application engineering process.

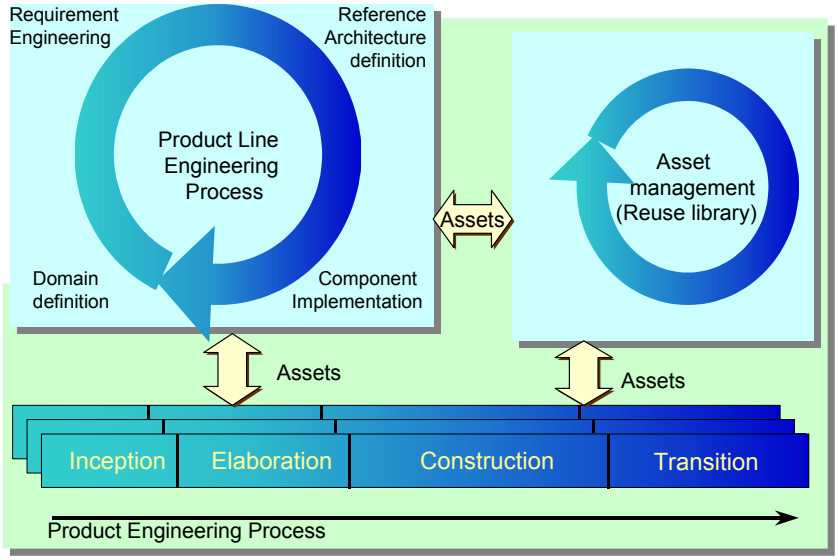

Figure 1. The Domain Engineering and the Product Engineering processes are related by assets interchange, also is needed an Asset management. Note that the Product Line Process and Asset management are continuous process, but the Product Process is iterative.

Traditionally, reuse researchers have been more interested in techniques and processes of domain engineering (*for reuse*), than in product or application process engineering (*with reuse*). We ourselves, in previous work [9], have paid little attention to this second aspect of the problem. Although we recognize the need for a specific process for domain engineering, this aspect only affects a minor part of an organization that seeks to introduce a product line approach: most engineers will go on developing products, and for these engineers a minimal modification of their well-established work disciplines is the most suitable thing. From a practical point of view, only the domain engineering process must be carried out by a specialized team (hired consultants can be responsible for the initiation). This approach allows the rest of the organization to focus on product development as in any other mature engineering. The central idea is that it is not possible to talk seriously of engineering without reuse and it should not be necessary to consider reuse as an independent branch of software engineering. We therefore propose to define two processes separately: a specialized one for domain engineering in the spirit of FORM [13] or Bosch [4] and a process adaptation, based on classical Unified Process (UP) [10] where is introduced the idea of the PL architecture. Both together aims at initiating a product line approach to software development.

The proposal is founded on a coarse-grained reuse model, the Mecano model [8, 9], and a suitable reuse library to manage the reusable elements that offer the

operative support to the reuse process. Figure 1 shows the two processes and the relation with the repository asset management. The figure reflects the difference between the product development process and the other two ones. Product line engineering and asset management are continuous processes without external observable output. The product process is iterative but has a date release as a relevant difference.

The rest of the paper is distributed as follows: the next section explains briefly the Mecano model and its relation to product lines. Section 3 discusses how to introduce reuse in a conventional process. In section 4 the process for reuse we proposed is presented. Section 5 shows how the UP must be altered to take the benefits of the Product Line approach. Section 6 presents a series of tools that support these processes. Section 7 relates our work to other known studies. The section 8 concludes the paper and proposes additional work.

2 Mecano Model and Product Lines

Different practical experiences in the reuse field have emphasized the need of defining a coarse-grained reusable software element, seeking the improvement of the reuse process and its results. The structure of these reusable elements should allow the increase of the reuse process abstraction level, the support of several abstraction levels, the traceability between its components, and the integration of these elements in a reuse process that includes both the domain engineering and the application engineering phases.

Mecano model [8, 9] defines the structural view of a coarse-grained reusable software element (or *mecano*), composed by a set of fine-grained reusable software elements (or assets), each one classified in one of three possible abstraction levels: requirements, design and implementation. Optionally, a *mecano* can include one or many functional descriptors. A functional descriptor is a set of links to the reusable assets that represents functional requirements highlighted by the domain engineer.

The reusable assets that form a *mecano* are interrelated by a set of structural relationships defined in Mecano model. Several categories of structural relationship are defined: intra-level relationships –aggregation, composition, use, extension and association– and inter-level relationship or reification.

The core Mecano model is represented in Figure 2, using UML [19] as modeling language. It represents the main components of this structure: the reusable assets and their relationships. The reusable assets are the reuse-centered components, while the relationships build the layer for automated retrieval processes through an entry point, usually a functional descriptor, and traceability across the reusable asset network. Both the reusable assets and the

relationships must have a type that represents the shared properties of these kinds of reusable assets or relationships. The Mecano model overview presented in this section is centered in the semi-formal view of the model. Also a formal definition of the model exists, based on the tube graph concept and in a context-dependent graph grammar [8].

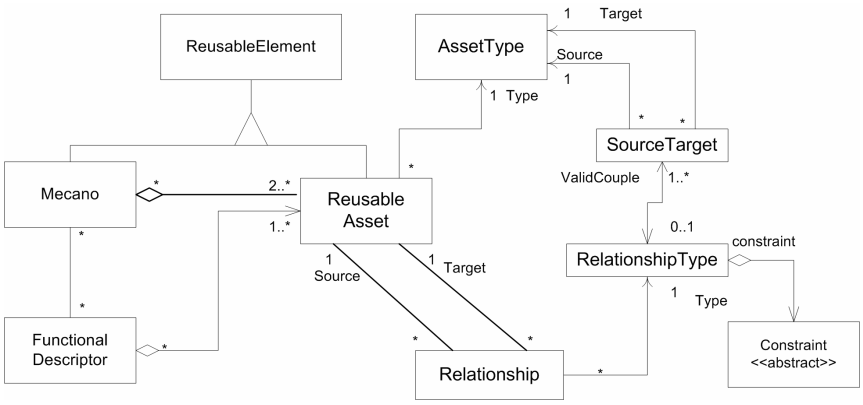

Figure 2. Mecano model defines a structural view of a coarse-grained reusable software element composed by assets, each one classified by its abstraction level.

Mecano support of product lines

As a result of the development of a product line, two main categories of software artifacts are involved: the artifacts shared by the product members in the product line and the product-specific artifacts [4]. This division is shown in Figure 3. From a fine-grained point of view, a product line is a set of interrelated reusable assets, where the three abstraction levels presented in the Mecano model can be clearly identified, i.e. the requirement level – that expresses the product line business model, the requirements of the product line, and the product line variability graph; the design level – that collects the product line architecture; and finally the implementation level – where the generic components, which are compliant with the constraints of the product line architecture, appear.

In this approach every product that belongs to a product line can be seen as a coarse-grained reusable software component modeled by a *mecano*. According to this, the product can be stored in a reuse library that supports the *mecano* management. The initial development consists in the product line definition, i.e.

a basic product line formed by a product line specification and a product line base-architecture.

Taking this basic product line as a first step, the development of new products starts; products that will feed the overall product line (the products of a product line and their components are reusable assets too) introducing these products in the reuse library or reuse repository as *mecanos*. This way the reuse cycle begins, which is formed by both the domain engineering phase and the application engineering phase.

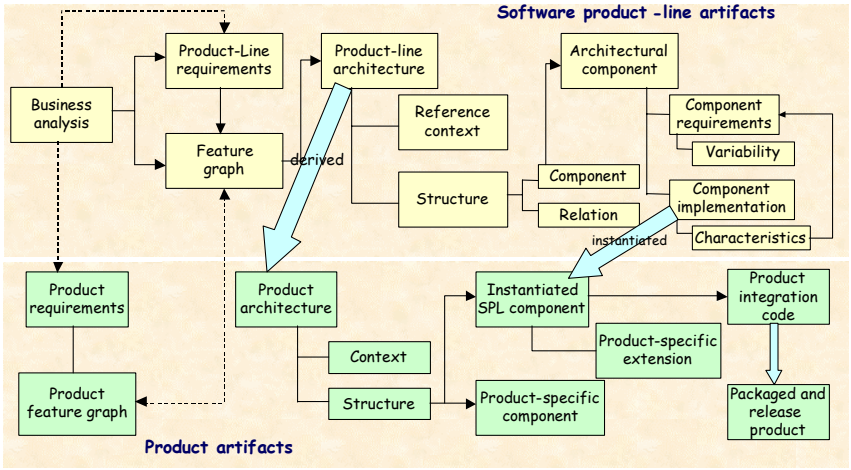

Figure 3. Product-line artifacts [4] divided in two categories: shared by product members and product-specific (instantiated from the first and new ones).

Figure 4 represents in a schematic way the relationships between the product line and *mecano* notions. Every product line is configured by a basic or core product line and a set of compliant products with this core. Each product has a whole software project scope that has been developed from the basic product line. On the other hand, every basic product line has both a specification and an architecture that it is defined from a set of generic software components. The products, due to their definition, are candidates to be represented by coarse-grained reusable elements, *mecanos* in this approach, but also the basic product line, the product line architecture and its components are *mecanos* too, because these elements are configurations of reusable assets that they are classified in different abstraction levels, where the product line specification and the component specifications are functional descriptors.

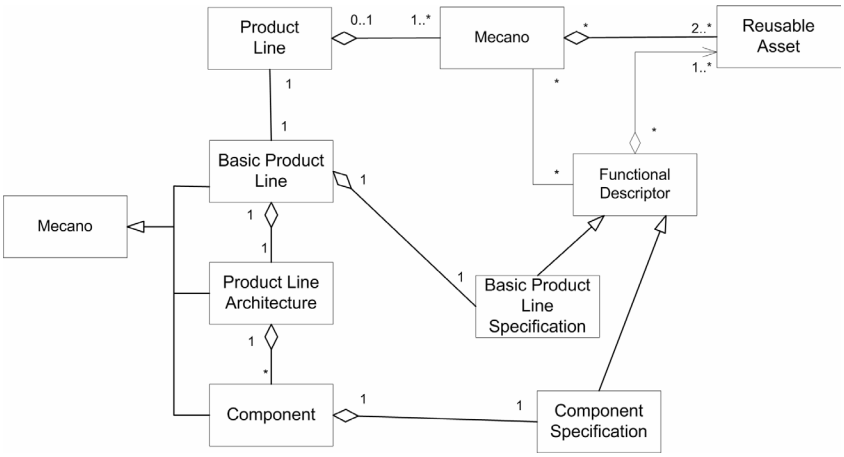

Figure 4. A Product Line is represented by Basic Product Line, which is composed by a specification and an architecture. These elements can be represented as *mecanos* and inserted in a repository.

3 Software Process and Reuse

As we stated above, the initiation of a product line requires an important investment in time, effort and money. For this reason although the product line approach promises many benefits, the organizations, specially the smallest ones, are skeptic to make this kind of inversion.

Based on Mecano model, and taking the advantages of the close connection of *mecanos* to product lines, we aim the introduction of a reuse approach based on product lines that requires less investment and presents results earlier than more traditional product line methods. This proposal incorporates best practices in reuse approaches, mainly of the domain engineering process, in parallel with conventional disciplines of the application engineering process.

Nowadays, two main approaches for software development are in dispute: the lightweight, agile proposals and the heavyweight, highly configurable approaches. Extreme Programming (XP) [3] is the best-known representative of agile processes and the Unified Process (UP) [10] is the best example in the opposite field. The main advantage of UP is that it is a process framework from which particular processes can be configured and then instantiated. UP has to be configured (this is actually a required step defined in UP itself). XP is an *ad hoc* process, difficult to scale or tailor [24]. These characteristics incline us towards UP as the basic process to adapt.

Most processes have some common elements. They require sequences of activities, which are performed by roles (individuals or teams) to produce artifacts. Processes have also a time dimension, with milestones that represent the completion of activities. We must define, therefore, the following dimensions of the process: time aspects, artifacts, activities, and roles.

A discipline (or workflow in old versions) in UP is the collection of activities producing a particular set of artifacts, which represents some important aspect in software development. In the last version, UP's disciplines are Business Modeling, Requirements, Analysis & Design, Implementation, Test, Deployment, Configuration & Change Management, Project Management, and Environment.

A recent initiative, the Software Process Engineering Metamodel (SPEM) raises the level of abstraction of the UP approach and allows the easy upgrade of a process. SPEM is a meta-model for defining processes and their components [20]. The aim of this proposal is to define the minimal set of the process modeling elements necessaries to describe any software development process, without adding specific models or constraints for any specific area or discipline. The central idea in SPEM is that a software development process is collaboration between roles that perform activities on work products (figure 5).

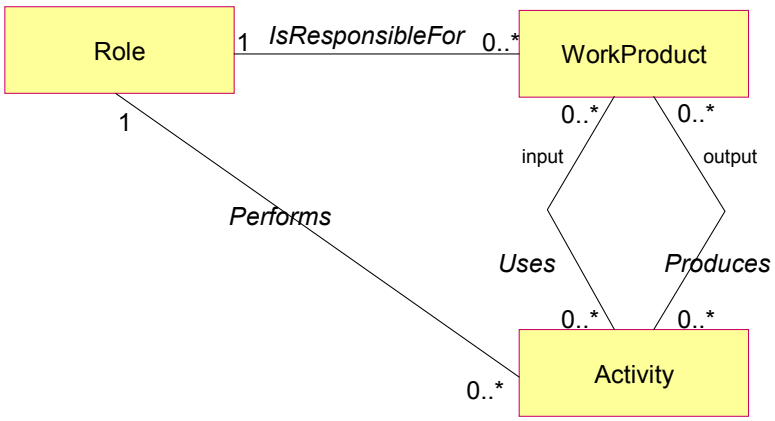

Figure 5. SPEM central idea: a software development process is a collaboration between abstract active entities called *process roles* that perform operations called *activities* on concrete, tangible entities called *work products* [20]

Process Role defines responsibilities over specific Work Products, and defines the roles that perform specific activities. A Work Product or artifact is anything produced, consumed, or modified by a process. It may be a piece of information, a document, a model, source code, and so on. Work Definition

describes the work performed in the process. Activity, Phase, Iteration, and Lifecycle are subclasses of Work Definition. An Activity is the basic Work Definition, only decomposable in atomic elements called steps and with a single Process Role responsible for it. A Phase is a specialization of Work Definition such that its precondition defines the phase entry criteria and its goal defines the phase exit criteria. Finally, a process Lifecycle is defined as a sequence of Phases that achieve a specific goal [20].

Guidance elements may be associated with model Elements to provide detailed information about it. Possible types of Guidance can be Guidelines, Techniques, Metrics, etc. A Technique, for instance, is a detailed algorithm used to create a work product. Another important element of SPEM is Package. A Package is a container that can own and import process definition elements. Process and Discipline are subclasses of Package. A Process is a process component intended to stand alone as a complete process. A Discipline is a particular specialization of Package that partitions the Activities within a process according to a common "theme".

With this support we have proposed the needed modifications to the UP disciplines, to facilitate the smooth introduction of the activities related to development with reuse. Another parallel process, specific for the development for reuse must be defined. In this case, a process different from UP has been elaborated, although with the same iterative and incremental philosophy. Next section shows this process and section 5 examines the adapted UP disciplines for reuse we propose.

4 Product Line Engineering Process

Generally, the idea of establishing a product line in a small organization or development department takes place inside a mature environment of information systems, in which utilities or common components for the products under development may have been identified. This situation suggests that this organization should have a minimum level 2 or 3 in a CMM (Capability Maturity Model) [21] scale, although the organization has had no experience in software reuse. Reifer [22] has proposed a set of additional key areas in the area of reuse to be included in the CMM catalog. In particular, in technical aspects he cites Domain engineering, Architecture engineering or Asset management.

Our process proposal is an iterative process with three main phases – product line inception, elaboration and construction (see Figure 6) - and five unique disciplines: domain definition, product line requirement engineering, product line reference architecture definition, component construction and test. Another disciplines (mainly of management) are shared (in same degree) with the product engineering process as asset management and quality assurance,

product line management and environment. The principal disciplines are shown at Figure 6. The names of the phases refer informally -also intentionally- to the three phases with the denominations used by UP to facilitate the identification of the main goals. This process is being successfully applied in the initiation of a product line in the field of flexible manufacturing work cells in the Computer Science Department of the University of Salamanca [7]. Some experiments have been initiated in other domains, such as software applications for handicapped people. In the next subsections, these phases and disciplines are explained in detail, specially their technical aspects.

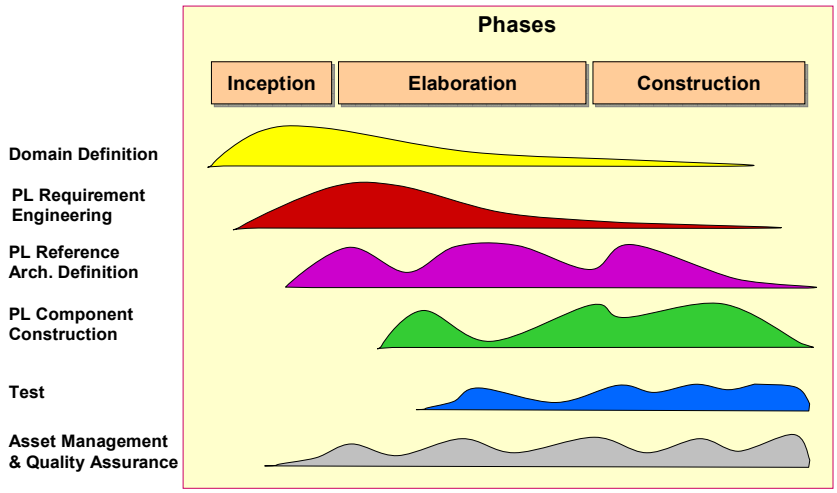

Figure 6. Product line process phases and disciplines. The proposal process is composed by three phases and six main disciplines.

4.1 Phases of Product Line Engineering Process

Product line inception phase

The basic purpose of the product line inception phase is the selection of a concrete application domain, properly focused with a wider strategy according to the global interests of the organization [23]. Therefore, the main discipline is the domain definition. Bosch distinguishes two approaches to initiate a product line inside a domain [4]. First, there is a core for the product line that it is based on a previously developed product family and the explicit experience in the development of these products. Second, the organization initiates a product line from scratch.

In our experience, an organization without previous practice in the product line setup does not embark upon the definition of a product line from scratch. The reason is that initiating a product line in a well-known area for which common elements have been identified is difficult, but starting a new product line in an unknown area is even more difficult and highly improbable.

All the data about the domain must be collected, and what is inside and outside must be decided helped by a market and business analysis; also a first domain analysis and an architecture prototype can be built. In a well-known domain, as usually will be, these first steps should be dynamic and the activities will be done faster.

The milestone of this phase is the fixation of the domain's basic goals, its scope, an initial domain analysis (to guide the initial reference architecture definition) and the initial definition of reference architecture. Finally, it is essential to decide if the product line is worth serious investment.

Product line elaboration phase

The elaboration phase has the same goals of the homonym phase in UP: the analysis of domain requirements and the choice and definition of the common reference architecture. Several iterations are desirable until the final architecture evolves.

The milestone is the definition of the requirements document (with commonalities and variabilities clearly determined) and the creation of the architecture definition of the product line. Also, the architecture for this product line must be validated, so another milestone part is the approbation of an initial architecture (the core architecture implementation); used as a proof of the architecture suitability. An important artifact obtained in this phase is the components build plan with the planning of each component construction. The disciplines involved in this phase are mainly product line requirement engineering and product line reference architecture definition. Also, construction of some component can be afforded for the creation of the proof architecture.

At this moment, the product engineering process can be enabled (at least the first disciplines: business modeling and analysis & design) because we have a complete architecture definition and evidence of the architecture suitability.

Product line construction phase

In the construction phase, the reference architecture is completely designed (the basic interfaces and responsibilities are designed at previous phases, but it is necessary define all the internal issues) and the common and variable

components are designed, implemented and tested. Then, these components are qualified and included in the asset repository as *mecanos*. This must be done for each component or set of components, and once finished inserted in the repository. For this reason, the phase can be represented by several parallel sub-phases, which start and finish independently (figure 7).

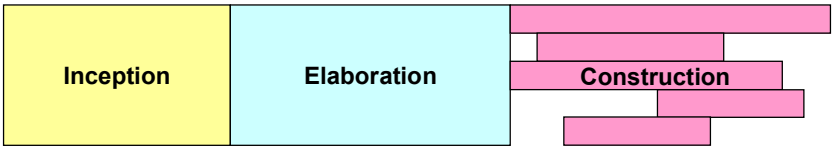

Figure 7. Product line phases. There are three, inception, elaboration and construction. For each component set there is a sub-phase.

After the first iteration of the product line engineering process, the product engineering process is totally enabled. The consequent iterations will originate a configuration management problem focused by the corresponding product line engineering discipline.

4.2 Disciplines of Product Line Engineering Process

Domain definition

The intention of this discipline is the study of the domain's basic goals, its scope and its definition. The first step is to collect all the available information about the possible applications (related to the product line). With this data, the sub-domains involved can be found and described. Also a market and a business analysis (a first domain model with the basic classes) must be performed to decide if a product line approach is profitable. Done at the PL process management discipline.

Next, the information is analyzed to set the domain scope and boundaries and to select the exemplars or specific applications that they are taken as examples of the systems to be made with the product line. Once selected, the exemplars must be described to obtain new vocabulary and analyzed at next discipline. This description will be used to find the product requirements and to detect commonality and variability between them. There are two ways to obtain the description: making a simplified business modeling or taking it from a former developed application.

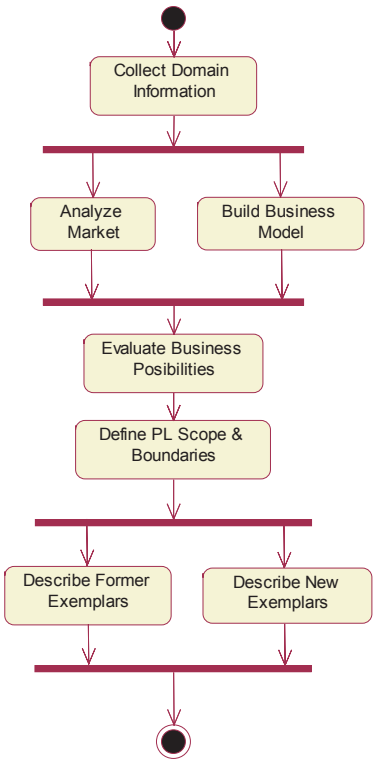

Figure 8: Basic activities of domain definition discipline.

Note that almost all activities update the glossary. It is essential for maintaining the consistency of the requirements work products and for identifying commonalities and variabilities inside the product line. Also it's critical the upgrade of the glossary at every new concept introduction.

The artifacts are a document that collects the domain's basic goals, its scope and its definition, an initial domain model, a list with the exemplars described and a first version of the product line glossary.

Product line requirement engineering

Requirements determination and management in a product line scope are activities that greatly influence the quality of its products. However, the concrete tasks and techniques are not exactly the same practices used in traditional methods for eliciting and analyzing software requirements in an independent product. The conclusion is that current practices in requirement

engineering do not support product line requirement capturing, structuring, analysis and documentation.

In the scope of a product line, the requirements of every product should be determined, even the requirements of the products that still have not been developed, but are inside the product line scope (further products).

In addition to the information that expresses the requirements themselves, it is important to know the variability of the requirements, and dependencies between them (as mandatory or exclusion). To represent this kind of information, the requirements are usually structured in definition hierarchies [17].

In our proposal this discipline is based on FORM (Feature-Oriented Reuse Method) [13]. Thus, each user requirement is an identifiable functional abstraction, or feature.

The purpose of feature modeling is to analyze commonalities and differences among a family of products in terms of application features, and then to organize the analysis results into a feature model, which is used to develop domain components. The features are classified according to the types of information they represent, which fall largely into four categories - application capabilities, operating environments, domain technologies, and implementation techniques [18]. Likewise, in each category the features are organized by a graphical AND/OR hierarchy diagram (see figure 9), i.e. the feature graph or feature diagram, which captures the logical structural relationships between requirements. Also, a feature can be mandatory, optional or alternative according to its existence among applications in a domain. Mandatory features are ones that must exist among applications in a domain, while optional features may not be necessary in some applications of a given domain. Alternative features indicate that no more than one feature can be selected for an application. This classification is also applicable to sub-features, so the restriction is only related to applications with the upper feature.

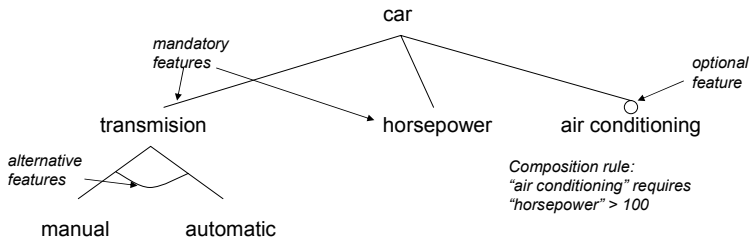

Figure 9. Example showing features of a car [12] with some possible features. Also is represented a composition rule that constraints one feature to other feature value.

Various relationships exist among these features; the relationship types are composed-of, generalization/specialization, and implemented-by. The Mecano model supports all these relationship types.

The Mecano model represents the feature graph concept by a functional descriptor [8]. Besides, this coarse-grained reusable element model completes the representation of the product line requirement specification with composition rules and a set of decision related to the product line characteristics. These rules and decision are represented in the Mecano model through association intra-level relationships representing mutual dependency and mutual exclusion relationship types.

In relation with the discipline, the requirement elicitation is based on a feature analysis, but also a use-case or scenarios analysis can help to obtain the features (these methods are usually more familiar to software developers) and to relate the features to the stakeholders. The question of which analysis must guide the other depends on the product line requirements analyst and his knowledge of the domain or the domain experts' availability.

If the analyst has experience and domain experts are available, the best strategy is a feature-driven one; otherwise the best is a use-case-driven strategy (see figure 10) [5]. This analysis must be done starting from the domain definition (scope and boundaries can help) that will provide a first idea and for each exemplar, to obtain the commonality and variability of the product line. An important issue is the integration of an exemplar with the rest of them: we must be sure there are no conflicts or repeated functionality.

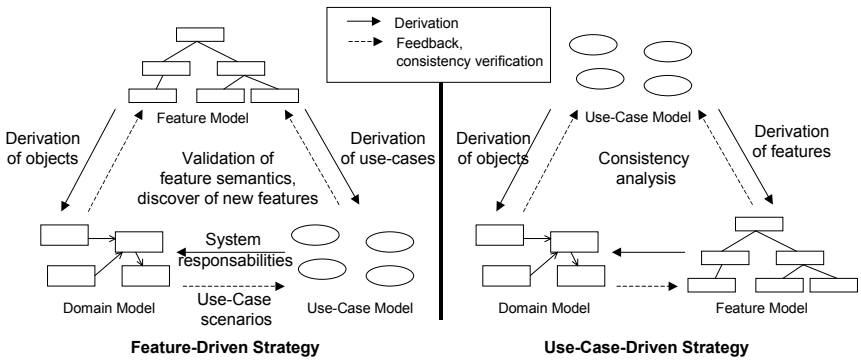

Figure 10: Feature elicitation strategies: if domain analyst has experience about the domain or domain experts are available is better Feature-driven strategy, otherwise Use-Case-driven is better. Adapted from [5]

The artifact of this discipline is the set of reusable assets representing the functional descriptors of the *mecano* that models the basic product line. These

are a set of models with the product line features (features model) and the relationship with the stakeholders (use-case model). Also the product line glossary is updated.

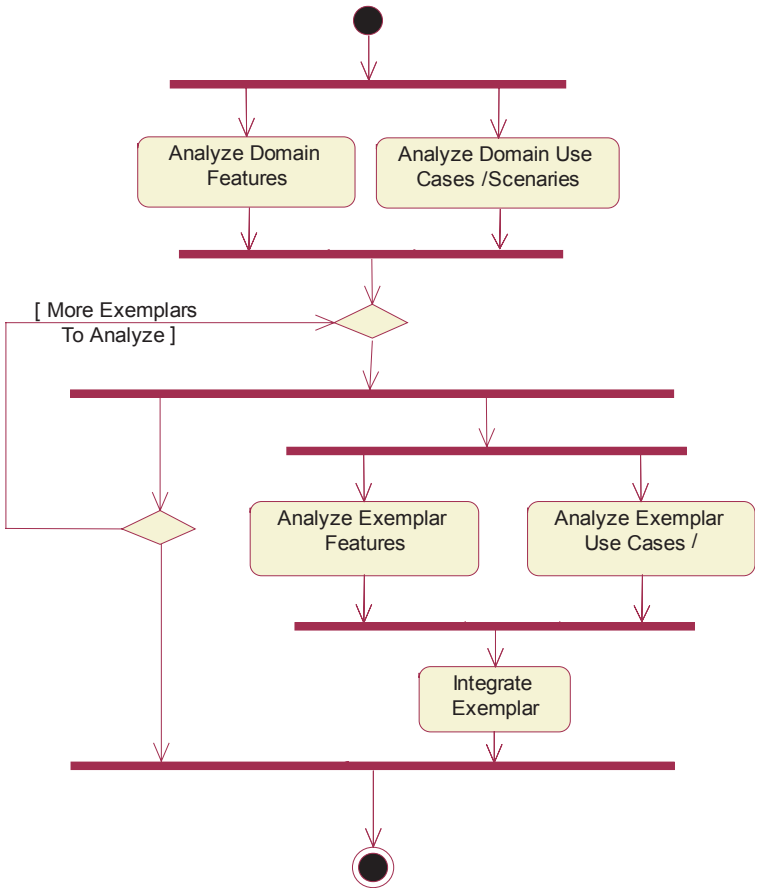

Figure 11: Basic activities of PL requirement engineering discipline.

Product Line Reference Architecture Definition

Once the basic product line requirements are determined, this is the most critical activity in the initiation of a product line from a reuse perspective: This is because this reference architecture will be reused in every product that feeds the product line in the application engineering process. In addition it must comply with the different product line applications (actual and furthers) requirements and be enough flexible to include product specific components.

The reference architecture of a product line is a coarse-grained reusable asset that can be modeled as a *mecano* (see Figure 4). Nevertheless, in our opinion and experience, the design of the reference architecture is probably the most creative aspect of the overall process, and accordingly, the more difficult to standardize.

The experience of the software architect and the kind of products determine the definition of the product line reference architecture. In the case of well-known domains, the use of classic architectures, such as client-server architecture, will be enough, but in other more complex or undefined situations, the entire architect inventive will be required.

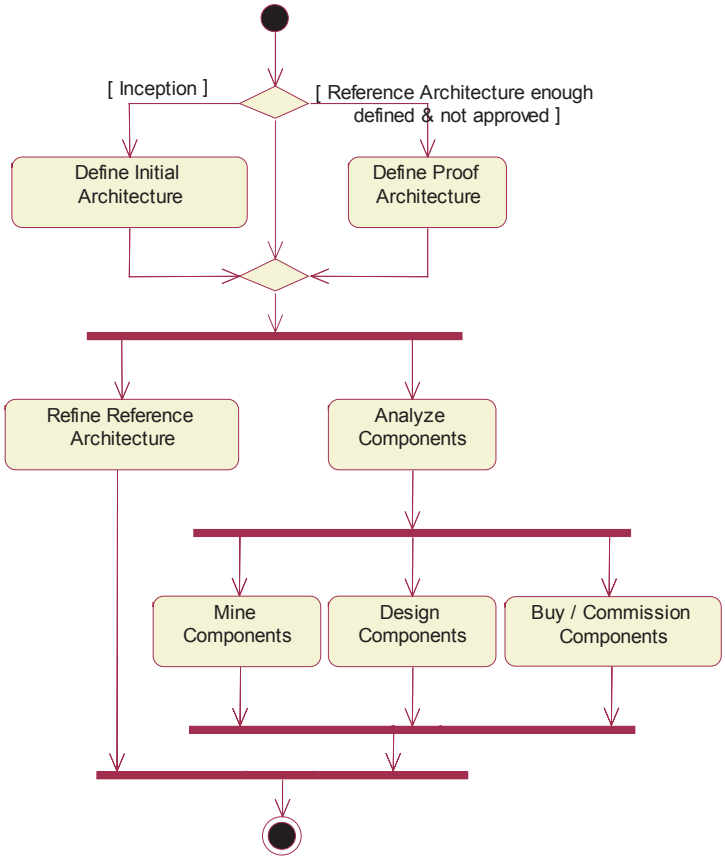

Figure 12: Basic activities of PL reference architecture definition discipline.

As a guide, the activities that we are detected for this discipline are: a first analysis from the early domain description with some ideas with the best architecture patterns for this domain. With this and the more detailed

requirements the architecture could be refined. From this refinement appear components, from this moment it's possible analyze them and decide if to mine, construct or buy/commission the component. All this is done iteratively firstly focused on the architecture refinement and then on the component analysis. When the architecture is enough defined a proof of concept to validate its suitability for the product line requirements could be done.

The artifacts of this discipline are the architecture structure and the different components analysis modeled each one as an independent *mecano* included the decision about their construction (mine, new or external). Also it is important to register the traceability of every component with its requisites and mainly with the implemented features.

Component Implementation

The design and implementation of a product line continues with the implementation and integration of the sets of components designed at previous discipline.

This discipline is essentially equivalent to the implementation discipline of UP, but with the inclusion of a new activity of integration of non-implemented components. This is useful for bought or commissioned components that also must been integrated. Related with mine components, usually is needed wrappers for existing software artifacts, that they are designed at mining activity of the former discipline.

The artifacts of this discipline are the product line components, modeled each one as an independent *mecano* related to the *mecano* that represents the product line reference architecture.

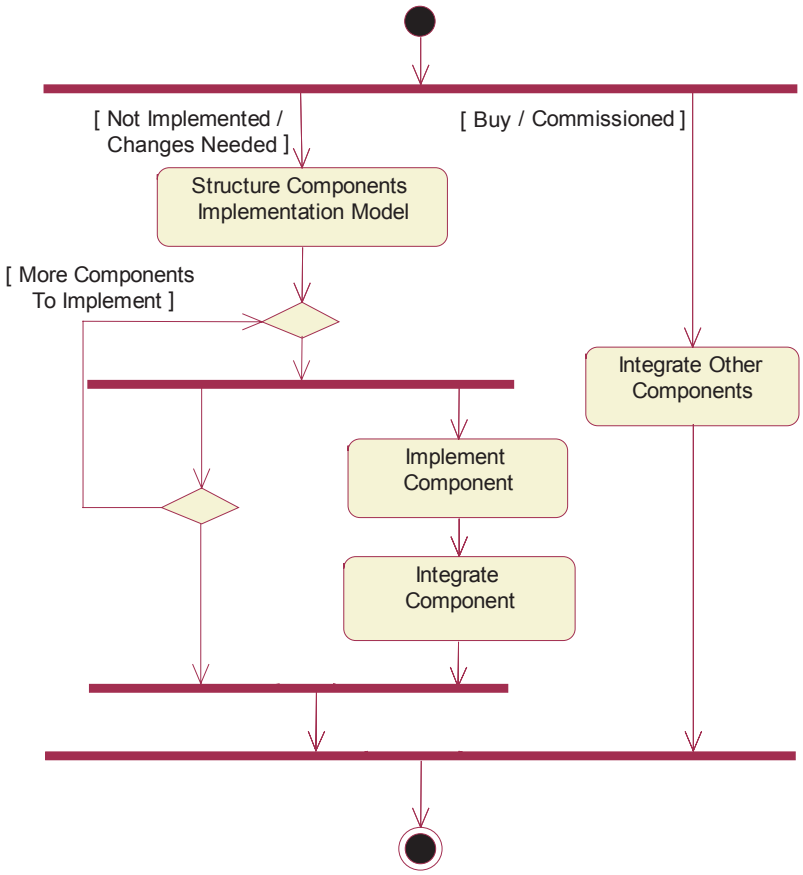

Figure 13: Basic activities of component implementation definition discipline.

Test

The contents of this discipline are equivalents to standard UP. Mainly is used at demonstrate the architecture suitability and at the components integration. Note that at a product line approach the existence of regression test is really useful and the components integration test can be reused at every product specific construction.

The artifacts of this discipline are the test plan, test procedure and test evaluation documents.

Asset management and quality assurance

At this discipline the components are qualified and inserted in the repository. In a product line approach, it is very important to identify a set of quality characteristics of every component since a specific product can require a quality minimum and this information must be available.

The reuse library or repository offers the operative support for the storage and management of the product line artifacts, which are represented by *mecanos* in our proposal.

The repository plays the connection role between the domain engineering and the application engineering processes, allowing the cycle to close [14]. In our proposal, it would be desirable that the organization had a repository that allows the management of assets (see section 6 for details).

The artifact of this discipline is a qualification report obtained as the product line is introduced in the repository.

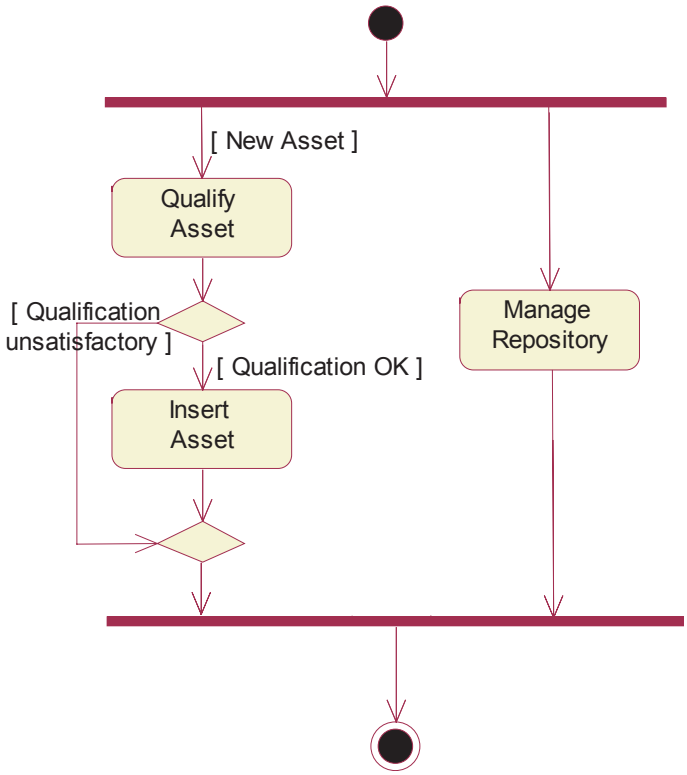

Figure 14: Basic activities of asset management & quality assurance discipline.

Configuration & Change Management

This discipline must control the changes to the shared components to maintain the integrity of the components and guarantee that all the specific products can use the changed components. The activities are similar to the same name UP discipline, but must be taken into account all the specific products before to allow any change. A change can be started by a product specific process (by means of a *product line change request*) or by a product line requirements change.

The artifact of this discipline is a review record with the decision about the change request and the reasoning of allow / deny it.

Environment

At this discipline, the main guidelines are developed and the tools used at product line are selected and initial product developments for each specific product are settled. These are business decisions and must be done for all products; therefore there is only a discipline, not one for each specific product. The activities are similar to the UP ones.

The artifacts of this discipline are the guidelines about design, test, use-cases, … and the tools to be used at the product line engineering and process specific processes. Also is developed the first development case for each specific product.

Product Line Management

This discipline must control the product line development. Also, the activities are similar to the same name UP discipline, because both are iterative and incremental processes. The first step is to decide about the product line profitability (with the data obtained at Domain Definition). Then, if it's decided to develop it, planning, managing and evaluating the different iterations (create the software development plan). The main difference is that also must plan, manage and evaluate the specific products, deciding if the projects suit to the product line and when to start and finish it. This must be done only at a determinated moment, in our approach only is possible at the iteration end, after evaluate the product line state.

Another difference is that at product lines, never there is a product line end because there isn't a final product. Therefore, the product line only can be canceled.

Also, controls the staffing for the product line and each product specific, first assigning almost all staff to the product line activities, and later to the specific products that will be started.

More specifically, here the component sets partition and when the architecture is proved are decided (at *Plan for Next Iteration* activity on first construction phase and on last iteration of Elaboration phase respectively).

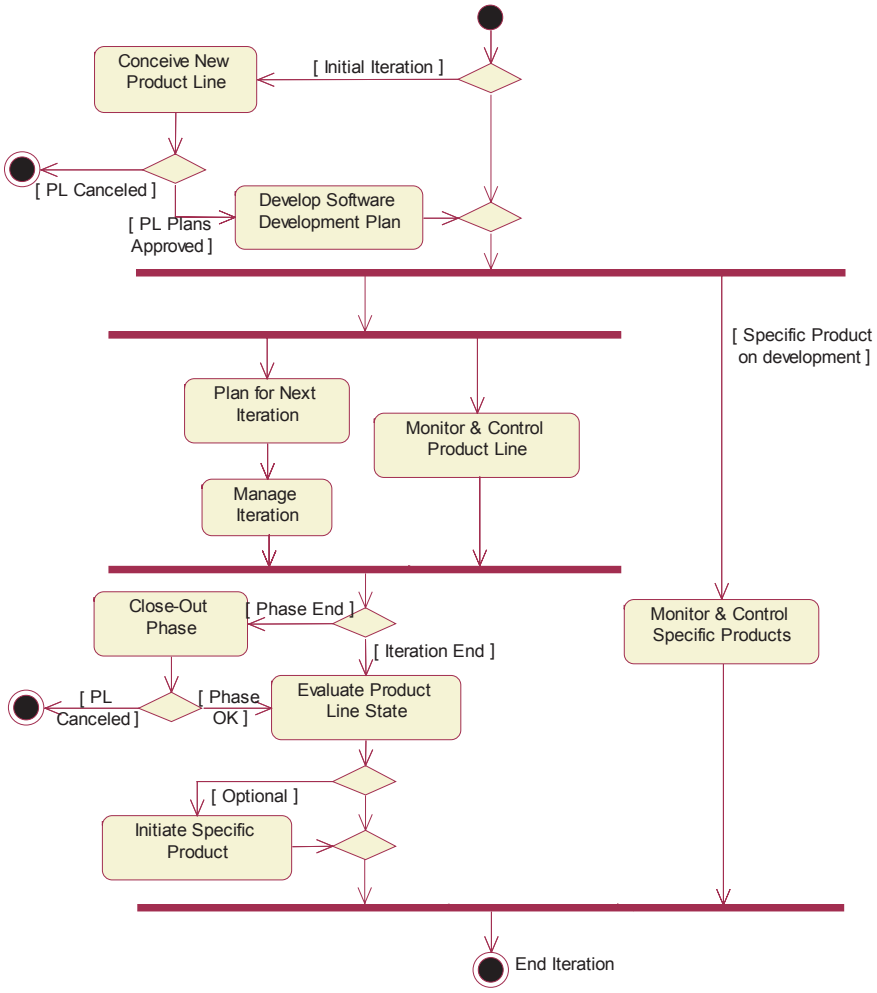

Figure 15: Basic activities of product line management discipline.

The artifacts of this discipline are the product line development plan, phase and iteration plans, product line state evaluations, the P.L. components

implementation plan and for each specific product an initial software development plan.

Discipline	Artifacts
Domain definition	Document expressing the goals, scope and the definition of the domain. List of exemplars. Product Line vocabulary (initial version).
Product line requirement engineering	Set of reusable assets representing the functional descriptors of the *mecano* that models the basic product line.
Product Line Reference Architecture Definition	A *mecano* that represents the product line reference architecture. Components design models and traceability to requirements.
Component Implementation	Product line components, modeled each one as an independent *mecano* related to the *mecano* that represents the product line reference architecture.
Test	Test plan, test procedure and test evaluation documents.
Asset management and Quality Assurance	A qualification report obtained when the product line is introduced in the repository
Configuration and Change Management	A review record with the decision about the change request and the reasoning of allow / deny it.
Environment	Guidelines about design, test, use-cases, … and the tools to be used at the product line engineering and process specific processes. Initial development case for each specific product.
Product Line Management	Product line development plan. Phase and iteration plans. Product line state evaluations. P.L. components implementation plan. Initial software development plan for each specific product

Table 1. Disciplines and obtained artifacts

5 Product Engineering Process

To introduce a product line approach in an organization, some changes to the product development process are desirables. These are mainly needed to manage three issues: the instantiation of the product architecture from the features subset for the application, the previous existence of a basic architecture and the presence of a repository where the new reusable components must be inserted (and the old ones reside).

Taking UP as the starting point, the main changes must be made to the requirements discipline where the application features must be found and the application feature model completed. This feature model is used to derive the architecture on the analysis & design discipline, where the architecture is instantiated (from the product line architecture). Also the new features and the changes in the old features are determined. The new features implementation can be done with a reusable approach if it could be useful to other product line products (or conventionally if not so).

Also it must be considered that, usually, there are former projects with similar requirements (we suppose that the product line is not created from scratch, but from a previous development work). This implies that there is already a previous work on requirements elicitation, analysis & design, implementation, etc, that greatly simplifies the work.

Now all disciplines are more deeply analyzed:

Business Modeling

This discipline is simplified in our approach, because its purpose, mainly to understand the organization structure and problems; it's done in the domain engineering process for all product line. In this discipline, it is only necessary to select a subset of the product line and complete them with the most specific problems.

Requirements

In this discipline, more changes are necessary as use-case modeling is not enough to obtain the requirement vision needed by the product line approach. In a product line, a feature model is required to guide the instantiation of the product architecture from the product line reference architecture. This feature model is constructed from the use-case model, a more known technique; each use-case is analyzed and the features obtained. Here it is essential to compare

(and relate) each feature with the product line ones and create only new features if necessary (there is no feature on the product line with this functionality and it is not possible to modify another one to comply with the functionality).

Finally, it is necessary to contrast the feature model with the product line feature model and to insert all required features and sub-features. From this checking, two lists are created: the new features and the features that need changes to be constructed in the next disciplines.

New artifacts are the product feature model, the list of new features and the list of conflicting features.

Analysis & Design

The main change here is the product line instantiation, once the feature model is finished. This guides all the architecture construction. From this core architecture, only it is necessary to create specific components not implemented at product line or modified them. This is done in the typical UP activities, but the first architecture (corresponding to the Candidate Architecture at UP) is created as a derivation of the product line. Then, following UP activities, the architecture will be refined, and the behavior of the components to implement analyzed and designed.

Here it is also important to decide if the new features / components must be implemented for reuse (if has a functionality useful to other product line application and the cost increment is affordable) or not. If such a decision is taken, the construction should be checked by the product line engineering process engineers or even done by them.

There are no new specific artifacts, but new artifacts are created at the architecture instantiation (the candidate architecture) or when new features are found while the architecture is refined.

Implementation

This discipline doesn't suffer changes of significance because the start point is the design from the analysis & design discipline where the components and how implement them are established, but the activities don't change. Only the purpose varies lightly; now only the new components and adaptations are implemented, not all the system components. Another difference is that from the beginning there are a lot of components implemented, so the final effort would be less.

Test

In this discipline, the only change is that there is an important part no implemented by the application team, this could increase the difficulty but it is supposed that the product line components are well tested, so the final effort would be less. Another source of conflicts are the product line components modification and integration, in this case regression test are useful.

Configuration & Change Management

Any change that only affects to the product specific components does not change the normal process. The problem is when a change on a product line component is needed; in this case, must be another activity that decide if a new component must be implemented (probably modifying the component externally with a wrapper or by inheritance) or if the component itself must be modified and do a change request to the product line engineers.

The new artifact is the product line change request, if necessary.

Deployment

This discipline is important because is the responsible for new components insertion into the general product line repository. So a new activity must be introduced: *Submit Insertion Component Request* that provides a new component to be qualified by the product line engineers, and then inserted into the general repository.

New artifact is the component insertion request.

Project Management

Essentially, the activities at this discipline are unchanged, but the first activities are done at the product line engineering process as it is discussed at product line section. Also is important, at the end of the process to give access to all the data to insert the application inside the product line, paying special attention to the new vocabulary introduced by the application, and to insert it to the repository as a new *mecano*.

Environment

This activity is performed at the product line engineering process. See above for a detailed explanation.

6 Tool support

To be successful, this approach to product line development needs some tools that support the new activities defined. We initially developed an asset repository that implements the *mecano* model [8]. The main interest of the model is the established traceability between requirements, designs and code. The access to the GIRO repository is granted through the GIRO pages (http://giro.infor.uva.es). Other repository engines that manage coarse-grained reusable assets (as Repository in a Box**,** http://www.nhse.org/RIB) can be adapted to support the model.

Starting from the GIRO repository implementation, the goal is to use it in a transparent way from the point of view of the developers. This is achieved by the design of a series of tools that connect standard CASE tools with the repository. An API for insertion and extraction of asset has been defined and implemented as a complement of the repository. Then, a couple of plug-ins for Together and Rational Rose has been developed and installed in the engineers' workstations. This allows the systematic insertion of product line assets in the repository, using an XML standard definition of UML artifacts. A module for searching the product applicable features and obtaining the assets related to them is currently being developed. This module will show the feature model and the feature description (see next paragraph) and will allow the desired ones to be selected, obtaining a partial architecture instantiation from the repository.

A second tool, which is specialized in requirements (Requirement Reuse or R2), helps to find the features, create the feature model, register the goals, capture the functional requirements (as scenarios, workflows or use cases) and trace relations between them. Additional modules provide the quality control of the requirement (by Petri Net simulation) and the PL glossary. This last module must check similarities between concepts to assure the correct understanding of them by the different stakeholders, thus removing the overlapping features.

Additionally, a "light version" of the R2 tool (based on a personal database instead of the complete ORACLE based tool) is available from the GIRO site.

Finally, an adaptation of a process tool is required. Currently, we are working with an adaptation of Rational RUP. RUP is an html-based tool (in a web style), allowing a certain degree of customization. Really, we need two versions of UP: the "product line UP" and the standard UP. The last is a

modification of the RUP tool to indicate all the UP changes described. The former is an implementation of the domain engineering process, defined in a similar way. The complete definition of both processes in SPEM format is available from the GIRO pages.

7 Related Work

In this section, we will briefly introduce related work on product line processes. FORM [16, 18] centers its process at domain analysis, introducing the concept of features, but without to analyze in depth the specific product construction, or management issues at the product line process. It defines three phases: *Context Analysis*, where the scope of the domain is defined; *Domain Modeling* that provides a description of the problem space in three main activities (*Information Modeling*, *Features Analysis* and *Functional Analysis*). At this phase also is obtained a Data Glossary with the definition of the involved concepts; The last phase is *Architectural Modeling* that provides the software solution for the application in the domain with the implementation of a Reference Architecture by means refinement from a Subsystem Model (abstract) through the Process Model until Module Model.

The Software Engineering Institute (SEI) has been working at product lines for several years. From this work they are found three *essential activities*: *Core Asset Development*, *Product Development* and *Management* similar to our processes (product line engineering, product engineering and management disciplines from both processes). Also is created of a framework for product line [6]. This framework describes different best-practices for product line construction divided in three areas (software engineering, technical management and organizational management), but doesn't define a process, only there is a practice with advices to do it. Some of these practices have been introduced at our process in form of process activities.

The SEI also is working at domain analysis refining the FORM technique with the Product Line Analysis (PLA) that combines FODA with use-cases to elicit, analyze, specify and verify the requirements of a product line. The use-case models are useful to find new features and to relate features with stakeholders. Another important issue as is the reference architecture definition activity is also described with the Attribute Driven Design (ADD) method (formerly the Architecture Based Design method) [1]. This method provides a series of steps for designing the conceptual software architecture from feature and use-case models until classes.

Finally, Bosch [4] defines a process centered on the architecture and focused on the quality attributes. He defines three main phases: *development of the architecture*, *deployment through product development* and *evolution of the*

assets, similar to our processes, but with an idea more sequential. The evolution in our proposal is implicitly on the iterative and incremental process. About the components construction he identifies two ways: traditional (with components) or with object-oriented frameworks that embodies an abstract design for solutions to a family of related problems. Also he gives the steps to follow at product line derivation and the possible problems about this.

8 Conclusions

In this paper, we have introduced a product line process that does not require great effort, time or money investment. This approach smoothes the organizational issues, taking as base the widely known UP, introducing some changes to allow a product line approach and supporting the new activities with a set of tools.

The *mecano* model is the base to support the product line concept in our approach. The product line artifacts are stored in a reuse library to permit the reuse life cycle articulated in product line engineering and product engineering disciplines.

In our approach, the conventional software process is gently adapted to include the peculiarities of a development based on a product line philosophy with minimal changes and with tool support. In addition, a new process is introduced, where the product line is defined in a systematic way, similar to UP, to decrease process learning.

We think that the characteristics of the presented process structure are an attractive proposal for organizations with limited resources. Thus, this kind of organizations can join the reuse field through a product line approach that allows their maturity level in software construction to be improved.

The associated tools we are developed are a firm support of the product line process. The experiences carried out on academic developments are rewarding. Our future work includes the introduction of this process in software houses, as an essential step to validate the approach and measure the perceptible advantages objectively.

Acknowledgements

This work has been supported by Spanish CICYT (Dolmen Project – TIC2000-1673-C06-05).

References

1. Bachmann, Felix; Bass, Len; Chastek, Gary; Donohoe, Patrick & Peruzzi, Fabio. *"The Architecture Based Design Method"* Technical Report, CMU/SEI-2000-TR-001. Software Engineering Institute (Carnegie Mellon) Pittsburgh, PA 15213.
2. Bass, L., Clements, P., Donohoe, P., McGregor, J. and Northrop, L. "*Fourth Product Line Practice Workshop Report*". Technical Report CMU/SEI-2000-TR-002 (ESC-TR-2000-002), Software Engineering Institute. Carnegie Mellon University, Pittsburgh, Pennsylvania 15213 (USA). 2000.
3. Beck, Kent Extreme Programming Explained, Addison-Wesley, 2000.
4. Bosch, J. "*Design & Use of Software Architectures. Adopting and Evolving a Product-Line Approach*". Addison-Wesley. 2000.
5. Chastek, G., Donohoe, P., Kang, K. C., Thiel, S. *"Product Line Analysis: A Practical Introduction"*. Technical Report CMU/SEI-2001-TR-001 ESC-TR-2001-001, Software Engineering Institute (Carnegie Mellon), Pittsburgh, PA 15213
6. Clements, Paul C. and Northrop, Linda. *"Software Product Lines: Practices and Patterns"*. SEI Series in Software Engineering, Addison-Wesley. 2001.
7. Curto, B., García, F. J., Moreno, V., González, J. and Moreno, Á. Mª. "*An Experience of a CORBA Based Architecture for Computer Integrated Manufacturing*". In proceedings of 8th IEEE International Conference on Emerging Technologies and Factory Automation – ETFA 2001. (Antibes – Juan les Pins, France, October 15-18, 2001). Pages 765-769. IEEE Press. 2001.
8. García Peñalvo, F. J. "*Modelo de Reutilización Soportado por Estructuras Complejas de Reutilización Denominadas Mecanos*". Ph. D. Dissertation, in Spanish. Universidad de Salamanca. Enero, 2000.
9. García, F. J., Barras, J. A., Laguna, M.A., and Marqués, J. M. "*Product Line Variability Support by FORM and Mecano Model Integration*". In ACM Software Engineering Notes. 27(1);35-38. January 2002.
10. Jacobson, I., Booch, G., Rumbaugh, J. "*The Unified Software Development Process*". Object Technology Series. Addison-Wesley, 1999.
11. Jacobson I., Griss M. and Jonsson P. "*Software Reuse. Architecture, Process and Organization for Business Success*". ACM Press. Addison Wesley Longman. 1997.
12. Kang, K. C., Cohen, S., Hess, J., Nowak, W. and Peterson, S. *"Feature-Oriented Domain Analysis (FODA) Feasibility Study"*. Technical Report, CMU/SEI-90-TR-21, Software Engineering Institute (Carnegie Mellon), Pittsburgh, PA 15213
13. Kang, K. C., Kim, S., Lee, J. y Kim, K. "*FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures*". Annals of Software Engineering, 5:143-168. 1998.
14. Karlsson, E.-A. (Editor). "*Software Reuse. A Holistic Approach*". Wiley Series in Software Based Systems. John Wiley and Sons Ltd. 1995.
15. Knauber, P. and Succi, G. "*Perspectives on Software Product Lines*". ACM Software Engineering Notes, 26(2):29-33. March 2001.
16. Krut, R. *"Integrating 001 Tool Support into the Feature-Oriented Domain Analysis Methodology"*. Technical Report, CMU/SEI-93-TR-011, Software Engineering Institute (Carnegie Mellon), Pittsburgh, PA 15213
17. Kuusela, J. and Savolainen, J. "*Requirements Engineering for Product Families*". In Proceedings of 22nd International Conference on Software Engineering – ICSE 2000. Pages 60-68. ACM Press. 2000.
18. Lee, K., Kang, K. C., Chae, W. y Choi, B. W. "*Feature-Based Approach to Object-Oriented Engineering of Applications for Reuse*". Software: Practice and Experience, 30(9):1025-1046. 2000.
19. Object Management Group. "*OMG Unified Modeling Language Specification. Version 1.4*". Object Management Group Inc. 2001.
20. Object Management Group. *"Software Process Engineering Metamodel Specification*". Version 1.0. Object Management Group Inc. November 2002.

21. Paulk, Mark C., Curtis, Bill, Chrissis, Mary Beth and Weber, Charles V. "*Capability Maturity Model, Version 1.1*". IEEE Software, 10(4):18-27. July, 1993.
22. Reifer, D. J. *"Practical Software Reuse"*. Wiley, 1997.
23. Simos, M., Creps, D., Klingler, C., Levine, L. y Allemang, D. "*Organization Domain Modeling (ODM) Guidebook – Version 2.0*". Technical Report STARS-VC-A025/001/00, Lockheed Martin Tactical Defense Systems, 9255 Wellington Road Manassas, VA 22110-4121. 1996.
24. Smith, J. "*A Comparison of RUP® and XP*". Rational Software White Paper. Technical Paper TP-167. May 2001.

Appendix A: Domain Engineering Process (SPEM style definition)

```
Discipline : Domain Definition

Subactivities
    WorkDefinition : Collect Domain Information
        Activity : Collect Information
        ProcessRole : Domain Engineer
        WorkProduct : Information Sources Document (PL Vision)
        WorkProduct : PL Description (PL Vision)
        WorkProduct : PL Glossary [Outlined]
            Steps
                Step : Determine interesting domains
                Step : Define data sources
                Step : Gather any available data about interesting
                domains
                Step : Gather any available data about current
                organization
                Step : Describe current organization structure
                Step : Identify stakeholders
                Step : Survey stakeholders to obtain more data
                Step : Organize all obtained data
        Activity : Collect Exemplars (Domain Products)
        ProcessRole : Domain Engineer
        WorkProduct : PL Exemplars Document [Outlined]
            Steps
                Step : Look for existing applications related to
                domain
                Step : Look for further applications related to
                domain
                Step : Gather any available data about exemplars
                Step : Organize all obtained data
                Step : Insert applications into the Exemplars
                Document.
        Activity : Understand Relevant Sub-domains
        ProcessRole : Domain Engineer
```

WorkProduct : **PL Description (PL Vision)**
WorkProduct : **PL Glossary [Outlined]**
Steps
Step : **Describe product line sub-domains**
Step : **Identify recurring problems and known solutions within sub-domains**
Step : **Organize all obtained data**
Activity : **Capture a Common Vocabulary**
ProcessRole : **Domain Engineer**
WorkProduct : **PL Glossary**
Steps
Step : **Find common terms**
Step : **Evaluate your results**
WorkDefinition : **Analyze market**
Activity : **Analyze market**
ProcessRole : **Domain Engineer**
WorkProduct : **PL Market Analysis (PL Vision)**
Steps
Step : **Describe market**
Step : **Identify customer segments**
Step : **Map products to segments**
Step : **Examine competitors**
Step : **Evaluate the results**
WorkDefinition : **Build Business Model**
Activity : **Analyze market**
ProcessRole : **Domain Engineer**
WorkProduct : **PL Business Model (PL Vision)**
WorkProduct : **PL Domain Model (outlined)**
Steps
Step : **Describe current business**
Step : **Set business goals**
Step : **Identify business process**
Step : **Identify business workers**
Step : **Evaluate the results**
Activity : **Capture a Common Vocabulary**
ProcessRole : **Domain Engineer**
WorkProduct : **PL Glossary**
Steps
Step : **Find common terms**
Step : **Evaluate the results**
WorkDefinition : **Evaluate Business Possibilities**
Activity : **Analyze Business Possibilities**
ProcessRole : **Domain Engineer**
WorkProduct : **PL Vision**
Steps

Step : **Analyze current organization situation**
Step : **Predict future cost and benefits using a product line approach**
Step : **Draw conclusions**

WorkDefinition : **Define PL Scope & Boundaries**
Activity : **Set and adjust scope / boundaries**
ProcessRole : **Domain Engineer**
WorkProduct : **PL Scope & & Boundaries (PL Vision)**
Steps
Step : **Complete data obtained at *Collect Domain Information***
Step : **Analyze product line data and Exemplars**
Step : **Identify domain characteristics**
Step : **Determine which characteristics should be considered part of the product line**
Step : **Evaluate the results**
Activity : **Find other Domains Relations**
ProcessRole : **Domain Engineer**
WorkProduct : **PL Related Domains (PL Vision)**
Steps
Step : **Analyze Product Line Data and Exemplars**
Step : **Identify Related Domains**
Step : **Describe Relations with the Related Domains**
Step : **Evaluate the Results**
Activity : **Select Exemplar Applications**
ProcessRole : **Domain Engineer**
WorkProduct : **PL Exemplars Document**
Steps
Step : **Check Exemplars Suitability to Product Line Scope and Boundaries**
Step : **Evaluate the Results**
Activity : **Capture a Common Vocabulary**
ProcessRole : **Domain Engineer**
WorkProduct : **PL Glossary [Updated]**
Steps
Step : **Find Common Terms**
Step : **Evaluate the Results**

WorkDefinition : **Describe Former Exemplars**
Activity : **Revise Exemplar Documentation**
ProcessRole : **Domain Engineer**
WorkProduct : **Exemplar Business Vision**
Steps
Step : **Revise application documentation**

Step : **Obtain application boundaries, stakeholders, goals & constraints**
Step : **Assess reengineering possibilities**
Step : **Formulate problem statement**
Step : **Evaluate the results**
Activity : **Capture a Common Vocabulary**
ProcessRole : **Domain Engineer**
WorkProduct : **PL Glossary [Updated]**
Steps
Step : **Find common terms**
Step : **Evaluate your results**
WorkDefinition : **Describe New Exemplars**
Activity : **Set & Adjust Exemplars Goals**
ProcessRole : **Domain Engineer**
WorkProduct : **Exemplar Business Vision**
Steps
Step : **Define Application Boundaries**
Step : **Identify Stakeholders**
Step : **Gain agreement on the goals of the target organization**
Step : **Identify constraints**
Step : **Formulate problem statement**
Step : **Evaluate the results**
Activity : **Capture a Common Vocabulary**
ProcessRole : **Domain Engineer**
WorkProduct : **PL Glossary [Updated]**
Steps
Step : **Find common terms**
Step : **Evaluate your results**

Discipline : **PL Requirement Engineering**

Subactivities
WorkDefinition : **Analyze Domain Features**
Activity : **Find Activities**
ProcessRole : **Domain Requirements Engineer**
WorkProduct : **PL Feature Model**
Steps
Step : **Find Features**
Step : **Describe Features and Exemplars Relations**
Step : **Relate Features with other Features**
Step : **Insert Features into Feature Model**
Step : **Evaluate the results**
Activity : **Capture a Common Vocabulary**
ProcessRole : **Domain Engineer**

WorkProduct : **PL Glossary [Updated]**
Steps
Step : **Find common terms**
Step : **Evaluate your results**
WorkDefinition : **Analyze Domain Use Cases / Scenarios**
Activity : **Find Actors and Use Cases**
ProcessRole : **Domain Requirements Engineer**
WorkProduct : **PL Use-Case Model**
Steps
Step : **Elicit stakeholders requests**
Step : **Find Actors**
Step : **Find Use Cases**
Step : **Describe how Actors and Use Cases interact in scenarios**
Step : **Package Use Cases and Actors**
Step : **Present the Use-Case Model in Use-Case Diagrams**
Step : **Describe how Features and Use Case Model interact**
Step : **Evaluate the results**
Activity : **Capture a Common Vocabulary**
ProcessRole : **Domain Engineer**
WorkProduct : **PL Glossary [Updated]**
Steps
Step : **Find common terms**
Step : **Evaluate your results**
WorkDefinition : **Analyze Exemplar Features**
Activity : **Find Basic Exemplars Features**
ProcessRole : **Domain Requisites Engineer**
WorkProduct : **Exemplar Feature Model**
WorkProduct : **PL Exemplar Document [Updated]**
Steps
Step : **Find Features**
Step : **Describe Features and Exemplars Relations**
Step : **Relate Features with other product line Features**
Step : **Insert Features into Feature Model**
Step : **Evaluate the results**
Activity : **Capture a Common Vocabulary**
ProcessRole : **Domain Engineer**
WorkProduct : **PL Glossary [Updated]**
Steps
Step : **Find common terms**
Step : **Evaluate your results**
WorkDefinition : **Analyze Exemplar Use Cases / Scenarios**

Activity : **Find Exemplar Basics Actors and Use-Cases**
ProcessRole : **Domain Requisites Engineer**
WorkProduct : **Exemplar Architectural Use-Case Model**
WorkProduct : **PL Exemplar Document [Updated]**
Steps
Step : **Elicit stakeholders requests**
Step : **Find Actors**
Step : **Find Use Cases**
Step : **Relate Use Cases and Actors to product line ones**
Step : **Describe how Actors and Use Cases interact on scenarios**
Step : **Package Use Cases and Actors**
Step : **Present the Use-Case Model in Use-Case Diagrams**
Step : **Describe how Features and Use Case Model interact**
Step : **Evaluate the results**
Activity : **Capture a Common Vocabulary**
ProcessRole : **Domain Engineer**
WorkProduct : **PL Glossary [Updated]**
Steps
Step : **Find common terms**
Step : **Evaluate your results**
WorkDefinition : **Integrate Exemplar**
Activity : **PL Object Modeling**
ProcessRole : **Domain Architect**
WorkProduct : **PL Domain Model [Updated]**
WorkProduct : **PL Feature Realization [Outlined]**
WorkProduct : **PL Vision**
Steps
Step : **Identify candidate Objects from the Feature Model**
Step : **Allocate responsibilities to candidate objects**
Step : **Relate objects to older domain objects**
Step : **Define information exchange between Objects**
Step : **Update glossary with object definitions**
Step : **Evaluate the results**
Activity : **Manage Dependences**
ProcessRole : **Domain Requisites Engineer**
WorkProduct : **Exemplar Vision [Outlined]**
WorkProduct : **Exemplar Architectural Use-Case Model [Updated]**

WorkProduct : **Exemplar Feature Model [Updated]**
WorkProduct : **PL Domain Model [Updated]**
Steps
Step : **Assign attributes**
Step : **Establish and verify traceability**
Step : **Manage changing requirements**
Activity : **Integrate Exemplar**
ProcessRole : **Domain Requisites Engineer**
WorkProduct : **PL Exemplar Document [Updated]**
WorkProduct : **PL Use-Case Model**
WorkProduct : **PL Feature Model**
WorkProduct : **PL Domain Model**
Steps
Step : **Integrate exemplar Features and Use-Case Model**
Step : **Evaluate the results**

Discipline : **PL Reference Architecture Definition**

Subactivities
WorkDefinition : **Define Initial Architecture**
Activity : **PL Architectural Analysis**
ProcessRole : **Domain Architect**
WorkProduct : **PL Architecture Document [Outlined]**
WorkProduct : **PL Vision**
Steps
Step : **Develop architecture overview**
Step : **Define the high-level organization of the subsystems**
Step : **Identify key abstraction**
Step : **Create Feature realizations**
Step : **Identify stereotypical interactions**
Step : **Analyze the results**
WorkDefinition : **Refine Reference Architecture**
Activity : **PL Architectural Analysis**
ProcessRole : **Domain Architect**
WorkProduct : **PL Architecture Document [Outlined]**
WorkProduct : **PL Vision**
Steps
Step : **Define the high-level organization of the subsystems**
Step : **Identify key abstraction**
Step : **Create Feature realizations**
Step : **Identify stereotypical interactions**
Step : **Analyze the results**

Activity : **Identify Design Mechanisms**
ProcessRole : **Domain Architect**
WorkProduct : **PL Design Model [Updated]**
WorkProduct : **PL Architecture Document [Updated]**
WorkProduct : **Supplementary PL Specifications [Updated]**
Steps
Step : **Categorize clients of analysis mechanisms**
Step : **Inventory the implementation mechanisms**
Step : **Map design mechanisms to implementation mechanisms**
Step : **Document architectural mechanism**
Activity : **Identify Design Elements**
ProcessRole : **Domain Architect**
WorkProduct : **PL Design Model [Updated]**
WorkProduct : **PL Architecture Document [Updated]**
WorkProduct : **Supplementary PL Specifications [Updated]**
Steps
Step : **Analyze concurrency requirements**
Step : **Identify processes and threads**
Step : **Identify process lifecycles**
Step : **Identify and specify events**
Step : **Identify and specify signals**
Step : **Identify classes, active classes and subsystems**
Step : **Identify subsystem interfaces**
Step : **Identify capsule protocols**
Activity : **Review the Architecture**
ProcessRole : **Domain Architecture Reviewer**
WorkProduct : **PL Architecture Review**
Steps
Step : **Conduct review meetings**
WorkDefinition : **Define the Proof Architecture**
Activity : **Define the Proof Architecture**
ProcessRole : **Domain Architect**
WorkProduct : **PL Proof Architecture Document**
WorkProduct : **PL Architecture Document [Updated]**
Steps
Step : **Analyze features and exemplars**
Step : **Identify architecture core elements**
Step : **Select proof architecture elements**
Step : **Evaluate the results**
WorkDefinition : **Analyze Components**
Activity : **Features and Use Case Analysis**
ProcessRole : **Product Line Analyst**
WorkProduct : **Feature Realizations**

WorkProduct : **Analysis Classes**
Steps
Step : **Find analysis classes from features & use cases**
Step : **Distribute behavior to analysis classes**
Step : **Describe analysis classes**
Step : **Evaluate the results**
Activity : **Analyze Legacy Components**
ProcessRole : **Domain Analyst**
WorkProduct : **Legacy Components Document**
Steps
Step : **Find legacy components that fits analysis classes**
Step : **Understand needed components changes**
Step : **Evaluate the results**
Activity : **Determine if Mine / Construct / Buy -- Commission**
ProcessRole : **Domain Architect Reviewer**
WorkProduct : **Component Review Record**
WorkProduct : **PL Architecture Document [Updated]**
Steps
Step : **Review feature realizations**
Step : **Review legacy components**
Step : **Review organization policy & resources**
Step : **Evaluate the results**

WorkDefinition : **Mine Components**
Activity : **Mine Components**
ProcessRole : **Domain Designer**
WorkProduct : **Feature Realizations**
WorkProduct : **Design Classes**
Steps
Step : **Analyze candidate components**
Step : **Analyze mining options**
Step : **Design component modification classes**
Step : **Evaluate the results**

WorkDefinition : **Design Components**
(like UP Analysis & Design :: Design Components, *but oriented to a reuse approach with frameworks, automatic component assembly, …)*

WorkDefinition : **Buy / Commission Components**
Activity : **Buy / Commission Components**
ProcessRole : **Domain Designer**
WorkProduct : **Feature Realizations**
WorkProduct : **Design Classes**
Steps

Step : **Analyze buy / commission options**
Step : **Analyze contractors options**
Step : **Evaluate the results**

Discipline : **Component Implementation**

Subactivities
WorkDefinition : **Structure Components Implementation Model**
Activity : **Structure the Implementation Model**
ProcessRole : **Domain Architect**
WorkProduct : **PL Architecture Document** (Implem. View)
WorkProduct : **PL Implementation Subsystems**
WorkProduct : **PL Implementation Model**
Steps
Step : **Create the initial implementation model structure**
Step : **Adjust implementation subsystems**
Step : **Define imports for each implementation subsystem**
Step : **Evaluate the implementation model**
WorkDefinition : **Implement Component Set**
[like UP Implementation :: Implementation Components *with a set of components not at Proof Architecture and grouped at* Plan Component implementation*]*
WorkDefinition : **Integrate Component Set**
Activity : **Integrate Components**
ProcessRole : **Domain Integrator**
WorkProduct : **Component Set Build**
WorkProduct : **Product Line Build**
Steps
Step : **Evaluate automatic integration possibilities**
Step : **Integrate component**
Step : **Evaluate the results**
WorkDefinition : **Integrate Other Components**
(Typically bought or commissioned)
Activity : **Integrate Components**
ProcessRole : **Domain Integrator**
WorkProduct : **Architecture Build**
Steps
Step : **Evaluate automatic integration possibilities**
Step : **Integrate component**
Step : **Evaluate the results**

Discipline : **Test**
[like RUP discipline Test]

Discipline : **Asset management and quality assurance**
    WorkDefinition : **Assure Asset Quality**
        Activity : **Qualify Asset**
        ProcessRole : **Repository Manager**
        WorkProduct : **Component Set Build**
        WorkProduct : **Asset Measures Guidelines**
        WorkProduct : **Component Set Requirements**
        WorkProduct : **Asset Repository Insertion Review**
            Steps
                Step : **Check documentation**
                Step : **Check requisites fulfillment**
                Step : **Make asset measures**
                Step : **Evaluate the results**
    WorkDefinition : **Insert Asset into the Repository**
        Activity : **Insert Asset**
        ProcessRole : **Repository Manager**
        WorkProduct : **Component Set Build**
        WorkProduct : **Asset Repository Insertion Review**
            Steps
                Step : **Complete asset data**
                Step : **Insert asset data into repository**

Discipline : **Configuration & Change Management**
[like RUP discipline Configuration & Change Management, but change requests from specific product processes are possible]

Discipline : **Product Line Management**
    WorkDefinition : **Conceive New Product Line**
        Activity : **Identify and Assess Risks**
        ProcessRole : **Product Line Manager**
        WorkProduct : **Product Line Risk List**
        WorkProduct : **Risk Management Plan**
            Steps
                Step : **Identify potential risks**
                Step : **Analyze and prioritize risks**
                Step : **Identify risk avoidance strategies**
                Step : **Identify risk mitigation strategies**
                Step : **Identify risk contingency strategies**
        Activity : **Initiate Product Line**
        ProcessRole : **Product Line Manager**
        WorkProduct : **Product Line Software Development Plan**
        WorkProduct : **Iteration Plan (first iteration)**
            Steps
                Step : **Assign project review authority (PRA)**

Step : **Assign project manager**
Step : **Assign project planning team**
Step : **Approve project acceptance criteria**
Activity : **Product Line Approval Review**
ProcessRole : **Product Line Manager**
WorkProduct : **Product Line Software Development Plan**
WorkProduct : **Product Line Vision PL**
WorkProduct : **Product Line Approval Review Record**
Steps
Step : **Conduct project approval review meeting**
Step : **Record decision**

WorkDefinition : **Develop P. L. Software Development Plan**
[like RUP workproduct Develop Software Development Plan]

WorkDefinition : **Manage and Control Product Line**
[like RUP workproduct Manage & Control Project]

WorkDefinition : **Plan for Next Iteration**
[like RUP workproduct Plan for Next Iteration. Note that here is where is decided if create in this iteration the Proof Architecture at Elaboration Phase]

WorkDefinition : **Manage Iteration**
[like RUP workproduct Manage Iteration]

WorkDefinition : **Close-Out Phase**
[like RUP workproduct Close-Out Phase with an extra activity if Elaboration Phase]
Activity : **Plan Product Line Components Construction**
ProcessRole : **Domain Architect**
WorkProduct : **P. L. Components Implementation Plan**
Steps
Step : **Identify implementation relations**
Step : **Define component sets**
Step : **Evaluate the results**

WorkDefinition : **Evaluate Project Product Line State**
[like RUP workproduct Manage Iteration]

WorkDefinition : **Initiate Specific Product**
Activity : **Assess Product suitability to Product Line** (NEW)
[Not necessary if the product is an exemplar]
ProcessRole : **Product Line Manager**
WorkProduct : **Product Line Feature List**
WorkProduct : **Product Specific Vision [Outlined]**
WorkProduct : **Product Suitability Document**
Steps
Step : **Describe specific product**
Step : **Analyze current organization situation**
Step : **Analyze product characteristics**

Step : **Analyze product line features**
Step : **Draw conclusions**
Activity : **Identify and Assess Risks**
ProcessRole : **Product Line Manager**
WorkProduct : **Product Risk List**
WorkProduct : **Risk Management Plan**
Steps
Step : **Identify potential risks**
Step : **Analyze and prioritize risks**
Step : **Identify risk avoidance strategies**
Step : **Identify risk mitigation strategies**
Step : **Identify risk contingency strategies**
Activity : **Initiate Specific Product**
ProcessRole : **Product Line Manager**
WorkProduct : **Product Software Development Plan**
WorkProduct : **Iteration Plan (first iteration)**
Steps
Step : **Assign project review authority (PRA)**
Step : **Assign project manager**
Step : **Assign project planning team**
Step : **Approve project acceptance criteria**
Activity : **Product Specific Approval Review**
ProcessRole : **Product Line Manager**
WorkProduct : **Software Development Plan**
WorkProduct : **Product Vision**
WorkProduct : **Product Approval Review Record**
Steps
Step : **Conduct project approval review meeting**
Step : **Record decision**

Discipline : **Environment**

WorkDefinition : **Prepare Environment for Product Line**
[like RUP workproduct Prepare Environment for Project, but with entire Product Line]
WorkDefinition : **Prepare Environment for Project**
[like RUP workproduct Prepare Environment for Project, but is executed when a new project is initiated (see Initiate Specific Product)]
WorkDefinition : **Prepare Environment for an Iteration**
[like RUP workproduct Prepare Environment for an Iteration]
WorkDefinition : **Prepare Guidelines for an Iteration**
[like RUP workproduct Prepare Guidelines for an Iteration]
WorkDefinition : **Support Environment During an Iteration**
[like RUP workproduct Support Environment During an Iteration]

Appendix B: Product Engineering Process Additions (SPEM style definition)

```
Discipline : Requirements

Subactivities
    WorkDefinition : Analyze the Problem
        Activity : Find Basic Product Features (NEW)
        ProcessRole : System Analyst
        WorkProduct : Product Feature List [outlined]
            Steps
                Step : Find Features
                Step : Describe Features and Exemplars Relations
                Step : Relate Features with other Features
                Step : Insert Features into Feature Model
                Step : Evaluate the results
    WorkDefinition : Understand Stakeholder Needs
        Activity : Find Basic Product Features (NEW)
        ProcessRole : System Analyst
        WorkProduct : Product Feature List [outlined]
            Steps
                Step : Find Features
                Step : Describe Features and Exemplars Relations
                Step : Classify Features
                Step : Relate Features with other Features
                Step : Insert Features into Feature Model
                Step : Evaluate the results
    WorkDefinition : Define the System
        Activity : Find Product Features (NEW)
        ProcessRole : System Analyst
        WorkProduct : Product Feature List
            Steps
                Step : Find Features
                Step : Describe Features and Exemplars Relations
                Step : Classify Features
                Step : Relate Features with other Features
                Step : Insert Features into Feature Model
                Step : Evaluate the results
    WorkDefinition : Integration into Product Line (NEW)
        Activity : Validate Product Features
        ProcessRole : System Analyst
```

WorkProduct : **Product Feature List [Validated]**
Steps
Step : **Check Feature names conflicts**
Step : **Check Features existence at Product Line Feature Model**
Step : **Check Feature functionality**
Step : **Change Features to complains to Product Line Feature Model**
Activity : **Create Product Feature Model**
ProcessRole : **System Analyst**
WorkProduct : **Product Feature Model**
WorkProduct : **Product New Features List**
Steps
Step : **Merge with product line feature model**
Step : **Prune resulting feature model**
Step : **Check new features**
Activity : **Manage Dependences / Conflicts**
ProcessRole : **System Analyst**
WorkProduct : **Product Feature Model**
WorkProduct : **Product Conflicting Features Model**
WorkProduct : **Product New Features List**
Steps
Step : **Analyze conflicting feature and required changes**
Step : **Analyze changes effect onto related features**
Step : **Decide if implement a new component o modify existent**

WorkDefinition : **Manage the Scope of the System**
Activity : **Prioritize Features / Use Cases** (NEW)
ProcessRole : **Software Architect**
WorkProduct : **Software Architecture Document [Feature/Use Case View]**
Steps
Step : **Prioritize Features**
Step : **Prioritize Use-Cases and Scenarios for each Feature**
Step : **Document the Feature/Use-Case view**
Step : **Evaluate the result**

WorkDefinition : **Refine System Definition**
[Only for new features or modified]
Activity : **Detail Feature** (NEW)
ProcessRole : **System Analyst**
WorkProduct : **Product Feature List**
Steps
Step : **Detail characteristics abstracted by Feature**

Step : **Relate Features with other Features**
Step : **Insert Features into Feature Model**
Step : **Evaluate the results**

WorkDefinition : **Manage Changing Requirements**
Activity : **Review Features** (NEW)
ProcessRole : **System Analyst**
WorkProduct : **Product Feature List**
WorkProduct : **Product Conflicting Features Model**
WorkProduct : **Product New Features List**
Steps
Step : **Analyze new Features needed**
Step : **Analyze Features changes needed**
Step : **Analyze new / changed Features relation with PL Feature Model**
Step : **Evaluate the result**

Discipline : **Analysis & Design**

Subactivities
WorkDefinition : **Define Architecture Candidate**
Activity : **Derivate Product Line Architecture** (NEW)
ProcessRole : **Software Architect**
WorkProduct : **Core Architecture**
Steps
Step : **Select PL Architecture assets** (from features)
Step : **Integrate assets**
Step : **Evaluate your result**
Activity : **Use-Case Analysis** (only for no implemented features – NEW CONDITION)
WorkDefinition : **Refine Architecture**
Activity : **Identify Features** (NEW)
ProcessRole : **Software Architect**
WorkProduct : **Product Feature Model**
WorkProduct : **New Features List**
Steps
Step : **Identify new characteristic**
Step : **Classify Feature**
Step : **Relate Features with other Features**
Step : **Insert Features into Feature Model**
Step : **Evaluate the results**
Activity : **Review Design**
ProcessRole : **Architecture Reviewer** (NEW)
WorkProduct : **Product Line Change Request**
Steps
(NEW) Step : **Review Feature**

WorkDefinition : **Analyze Behavior** (only non existing features)
Activity : **Identify Features** (NEW)
ProcessRole : **Software Architect**
WorkProduct : **Product Feature Model**
WorkProduct : **New Features List**
Steps
Step : **Identify new characteristic**
Step : **Classify Feature**
Step : **Relate Features with other Features**
Step : **Insert Features into Feature Model**
Step : **Evaluate the results**
Activity : **Review Design**
ProcessRole : **Architecture Reviewer**
WorkProduct : **Product Line Change Request**
Steps
(NEW) Step : **Review Feature**

Discipline : **Configuration & Change Management**

Subactivities
WorkDefinition : **Manage Change Request**
[If it's a change request of a product line component, the submission must be done to the product line team; in this case the change request will be a process line change request]

Discipline : **Deployment**

Subactivities
WorkDefinition : **Produce Deployment Unit**
Activity : **Submit Insert Component Request** (NEW)
ProcessRole : **Deployment Manager**
WorkProduct : **Insert Component Request**
Steps
Step : **Complete Insert Component Request**
Step : **Submit the Insert Component Request**

Discipline : **Project Management**

Subactivities
[The first activity is already done by Product Line Management, then is eliminated]
WorkDefinition : **Close-out Project**
Activity : **Resume Project** (NEW)
ProcessRole : **Project Manager**
WorkProduct : **Product *Mecano* Description Document**
Steps

Step : **Create Project Functional Descriptor**
Step : **Evaluate Results**