Introducing Systematic Reuse in Mainstream Software Process

Miguel A. Laguna  
Univ. of Valladolid  
mlaguna@infor.uva.es

Bruno González-Baixauli  
Univ. of Valladolid  
bbaixauli@infor.uva.es

Oscar López  
Tech. Inst. of Costa Rica  
olopez@infor.uva.es

Francisco J. García  
Univ. of Salamanca  
fgarcia@usal.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 product line engineering and the adaptation of Unified Process for specific product construction. This proposal reduces the money and time costs related to the progressive introduction of software reuse in an organization. Some tools that provide support to the process, including a requirement tool and a repository of reusable elements, have been developed.

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 [3].

Finally, product lines (PL) 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 [14].

However, product lines is a very complex concept that requires a great effort in both technical – architecture definition, development, usage and instantiation [10,3,4] – and organizational – business view [1] – dimensions. In addition, the standard proposals of the software development process traditionally ignore the reuse issues, in spite of their recognized advantages [11]. 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 an 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.

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 [8], 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 [12] or Bosch [3] and a process adaptation, based
on a conventional software process, where some changes are introduced.

The proposal is founded on a coarse-grained reuse model and a related reuse library to manage the elements that offer the operative support to the reuse process [7,8]. Figure 1 reflects the difference between the product and product line development processes. Product line engineering and asset management are continuous processes without external observable output. Product process is iterative but has a final release as relevant difference.

![Figure 1. Domain and Product Engineering processes](image)

Nowadays, two main approaches for software development are in dispute: the lightweight, agile proposals and the heavyweight, highly configurable approaches. Extreme Programming (XP) [2] is the best known representative of agile processes and Unified Process (UP) [11] 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 (this is actually a required step defined in UP itself). XP is an ad hoc process, difficult to scale or tailor [22]. 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. Process has also a time dimension, with milestones that represent the completion of activities. We must define, therefore, the following dimensions of process: time aspects, artifacts, activities, and roles. A recent initiative, the Software Process Engineering Meta-model (SPEM) provides a language for defining processes and their components [18]. With this support we have proposed the required modifications to the UP disciplines, in order to facilitate the smooth introduction of the activities related to development with reuse. Another parallel process, specific for the development for reuse has been defined. In this case, a process different from UP has been elaborated, although with the same iterative and incremental philosophy.

The rest of the paper is distributed as follows: The next section shows the Product Line Engineering Process and section 3 outlines how UP must be changed to extract benefits from this approach. In both sections, due to space limitations, emphasis is put on technical aspects. Section 4 presents a series of tools that support these processes, and section 5 concludes the paper and proposes additional work.

### 2. Product Line Engineering Process

Generally, the idea of establishing a product line in an organization emerges in mature environments, where utilities or common components for development of new software products may have been identified. This suggests that the organization should have a level 2 or 3 in CMM (Capability Maturity Model) scale [19], although the organization has had no experience in software reuse. Reifer [20] has proposed a set of additional key reuse areas to be included in the CMM catalog. In particular, he cites Domain engineering, Architecture engineering or Asset management. Additionally, the ISO organization has recently amended his classical 12207 Software life cycle processes standard, which now includes Asset Management Process, Reuse Program Management Process and Domain Engineering Process [9].

Our proposal consists of an iterative process with three main phases (product line inception, elaboration and construction) and several technical and non-technical disciplines: Domain definition, PL Requirement engineering, Reference architecture definition, Component implementation, Test, and Asset management and quality assurance. 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. Some of the non-technical disciplines, such as Project management, Environment, or Configuration management, are common to both processes (and control and unify them) but Asset management is a specific discipline in product line engineering. 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 [6]. Some experiences 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.
2.1. Product Line Engineering Phases

Product line Inception phase

The basic purpose of the phase of PL Inception is the selection of the concrete application domain, properly focused with a wider strategy according to the global interests of the organization [21]. Therefore, the main discipline is the Domain definition. Bosch distinguishes two approaches to initiate a product line inside a domain [3]: the product line is based on a previously developed product family and the experience in the development of these products, or the organization initiates a product line from scratch. Some registered experiences allow us to conclude that an organization without previous practice in software reuse does not begin 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 relevant information about the domain must be collected, and its limits must be set. A first domain analysis and an architectural prototype can be built. In a well-known domain, these first steps should be dynamic. The milestone of the phase is the establishment of the domain’s basic goals, its scope, and an initial domain analysis that guides the initial reference architecture definition. 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 requirement document (with commonalities and variabilities clearly determined) and the creation of the architecture definition of the product line. This architecture must be validated by means of a partial architecture (used as a proof of the architecture suitability). An important artifact obtained in this phase is the components building plan with the scheduling of component construction. The disciplines involved in this phase are mainly PL requirement engineering and PL reference architecture definition. Some component construction must be completed for building the proof architecture.

At this moment, the product engineering process is enabled (at least the disciplines of business modeling, requirements and analysis & design) since 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 were designed at previous phases, but it is necessary to define all the internal issues) and the common and variable components are designed, implemented and tested. Then, these components are qualified and inserted in the asset repository. This must be done for each component or set of components. For this reason, the phase can be split in several parallel sub-phases.

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

2.2. Product Line Engineering Disciplines

Domain definition

The intention of this discipline is the study of the domain’s basic goals, its scope and definition. The first step consists of analyzing all the available information about the possible applications (related to the product line) and describing the involved sub-domains. Market and business analysis must be done to decide if a product line approach is cost-effective. This information is also useful for selecting exemplars. Next, the information collected is analyzed to set the domain scope and boundaries. Some exemplars must be described to obtain new vocabulary items. This description will be used to find the product requirements and to detect commonality and variability between them. The glossary is essential for maintaining the consistency of the requirements and for identifying commonalities and variabilities inside the product line. For this reason, the continuous upgrade of the glossary is also critical.

The deliverables collect the domain’s basic goals, its scope and definition, an initial domain model, a list with the exemplars and a first version of the product line glossary.

Product line requirement engineering

Requirement determination and management of a product line are activities that greatly influence the quality
of its products. Additionally, tasks and techniques are not exactly the same practices used in conventional methods for eliciting and analyzing software requirements in an independent product. The conclusion is that current practices in requirement engineering do not support PL 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.

In addition to the information that expresses the requirements themselves, it is important to know the variability of the requirements, and the dependencies between them. To represent this kind of information, the requirements are usually structured in definition hierarchies [15]. In our process, this discipline is based on FORM (Feature-Oriented Reuse Method) [12]. 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 to organize the results into a feature model, which is used to refine the domain model. 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 [16]. Likewise, in each category, the features are organized by a graphical AND/OR hierarchy diagram, i.e. the feature graph or feature diagram, which captures the logical structural relationships between requirements.

![Figure 2. Feature elicitation strategies](image1)

Requirement elicitation can also be based on use case analysis (use cases is usually a more familiar technique). The question of which analysis must guide the other depends on the PL requirements analyst and his knowledge of the domain or the domain expert’s availability. If the analyst has experience and domain experts are available, the best strategy is a feature-driven one; otherwise, a use-case-driven strategy is better (see figures 2 and 3) [4]. This analysis must be done starting from the domain definition (scope and boundaries can help) that provides a first cut and for every exemplar, in order to obtain the commonality and variability of the product line. An important issue is the integration of each exemplar with the rest: we must check there are no conflicts or repeated functionality.

The deliverable of this discipline is the set of reusable assets representing the functional descriptors of 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).

![Figure 3. Activities of PL requirement engineering](image2)

**Product line reference architecture definition**

Once the basic PL requirements are determined, this is the most critical activity in the initiation of a product line from a reuse perspective: 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 PL applications requirements (actual and further) and be flexible enough to include product specific components. The design of the reference architecture is probably the most creative aspect of the overall process, and accordingly, the most difficult to standardize. The experience of the software architect and the kind of products determine the definition of the PL reference architecture. In the case of well-known domains, the use of classic architectures, such as client-
server, will be enough, but in more complex or undefined situations, the entire architect inventive will be required.

This discipline is essentially equivalent to the implementation discipline of UP, with the inclusion of a new activity, Integrate components (figure 5). Initially, the reference architecture and next the rest of the components are implemented and/or integrated. These components can be designed from scratch, bought, commissioned or incorporated from mined code of existing software artifacts (either by applying different design-for-reuse techniques such as refactoring, or by developing wrappers). The deliverables of this discipline are the PL components, each one as an independent asset related to the PL reference architecture.

**Figure 4. Activities of PL architecture definition**

As a guide, the activities of this discipline are: from the domain description and detailed requirements, an initial architecture must be defined, analyzed and refined until a possible solution is obtained. Once the architecture is well defined, a subset of this architecture is selected to validate its suitability for the PL requirements (proof architecture). Finally, if the architecture is approved, the design is completed by mining, designing or buying/commissioning the components. Figure 4 shows the activities of the discipline.

The deliverables of this discipline are the architecture structure and a set of component analysis and design. It is important to register the traceability of every component from its requirements and implemented features.

**Component implementation**

The design and implementation of a product line, and in particular its reference architecture, requires the construction of the set of components that compose this basic architecture.

**Test**

The contents of this discipline are equivalent to standard UP. Unit testing is carried out in the implementation discipline but a careful integration testing process must be carried out to verify the proof and reference architectures. Deliverables of this discipline are the test plan, test procedure and test evaluation documents.

**Asset management and quality assurance**

Most non-technical disciplines must be shared between PL and product engineering processes. Nevertheless, in
particular, management of the assets that form the product line is a key question. In the course of 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. A reuse library or repository offers the operative support for the storage and management of the PL artifacts.

The repository connects the domain engineering and the application engineering processes, allowing the cycle to close [13]. In our proposal, it would be desirable that the organization had a repository engine that allows the management of assets (see section 4 for details).

The deliverable of this discipline is a qualification report obtained as a PL element is introduced in the repository.

3. Product Engineering Process

To introduce a product line approach in an organization, some changes to the product development process are desirable. These are mainly required to manage three issues: the derivation of the PL features subset for the application, the previous existence of reference architecture and the presence of a repository where the new reusable components must be inserted.

Taking UP as the starting point, the main changes must be made to the Requirements discipline: the product features must be completed starting from the PL features subset for the application, the previous existence of reference architecture and the presence of a repository where the new reusable components must be inserted.

In this discipline, it is only necessary to select a subset of the PL requirements and complete them with the 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 approach, a feature model is required to guide the instantiation of the product architecture from the PL reference architecture. This feature model is constructed from the use case model: each use case is analyzed and the features obtained. It is essential to check each feature with the PL feature model and to add new features only if necessary (there is no feature on the PL with this functionality and it is not possible to modify another to comply with it). Finally, it is necessary to contrast the feature model with the PL feature model and to insert all required features. From this checking, two lists are created: new features and features that need changes (to be constructed in the next disciplines). New deliverables are the product feature model, the list of new features and the list of conflicting features.

Analysis & Design

The main change in Analysis & Design is the PL instantiation, once the feature model is finished. This guides all the architecture construction. From this core architecture the rest of components are built by creating or modifying PL components. This is done in the typical UP activities, but the candidate architecture is created as an instantiation of the PL reference architecture. Then, following UP activities, this architecture will be refined, and the components that must be implemented will be analyzed and designed. It is vital to decide if the new features and components must be implemented for reuse (if a functionality useful to other PL applications exists and the cost increment is affordable). If such a decision is taken, the construction should be controlled by PL engineers or even done by them.

There are no new deliverables, but new artifacts are created as the architecture instantiation (product candidate architecture) is refined or new components are required.

Implementation and Test

Implementation and Test disciplines do not suffer substantial changes, but the focus varies according to the scope of the configuration or project management problems. This kind of problems must take account of the overall product line approach: every project must be considered a sub-project of the product line project. The changes in disciplines are outlined in the next paragraphs.

Business Modeling

Business modeling discipline is simplified in our approach, because its goal, mainly to understand the organization structure and problems, is achieved in the domain engineering process for the complete product line.
modified components. In these cases, regression tests are useful.

Configuration & change management

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

Deployment

This discipline is responsible of insertion of new components into the general product line repository. Therefore, a new activity must be introduced: Submit insertion component request. This provides a new component to be qualified by product line engineers, and then inserted into the general repository. Component insertion request is the new deliverable added.

Project management

Essentially, the activities at this discipline are unchanged, but the existence of a study about the suitability of the application inside the product line is usually necessary, except when it is one of the exemplars used to define the product line. New deliverables are the product suitability document and the product descriptor document.

Environment

This discipline doesn’t change, but is important to pay special attention to the global repository. Additionally, some new tools that support the feature analysis and product line management (the asset library) must be provided. This is focused in the next section.

4. Tool support

To be successful, this approach to product line development needs some tools that support the new activities defined. We have initially developed an asset repository that implements the reuse model [7]. The main interest of this model is the established traceability between requirements, designs and code. The access to the repository is granted through the web of the GIRO\(^1\) group (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 construction of tools that connect standard CASE tools with the repository. An API for insertion and extraction of assets 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, 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 the relationships 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 on 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.

\(^1\)In Spanish, “Grupo de Investigación en Reutilización y Orientación a objetos”, Research Group in Reuse and Object-Orientation
5. Conclusions and future work

In this paper, we have introduced a product line process that does not require great effort, time or money investment. This approach smooths 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 reuse 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 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 developing 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. A workflow-based tool for multi-user process support is also one of our future goals.

Acknowledgements

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

References