Bayer, J., Gacek, C., Muthig, D. & Widen, T. (2000). PuLSE-I: Deriving instances from a product line infrastructure. In: Engineering of Computer Based Systems, 2000. (ECBS 2000) Proceedings. Seventh IEEE International Conference and Workshop. (pp. 237 - 245). London, UK: IEEE Computer Society. ISBN 0-7695-0604-6

PuLSE-I: Deriving Instances from a Product Line Infrastructure

Joachim Bayer, Cristina Gacek, Dirk Muthig, Tanya Widen

Fraunhofer Institute for Experimental Software Engineering (IESE)
Sauerwiesen 6, D-67661 Kaiserslautern, Germany
{bayer, gacek, muthig, widen}@iese.fhg.de

Abstract

Reusing assets during application engineering promises to improve the efficiency of systems development. However, in order to benefit from reusable assets, application engineering processes must incorporate when and how to use the reusable assets during single system development. However, when and how to use a reusable asset depends on what types of reusable assets have been created.

Product line engineering approaches produce a reusable infrastructure for a set of products. In this paper, we present the application engineering process associated with the PuLSE product line software engineering method — PuLSE-I. PuLSE-I details how single systems can be built efficiently from the reusable product line infrastructure built during the other PuLSE activities.

1. Introduction

Application engineering is the development of a single software system. During application engineering, in order to save time and effort, developers often practice ad-hoc reuse of software assets.

This ad-hoc process has many problems. First of all, locating candidate reusable assets is problematic. Developers typically only look for what they already know about, thereby, missing many other candidates. Second, once assets are found, it is often not possible to directly reuse them. This is because the assets were developed with certain assumptions about their environment, which are often not documented. These uncaptured assumptions often affect how the asset must be adapted. They may also determine whether the asset can be reused at all. Finally, managers have trouble estimating the development effort because they do not know what might be reusable and how much effort it could take to reuse what is available. Due to these problems, the original reason to try to reuse assets, which is to save time and effort during development, is not realized.

Product line engineering is an approach to improve development efficiency through reuse for families of systems whose functionalities overlap. Product line engineering methods typically focus on building a reuse infrastructure, which contains reusable assets, that can be used for efficiently building the members of the product line. The single products are assembled from reusable assets, and can therefore be developed more efficiently. Additionally, the product line infrastructure supports locating candidate assets for reuse, evaluating these assets, adapting the assets, as well as helping managers estimate and plan more accurately.

However, in order to benefit from the reuse infrastructure, defined methods for application engineering with the reusable assets must accompany the processes for developing the reusable infrastructure. These methods are necessarily tightly coupled with the assets to be developed during product line engineering.

In this paper, we present PuLSE-I, the application engineering process of PuLSE™ (Product Line Software Engineering)[1]. PuLSE is a full life-cycle product line process [2]. The application engineering process is centered around the instantiation of the product line infrastructure (the I in PuLSE-I stands for instantiation).

PuLSE-I is tightly dependent on the outputs of the other PuLSE activities, therefore, we present an overview of PuLSE in Section 2. Section 3 presents PuLSE-I, the complete process and its underlying steps. Subsequently an analysis of our approach and discussion of related work is given in section 4. Section 5 concludes the paper.

2. PuLSE

PuLSE is a method for enabling the conception and deployment of software product lines within a large variety of enterprise contexts. This is achieved via a product-centric focus throughout its phases, customizability of its components, an incremental introduction capability, a maturity scale for structured evolution, and adaptions to a few main product development situations.

Figure 1 shows an overview of PuLSE.

PuLSE is centered around three main elements: the deployment phases, the technical components, and the support components.

The *deployment phases* are logical stages of the product line life cycle. They describe activities performed to set up, use, and evolve product lines. The deployment

1. PuLSE is a registered trademark of the Fraunhofer IESE

phases are:

- PuLSE initialization: PuLSE is customized to the context of its application. The principle dimensions of adaption are the nature of the domain, the project structure, the organizational context, and the reuse aims.
  The initialization phase is realized by the technical component for customizing, PuLSE-BC.
- Product line infrastructure construction: The product line infrastructure is set up. This is done by scoping, modeling, and architecting the product line.
  These activities are realized by the corresponding technical components PuLSE-Eco, PuLSE-CDA, and PuLSE-DSSA, respectively.
- Product line infrastructure usage: The product line infrastructure is used to create a single product line member. This is done by instantiating the product line model and architecture.
  The PuLSE-I technical component realizes this phase.
- Product line infrastructure evolution: Concepts within the domain or other requirements on the product line may change over time. The evolution of the product line is handled in this phase.
  The process for controling evolution is realized by the PuLSE-EM technical component.

The *technical components* provide the technical know-how needed to operationalize the product line development. They are used throughout the deployment phases. The technical components are:

- Baselining and Customzation (PuLSE-BC): Baseline the enterprise and customize PuLSE. The result is an instance of PuLSE — that is, instances of the other technical components — tailored to the specific application context.
- Economic scoping (PuLSE-Eco [6]): Identify, describe, and bound the product line. This is done by determining the characteristics of the products that constitute the product line. Economic scoping in PuLSE means that the scope is determined with respect to business objectives and planned products.
  The output of PuLSE-Eco are the product characteristic information and the scope definition. These outputs together describe the contents of the product line. The product characteristic information describes the common and variable characteristics of all products in the product line.
  The scope definition identifies the range of characteristics that systems in the product line should cover. The basis for the scope definition is a product map that relates the characteristics to the different products. A product map is a table, which lists the characteristics mentioned in the product characteristic information as its rows and the products as its columns. The table cells contain a cross when a product contains a characteristic.
  To determine the scope, the benefit provided by including a characteristic into the scope relative to business objectives is determined. The scope definition is then an identification of a subset of the characteristics that shall be developed for reuse.
  The benefit is calculated with functions. Characterization functions describe the benefit of having a certain characteristic in a certain product. The business objectives are expressed in terms of benefit functions that describe the benefit accrued by integrating a certain characteristic into the product line scope. By gathering values for the characterization functions, the benefit functions can be solved to determine the appropriate scope.
- Customizable Domain Analysis (PuLSE-CDA [3]): Elicit the requirements for a domain and document them in a domain model (a.k.a. product line model).
  A product line model is composed of multiple workproducts that capture different views of a domain. Each view focuses on particular information types and relations among them. In the workproducts, common requirements (commonalities) and requirements that vary for the different systems (variabilities) are modeled. Therefore, they are referred to as generic workproducts. There are three types of variabilities: optional, alternative, and range requirements.
  Each generic workproduct has defined meta elements for each variability type. Meta elements indicate points of variation and enable the instantiation of the

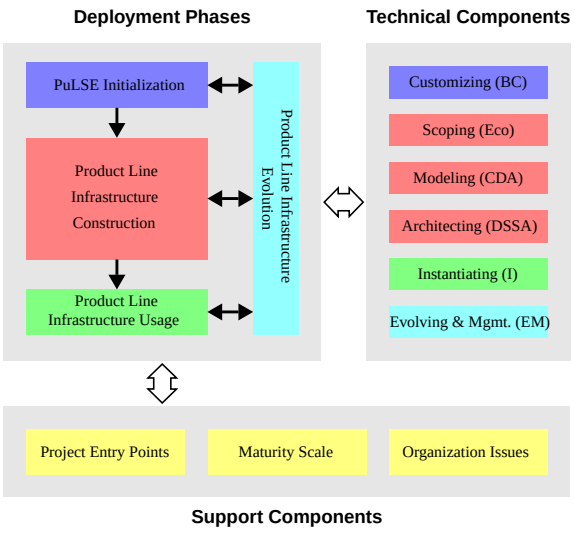

Figure 1. PuLSE Overview

workproducts.
The variabilities (expressed by meta elements) are connected to decisions that, when completely resolved, specify a particular system, a member of the product line. The decisions are at different levels of abstraction and are hierarchically structured based on constraints among them. The decision hiearchy is called the domain decision model.
To specify a particular system in the product line, the product line model is completely instantiated. The instance of the product line model is generated by passing all resolutions of the decisions to the connected meta elements, which instantiate their corresponding part of the product line model.

- Domain Specific Software Architecture development (PuLSE-DSSA [5]): Develop a reference (or domain specific) architecture based on the product line model. A reference architecture description consists of multiple models that describe different views on the reference architecture. Each of the views is composed of view-specific components and connectors that describe the architecture from a different perspective. Similar to a product line model, a reference architecture description is an architecture description that also captures variability in the architectures for the different systems in the product line.
  During the reference architecture development, certain decisions arise that are not driven by the domain. These decisions may introduce domain-independent variabilities. The resulting decision model is called the architecture decision model.
  An optional output of PuLSE-DSSA is a prototype that may have been created.
- Instantiation (PuLSE-I): Specify, construct and validate one member of the product line. This encompasses the instantiation of the product line model and the reference architecture, the creation and/or reuse of assets that constitute the instance, and the validation of the resulting product. Additionally, reusable assets that are needed, that have not been created yet, are developed and put into the reusable asset base.
- Evolution and Management (PuLSE-EM): Guide and support the application of PuLSE throughout the deployment phases initialization, construction, usage, and evolution.
  PuLSE-EM is centered around three basic tasks: product line management, evolution, and learning. Product line management provides means for scheduling and coordinating the technical components, as well as for observing the product line and its environment to be able to respond quickly to emerging needs. Product line evolution supports systematic change request processing. This includes the evaluation of change requests and the assessment of their effects on existing parts of the product line infrastructure. Learning analyzes the product line and changes that occur over time. The goal is to learn about patterns of product line evolution that would allow for acting in anticipation of future problems, needs, or changes.
  Additionally, PuLSE-EM includes the configuration management framework that underlies and supports the product line infrastructure.

The *support components* provide guidelines that support the other components. They are:

- Project entry points: Project entry points are guidelines to customize PuLSE for a set of standard situations. For example, in reengineering driven PuLSE projects, legacy assets are a major source of information and guidelines on how to integrate them are given in the respective entry point.
- Maturity scale: It is used to evaluate the quality of a PuLSE process application in enterprises with the intention to identify and improve weak points. The levels on the scale are: initial, defined, controlled, and optimizing.
- Organization issues: For PuLSE to be most effective, an organization structure has to be set up and maintained that supports the development and management of product lines. Guidelines on how to do that are given here.

3. PuLSE-I

PuLSE-I uses the product line infrastructure to create and maintain one member of the product line. The PuLSE-I process is illustrated in Figure 2. The trigger for starting PuLSE-I is a customer or the management having a product request that can be satisfied by the product line (i.e., the requested product is potentially in the scope of the product line). Based on the product request and the scope definition developed in PuLSE-Eco, a plan is created for the development of the requested product. This step is discussed in more detail in section 3.1.

The specification for the product is developed by instantiating the product line model. Driven by the domain decision model and using the generic workproducts from PuLSE-CDA, the new product is specified. The instantiation of the product line model is described in section 3.2.

Driven by the architecture decision model and using the product specification, the architecture for the new product is derived from the reference architecture. The instantiation of the reference architecture is described in section 3.3.

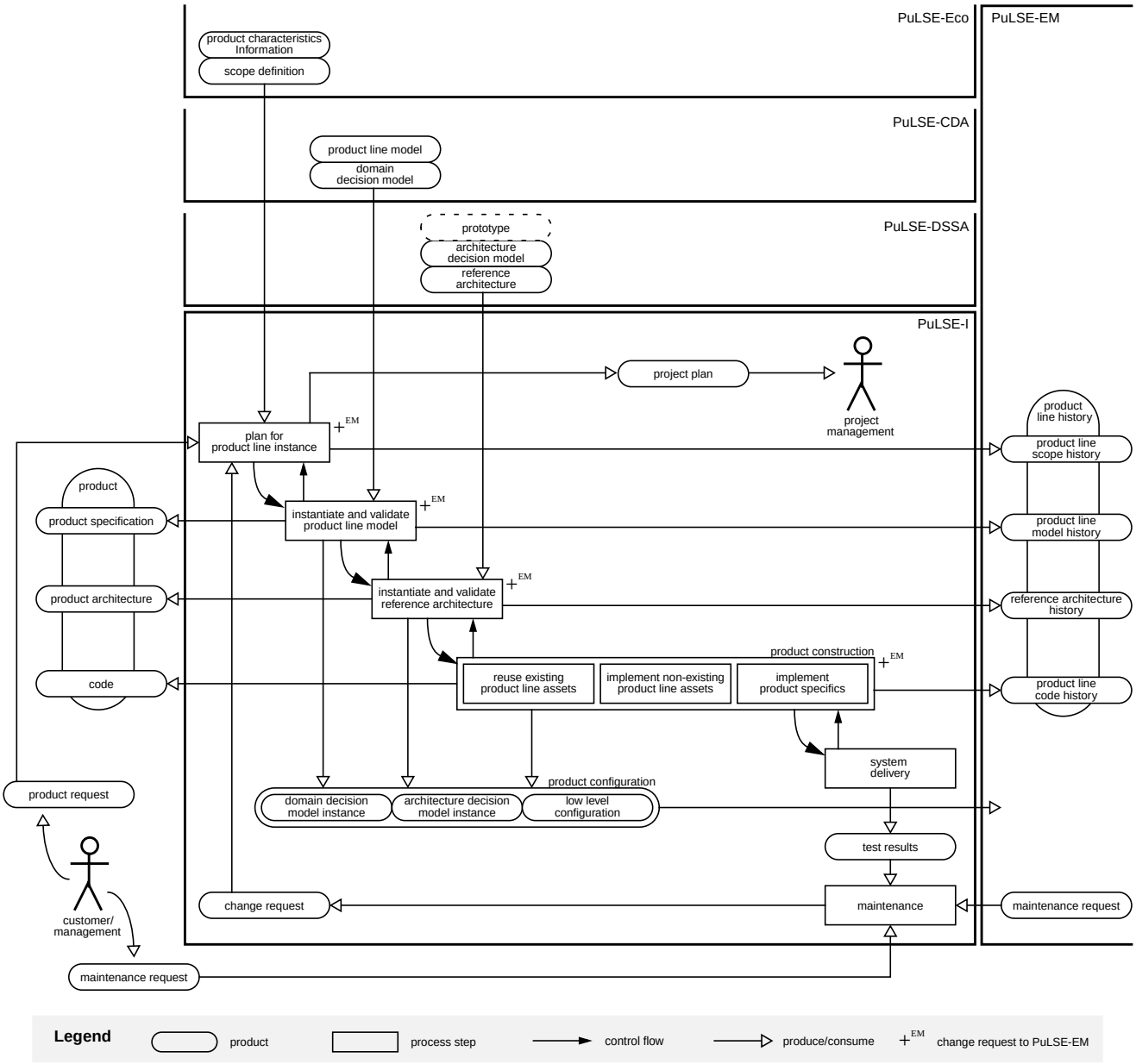

Figure 2. PuLSE-I Process

In the next step, the product is assembled from assets. These assets are the result of one of the following activities: reuse of an existing product line asset, implementation of a non-existing product line asset, or implementation of a product-specific asset. These steps are described in section 3.4.

After the product has been finished, an acceptance test is performed before the system is delivered (section 3.5). When this test has been passed, the product is deployed and enters its maintenance phase. Maintenance of products created on the basis of a product line is discussed in section 3.6.

3.1. Plan for Product Line Instance

PuLSE-I is started when there is a request for a new product. If the product was considered during scoping, all characteristics this product must have are covered by the product line infrastructure. That is, they are in the scope definition of the product line, integrated in the product line model, and supported by the reference architecture. Hence, all information needed for planing the project is available.

However, as product lines are long term investments, often new products that were not considered up front will

be requested, or individual customers may want customized versions of the planned products. In this case, an activity must take place to gather the needed information before a project plan can be created. That is, the overlap between the required system and the current scope is evaluated and the realization effort for features, characteristics representing functional requirements, beyond the product line scope is estimated.

To get the needed information, first, the characteristics required by and excluded from the new system are listed. A new column is added to the product map and all required characteristics are checked. Each additional characteristic, which is beyond the current product line boundaries, is integrated into the product map. Then, the extended product map is passed as a change request to PuLSE-EM, the management and evolution component of PuLSE.

In EM, the extended product map is forwarded to PuLSE-Eco and the scoping process is re-entered to decide for each additional characteristic whether it will be integrated into the product line infrastructure. If necessary, other technical components, the product line modeling and architecting components, are involved in these decisions. The result passed back to the planning step of PuLSE-I is the new scope definition, which represents the (potentially) adapted scope of the product line, as well as the expected effort for the integration of the new characteristics.

The realization of required characteristics that are still outside the product line scope must be planned as system-specific assets (i.e., independent of the product line infrastructure). Later, these assets must be integrated with system parts based on reused product line assets. If this is not possible for all characteristics, some even required characteristics must be refused because they will conflict with parts of the product line infrastructure. In extreme cases, when such a necessary exclusion is not acceptable for the customer, the system may be developed but not considered as part of the product line.

The result of the evaluation of the overlap between the required system and the (potentially changed) product line scope is the list of characteristics that the final system will have. This list includes effort estimation for system-specific characteristics, as well as for characteristics within the product line scope. This information is equivalent to the information available when PuLSE-I is started for systems that have already been considered during the initial scoping of the product line. Hence, the step for creating the project plan, which uses the list as major input, can be started.

The creation of a project plan for the development of a product line member in PuLSE-I does not significantly differ from the creation of a project plan for the development of an individual product (i.e., one that is not built using a product line infrastructure) [9]. However, the estimation of reuse and thus of effort is based on more reliable and explicit experience, as well as the risk of planning for incompatible features is reduced.

In addition, the understanding of development costs for single characteristics enables a more substantial negotiation between developers and customers, or developers and marketing. It enables developers to propose alternative characteristics that reduce costs, to offer additional characteristics that can be integrated with little effort (but add value to the product), as well as to identify and plan useful product increments.

The final result of the planning step is a detailed project plan. This plan considers the set of characteristics upon which the customer (or the marketing) and the developers have agreed.

3.2. Instantiate Product Line Model

The product line model, which has been created during the infrastructure construction phase, represents the requirements for the whole product line. The result of its instantiation is equivalent to the specification of a particular system, a member of the product line. That is, each required characteristic is specified by a detailed description of how it will be supported.

Each characteristic specification uses the same product model but there are two different processes for creating them. The process depends on whether the characteristic is covered by the product line scope, or whether it is system-specific.

When it is in the scope of the product line, the characteristic points to a set of top-level decisions in the decision model. Where, a decision is an issue that represents a domain-specific variability. It consists of a set of possible resolutions and the descriptions of their impact on variation points within the product line model, or the constraints on the resolution of other decisions.

For specifying the product line instance, all decisions that are constrained by top-level decisions to which at least one of the required characteristics points and all remaining decisions that have an impact on used assets must be resolved. The constraint path defined by the relevant top-level decisions through the decision hierarchy guides the specification process and, thus, simplifies the complete requirements elicitation. During this process, decision by decision is resolved driven by the customer, and thus the product line model is incrementally instantiated. The instantiation result is defined by the impacts of each chosen resolution in the decision model. With a tool (e.g., DIVERSITY/CDA [4]), the impacts of decisions made can be visualized interactively enabling an immediate valida-

tion and feedback loop.

It is possible that none of the provided resolutions result in a sufficient specification, that is, the requirements from the current customer for some characteristic are not captured in the product line model. In such a case, the uncovered parts of the requirements are manually modeled in the context of the partially instantiated product line model. Additionally, a change request is sent to PuLSE-EM to request the modification of the product line model. The modification should enable the generation of the required specification. If this is not possible, alternative specifications, which fit into the existing infrastructure, may be proposed to the customer. In cases, when only specifications are acceptable that do not conform with the existing product line infrastructure, the features that cause problems must be ranked as system-specific. This must be done although the feature is covered by the product line at the more general characteristic level.

When a characteristic is not covered by the product line, its requirements are directly elicited like it is done in the development of single systems. However, the product model is given by the product line infrastructure. Therefore, the specifications used for product line and system-specific features are compatible.

Finally, the specification of all features are integrated into a single, unified document, the product requirements of the system to be developed.

3.3. Instantiate Reference Architecture

The reference architecture created during the infrastructure construction phase must be instantiated for the particular system at hand (i.e., a software system architecture is created, which is derived from the reference architecture and reflects the current requirements).

The instantiation of the reference architecture into a product-specific architecture is realized in two different steps. First the reference architecture must be refined into an intermediate architecture, that has all variabilities from the reference architecture instantiated. Decisions that were resolved during the instantiation of the product line model are used, yet the resolution of architecture specific decisions may also be required. When the currently considered resolutions for one or more architectural specific decisions are not satisfactory for the system under development, a change request is sent to EM which may result in the current requirements being revisited and changed, the architectural decision model being updated to cover the current needs, or the conflicting requirements being treated as instance specific ones. By the time all relevant decisions present in the architectural decision model have been resolved the resulting architecture does reflect no more potential variation points, however it still requires the instance specific parts to be added. The result of this step is the definition of which parts of the reference architecture should be reflected in the instance architecture, and which ones should not.

Subsequently, this intermediate architectural representation must be extended in order to accommodate the instance-specific requirements. That is, components and connectors that were not present in the reference architecture but are essential for achieving the features required must be included in the intermediate architecture in order to obtain a complete instance architecture. This addition of components and connectors is not a simple process. Special care must be taken so that architectural mismatches are avoided when possible [7]. When such mismatches are unavoidable, they must be handled appropriately (e.g., by using wrappers or instrumented connectors [1]). Depending on the kind of mismatches encountered, it may even be required that some of the reference architecture items be modified to support this instance. In that case the modified reference architecture items would be treated as instance-specific items and not as part of the asset base.

The resulting product architecture is then used for product construction.

3.4. Product Construction

Based on the product architecture, lower level design, implementation, and testing must occur. This can be done by (adapting and) reusing existing product line assets, implementing non-existing product line assets, or implementing product specific parts. Which assets should be built as part of the product line and which ones should be considered instance specific is decided while scoping the product line.

Parts that are common to the product line may already exist. If so, these may be reused, if not, these should be developed for reuse. Parts that are instance specific must also be implemented. All resulting parts must be integrated and tested. This overall process should preferably be done in an iterative manner.

The existence or not of reusable assets is determined by following links between assets. During a PuLSE application, traceability links are established. These are pointers between assets that facilitate both assets search and their consistent evolution. The domain model has concepts that are reflected in the reference architecture, the reference architecture has lower level design and code assets that implement it, and so on. All of these relations among artifacts are then reflected via links for both traceability purposes as well as for instantiation and implementation support.

Reusing existing product line assets consists of finding the appropriate assets, adapting them as needed, perform-

ing unit test on them to confirm that they may really support the current need, integrating them in the overall product being built, and testing the resulting product. Locating the appropriate assets is achieved by means of traceability links from reference architecture to lower level design and code. Since the product line asset base is kept under configuration management, the usage of its assets is documented and tracked.

Assets that are required in the product line asset base but do not yet exist must be developed. Since these assets are to be part of the product line asset base, their foreseen uses are already reflected in both the domain model and reference architecture. Lower level design of these parts must be done in a generic fashion in order to support all foreseen uses it may have. Clearly the actual code implementation should follow such generic low level design. The assets that have been built for reuse must be thoroughly tested and potentially inspected. Their unit test must reflect the current usage as well as the expected ones, which may be achieved by simulating expected future interfaces. After the newly developed asset has been thoroughly tested and approved, it should be added to the product line asset base[1] and placed under configuration management accordingly. At that time the asset should also be integrated in the product under development and undergo integration and/or system test.

The reuse infrastructure and reusable assets for specific product lines may adopt different implementation approaches. Reusable code assets may be black box components, parameterizable black box components, templates, customizable via scripts, or require manual adaptations. Hence, upon retrieval of a previously existing reusable asset or the development of a new one, some adaptation effort may be required. The amount of adaptation effort required is a function of the implementation approach adopted by the asset.

As previously discussed, specific products will also contain some unique requirements which must be fulfilled. Their design, implementation, and testing are performed according to criteria different to those imposed on the reusable assets. They must reflect the current product requirements and instantiated architecture, but do not have the same genericity and flexibility needs as the ones imposed on the reusable assets. This is not to say that they will be handled with less rigour, this simply means that their acceptance criteria are less stringent. Still inspections, unit tests, integration, and integration tests must be performed.

1. Adding a reusable asset to the product line asset base involves actions like placing it under configuration management, introducing the appropriate links, as well as following the appropriate classification scheme being used.

3.5. System Delivery

After the product construction, which is finished when the system passed the integration test, the system is delivered. The delivery process depends on the market size.

For the mass market, the system must first be packaged together with an installation guide to enable any customer to install it at his/her machine. This includes the evaluation and recommendation of possible system environments. Before the system is publicly available, it enters a beta test phase. That is, the system is distributed among a selection of users to test it in real usage environments.

For small markets, like for individual systems for single customers, the system is installed by the developers at the customer's site. There are several strategies for introducing the new system into the customer's business processes. It can immediately replace the running system which is characterized by low costs, brief instability, and high confidence that the new system works. The opposite strategy, which duplicates work, is to run both systems, the old and the new one, in parallel. In most cases, the chosen strategy is a combination of these two: the new system replaces part by part the old one. Thereby, a part can be a system module whose functionality is replaced by the new system, or a subset of the customer environment that switches to the new system.

Independent of the introduction strategy, tutorial material, user documentation, and technical training courses ideally tailored to the needs of each user class must be planned and prepared. The introduction process for the new system (or at least parts of it) serves as acceptance test for the system.

The focus of both the beta testing and the individual introduction is on useability aspects, such as performance, security, reliability, and human factors (e.g., the user interface).

In a product line context, all activities during system delivery are repeated for each member of the product line. Therefore, the gained experience can be reused and ensures a high quality of the first system shipped to the customer. Of course, products like tutorial and training material can be reused like any other asset of the product line infrastructure.

3.6. Maintenance

After the system has been delivered and its PuLSE-I instance enters the maintenance phase. Maintenance is usually divided into four major categories: corrective, adaptive, perfective, and preventive [10]. The main goal of maintenance in the context of software product lines is that over the whole life-cycle all product line members are based on the same, common asset base. In PuLSE, the maintenance activities are distributed among the set of

PuLSE-I instances and PuLSE-EM, the product line infrastructure evolution and management component.

In this section, the maintenance process is described from a single PuLSE-I instance's point of view. Its maintenance process is triggered by maintenance requests initiated by users of the local system (i. e., the system that has been delivered by the PuLSE-I instance itself), or by tasks sent by PuLSE-EM because of changes made to the product line infrastructure.

When a user of the local system initiates a maintenance request, first, the maintenance must be categorized by the appropriate PuLSE-I instance to determine how it is handled.

For corrective maintenance, which handles error reports from users, the diagnosis of the occurred errors, as well as their local correction are done in the context of the maintenance request (i.e., in the context of the local system). If the correction impacts product line assets, the maintenance request together with the diagnosis and correction report are passed as change request to PuLSE-EM. Then, the locally corrected system is usually not released (only in critical cases) but the local PuLSE-I instance waits for the notification by PuLSE-EM when the changes are integrated into the common product line infrastructure. Hence, the maintenance process returns into its wait state.

For adaptive and perfective maintenance, which handle changes of the system environment and requests for new or modified capabilities, the planning step is re-entered to prepare a potential new iteration of the development process (see section 3.1). Preventive maintenance will never be requested by users of a specific system. It is done within the learning process of PuLSE-EM, which analyzes the evolution of the product line over time to predict changes anticipated for future products.

When the product line infrastructure has been changed because of a change request passed by any PuLSE-I instance to the product line management, PuLSE-EM sends notifications to all PuLSE-I instances that are in the maintenance wait state. When such a notification is received by a PuLSE-I instance, it checks whether the changed assets of the infrastructure have been used for generating its local system. If they have not been used, the local system and its future re-generations are not affected by the changes made. Then, the PuLSE-I instance returns immediately to the maintenance wait state.

In the other case, when the changed assets have been used, it must be ensured that the local system can be regenerated using the changed product line infrastructure. Therefore, the existing decision model instances are used. If the re-generation is possible, the product is still a member of the product line and thus can share the common product line infrastructure. Before the PuLSE-I instance returns into the maintenance wait state, it decides whether the regenerated system is released. This decision depends on the quality and quantity of the changes made since the last release of the local product.

If the re-generation of the product is not possible, a change request is sent to PuLSE-EM, which describes the problems occurred during the re-generation process. When the infrastructure can be modified to support both the latest maintenance requests and the requirements of the local product, the problems are solved and thus a re-generation can be successful. The maintenance process decides (as described above) upon releasing the generated product and then returns into its wait state.

If it is not possible to solve the problems concerning the generation, the local product is no longer a member of the product line. Consequently, there cannot be any releases or updates on the basis of the common product line infrastructure in the future. Except, the local requirements are changed to enable the product generation again. To decide upon useful adaptations of the requirements to the common infrastructure, the planning step is re-entered (see section 3.1). If no acceptable adaptation of the requirement can be found, the product is either maintained individually or can no longer be supported. However, the maintenance in the context of the product line infrastructure and thus the local PuLSE-I instance for the particular product is stopped.

4. Related Work and Analysis

PuLSE-I was developed to ensure a tight connection between the results created during product line infrastructure development and the process that uses these results. The PuLSE method is in the same category as domain engineering methods, for example, Organizational Domain Modeling (ODM), the Domain-Specific Software Architecture approach (DSSA), and Synthesis [12, 13, 11]. However, most of these methods do not supply specific processes for application engineering that define how to use the results produced for reuse. Synthesis does actually provide such a process, and in some respects PuLSE-I is similar to that process. Both application engineering processes are driven by a decision model. However, the assets in the reuse infrastructures created by PuLSE and Synthesis are different. Therefore, the application engineering processes differ by optimizing the benefits of the reusable assets produced.

PuLSE-I is also related to the process for Application System Engineering (ASE) described by Jacobson et al. [8]. However, the ASE process has no decision model. Therefore, all the models produced are checked for what is needed in the instance and system specifics are added to those chosen.

PuLSE-I has to also deal with many of the same issues

that more general reuse techniques, such as component libraries, deal with. These issues include finding, evaluating, adapting, and combining components. The PuLSE reusable assets: product line scope, domain model, and architecture, address all of these issues.

First of all, traces, or links, are maintained between the architecture to the reusable code components, so possible reusable components can be found with the instantiated architecture. Evaluation of the possible components to reuse out of a given set of candidates is also simplified through the explicit modeling of variabilities and their impact on components. Additionally, all of PuLSE's reusable assets (scope, domain model, architecture) provide information related to the context of use of reusable code assets, and, therefore, provide additional information (that is not available for generally reusable components) that help in adapting components. Finally, the architecture addresses the issue of how to combine components. There are still open issues concerning the creation of architectures that best support combining components.

5. Conclusion

To succeed with product line engineering it is necessary to have a systematic application engineering process that incorporates when and how to use the assets from the reuse infrastructure. This process enables product line members to be built efficiently. Such a process is necessarily tightly coupled with the type of reusable assets produced.

In this paper, PuLSE-I, the application engineering process for PuLSE, was presented. This process is centered around the decision models. The decision models, along with the traceability links between all other outputs, enable the identification of reusable assets at all levels from the top level decisions.

So far, experience with the process has been positive. However, it is still only described at a high level. We are currently working on a guidebook, with lessons learned from our experience, to extend the process.

Acknowledgements

The authors would like to thank all the other contributors to PuLSE: Oliver Flege, Peter Knauber, Roland Laqua, and Klaus Schmid.

References

[1] Balzer, R. "An Architectural Infrastructure for Product Families." In *Proceedings of the Second International ESPRIT ARES Workshop*, February 1998.

[2] Bayer, J., Flege, O., Knauber, P., Laqua, R., Muthig, D., Schmid, K., Widen, T., DeBaud, J.M. "PuLSE: A methodology to develop software product lines." In *Proceedings of the Symposium on Software Reusability* (SSR'99), May 1999.

[3] Bayer, J., Muthig, D., and Widen, T. "Customizable Domain Analysis," in *Proceedings of Generative and Component-Based Software Engineering Conference*, September 1999.

[4] Bayer, J., Muthig, D., and Widen, T. "Support for Domain and Variant Engineering: Diversity/CDA," *Submitted for publication*, 1999.

[5] Bayer, J., Flege, O., and Gacek, C., "Creating Product Line Architectures," submitted for publication, 1999.

[6] DeBaud, J.M. and Schmid, K. "A Systematic Approach to Derive the scope of Software Product Lines." In *Proceedings of the 20th International Conference on Software Engineering* (ICSE'99), pp. 34-43, IEEE Computer Society Press, 1999.

[7] Gacek C., *Detecting Architectural Mismatches During Systems Composition*, Doctoral Dissertation, Center for Software Engineering, University of Southern California, Los Angeles, CA 90089, USA, December 1998.

[8] Jacobson, I., Griss, M., and Jonsson, P. *Software Reuse Architecture, Process and Organization for Business Success*, ACM Press, 1997.

[9] Miller W. B., "Fundamentals of Project Management," in *Software Engineering Project Management* edited by Thayer R. H., IEEE Computer Society Press, 1997.

[10] Pressman R. S., Software Engineering; *A Practitioner's Approach*, McGraw-Hill Book Company, 1996.

[11] Software Productivity Consortium Services Corporation. *Reuse Adoption Guidebook*, Version 02.00.05, November 1993.

[12] Software Technology for Adaptable Reliable Systems. *Organization Domain Modeling (ODM) Guidebook*, Version 2.0. Unisys STARS Technical Report STARS-VC-A025/001/00, Reston VA, June 1996.

[13] Tracz, W. and Coglianese, L. Domain-Specific Software Architecture Engineering Process Guidelines, *Technical Report ADAGE-IBM-92-02*, Loral Federal Systems, 1992.