Fraunhofer Institut Experimentelles Software Engineering

Introducing the PuLSE™ Approach to an Embedded System Population at Testo AG

Authors:
Klaus Schmid
Isabel John
Ronny Kolb
Gerald Meier

Submitted for Publication to
SPLC 2004

IESE-Report No. 015.04/E
Version 1.0
February 2004

A publication by Fraunhofer IESE

Abstract

Over the last years product line engineering became a major theme in software engineering research and increasingly becomes a central topic of software engineering practice in the embedded domain.

Migrating towards a product line approach is not an easy feat. It is even less so, if this is done under tight technology constraints in an embedded environment. It becomes even more difficult, if the transition directly aims at integrating two product families into a single product population. In this paper, we discuss our experiences where we successfully dealt with these difficulties and achieved a successful product line transition. In our paper we strongly emphasize the role of technology transfer as many facet's of product line know-how had to be transferred to guarantee. From the experiences of this project many lessons learned, which can be transferred to different environments, can be deduced.

Keywords: PuLSE

Table of Contents

1 Introduction

Product line engineering focuses on the integrated development of a set of products instead of a single product. This shift in focus enables product line engineering to produce product suites much more cost-effectively than using the traditional approach.

While product line development is becoming more and more a common technology, there are still not many detailed accounts of product line transition. In particular, this is the case for scientifically monitored transitions that have been made with externally supplying the necessary technology. In this paper we will describe such a transition in a systematic manner. This transition has been performed at Testo AG using the PuLSE™ approach [2].

1.1 PuLSE™[1] Approach

The PuLSE™ approach [2] is a systematic approach to product line development. It consists of technical components addressing the different phases of product line development. These components are:

- **BC (Baselining and Customization)** – This addresses the tailoring of the generic PuLSE™ approach to the specific needs of an organization.
- **Eco (Economics)** – This component aims at scoping the product line development (i.e., identifying the products and required technical components for building the product line)
- **CDA (Customizable Domain Analysis)** – This is the flexible approach to domain analysis that can be applied with arbitrary notations and is customizable to different contexts.
- **DSSA (Domain-Specific Software Architectures)** – This is a scenario-based, incremental approach to deriving a product line architecture.
- **I (Implementation)** – This addresses the implementation of generic software components.
- **EM (Evolution & Management)** – This addresses the continuous evolution of the product line, including the configuration management processes.

[1] PuLSE stands for Product Line Software Engineering and is a registered trademark of Fraunhofer IESE.

In this project, the PuLSE-Eco approach was also used as a means to understand whether a product line introduction would be worthwhile and to derive a suitable introduction plan.

1.2 Testo AG

Testo AG is a dynamically growing enterprise, focusing on professional measurement technology Founded in 1957, it is meanwhile one of the leading suppliers of portable electronic measurement instruments. The company focuses in particular on measurement instruments for the industry and emission business. Measuring parameters like temperature, pressure, humidity and gas is the business of Testo AG.

The company currently has 23 subsidiaries and about 60 trading organizations on all five continents. Currently, a total of 1100 employees work for the company worldwide.

Figure 1 Typical Testo Products

A key contributor to the company's success was its stringent focus on portable measurement instruments. Continuously new products are developed, with a cycle time of about half a year to one-and-a-half year, depending on the overall complexity of the products. Usually, these products have been redeveloped completely (including hardware, while only performing opportunistic reuse). So far, two different product lines have been developed mostly independently in two different departments. As a result of this, the two departments developed two reference architectures for their systems. However, these reference architectures where not made explicit, so it was hardly possible to notice prior to the transition. Only in the context of some special cooperation projects did joint developments of the two departments occur. In 1999 a very ambitious project was started which focused on developing an integrated family of measurement instruments across the two departments. When the project delivered in 2001, it had become clear to everyone that there was considerable overlap between the functionality developed in both departments and among previous products, the current one, and potential future ones. This was a strong motivator for a systematic reuse approach.

2 Customizing the PuLSE™ Approach

Prior to the actual introduction of the PuLSE™ approach in an organization, a customization of the method to the specific conditions of the environment is made, using the PuLSE-BC component [11]. Usually, a rough customization is made based on the information which is initially available. This is later refined with additional information, e.g., from scoping.

In this section, we will discuss the main customizations that were performed for the context of Testo AG. We subdivide this description into two parts: adaptation of the PuLSE™ processes and notations and, second, other adaptation measures for the introduction approach, which go beyond the PuLSE™ approach.

2.1 Processes and Notations

The PuLSE™ method framework prescribes that the method is adapted to its context of application. We will now give an overview of the main customizations that were made. More detailed information will be given in the sections on the different phases.

2.1.1 PuLSE-Eco

PuLSE-Eco is usually the first component applied within a product line introduction. Thus, it typically requires little adaptation. In this case the application of PuLSE-Eco was rather unusual, as the approach was used as a pure product line feasibility (and payoff) study. This was actually contracted as an independent project. In addition, only little information on future product plans and existing architectures was available. This lead to several adaptations:

- An analysis of future products was performed together with product management.
- A re-documentation of an existing system architecture (only conceptual view) was added in order to elicit sufficient system knowledge to judge existing assets.
- Only little quantitative information on existing assets was available, thus the corresponding part of the PuLSE-Eco approach [9] was skipped.

- In case the evaluation would be positive, an introduction plan would be needed, thus additional precautions were taken to elicit the required information.

2.1.2 PuLSE-CDA

One of the key decisions for the introduction plan was that wherever successful techniques (e.g., inspections) where in place, they should remain in place for the product line introduction. One such successful technique was the usage of a use-case driven approach to requirements engineering. Thus, it was decided to stick to this for the introduction. Sticking with the textual use-case based documentation of requirements demanded a text-based way of handling variability. The approach we used is described in more detail in [13]. A key characteristic of our approach is that we separate the decision model from the actual product line or domain model.

2.1.3 PuLSE-DSSA

The application component of the PuLSE™ approach [3]. was in its application a bit non-typical, in the sense that we neither had a profound documentation of an existing architecture, nor sufficient (written) domain documentation. Thus, we started the effort with a re-documentation of the existing systems, instead of a domain analysis.

As previously no architecture documentation was written at Testo AG, it was clear from the start that most of the architecture work had to be done by Fraunhofer IESE. The notation that was used to document the architecture was the Unified Modeling Language (UML), with some stereotype-extensions to describe variability and architecture-related issues. The variability in the architecture was managed jointly with the variability in the domain descriptions, as described in the decision model. We will discuss the application of the PuLSE-DSSA approach in more detail in Section 3.3.

2.1.4 PuLSE-I

Mostly Testo AG implemented the various components. One reference component, however, was implemented by Fraunhofer IESE. As variabilities would happen on many different levels of granularity and had to be managed with many different development environments, it become clear very early that they had to be implemented mainly using conditional compilation as the base language was standard C. We will discuss the implementation approach in more detail in Section 3.4.

2.1.5 PuLSE-EM

While PuLSE-EM covers all areas of product line management and evolution, we had to restrict ourselves during this introduction, mainly for two reasons: approaches that were working very well were already in place and usually only two parallel product groups are under development (one per department). Thus, the existing approaches could still be used.

Thus, we focused on the improvement of the configuration management approach. We did so by providing a reference frame for the configuration management repository. This relied mostly on the code view, which was derived during architecting and extended it by a sub-division into a framework part (for all products) and a product part (one per product). By adequately applying the configuration management system, this allowed to provide to each developer a view where he could access his current project with making explicit application-specific and generic parts.

2.2 Other Measures

Besides the pure adaptation of the PuLSE™ approach, additional measures must be taken to support a product line introduction. A lot of this has to do with people management [10].

In the specific case of Testo AG, we also had to deal with the fact that the introduction of the product line technology would lead to a much stronger integration of the two departments. Thus, we decided to start the whole effort with a training program: a one-day seminar that should be taken by all members of the two departments. This seminar happened in two iterations, where the participants of each department where mixed from the two departments. During this seminar, we discussed both the fundamentals of product line development, as well as the key adaptations that were foreseen for Testo AG.

3 Applying the PuLSE™ Approach

While the previous section discussed our customization of the PuLSE™ approach, we will now discuss the application of the PuLSE™ approach to the product line stituation.

3.1 Planning a Product Line Introduction

In order to better understand how its initial attempts towards software reuse, which were based on libraries, could be improved, Testo AG addressed the Fraunhofer IESE in 2001. This lead to a first analysis based on the scoping technology PuLSE-Eco [9]. The analysis addressed:

- Existing software and technology
- Future product development plans
- Current reusable assets
- Organizational context
- Business situation

In our experience companies differ widely in terms of these aspects, and actually with regard to the accessibility of information on these topics. In the specific case of Testo AG, we found in particular the soft aspects (organizational and business) highly favorable to product line development. In this case, while two departments existed, they were willing to cooperate and to share assets. From a business point of view, the current business was highly profitable, it was thus possible to spend the required effort on up-front investment in infrastructure building; moreover, top-management was willing to support this step.

The technical side was less favorable in the sense that existing documentation was rather poor, architectural documentation was not existing. It was thus quite hard to ensure that a sufficient degree of commonality regarding the technical basis would be available. . A first analysis demonstrated what should become a major theme at least during the early steps of the product line introduction: often very similar solutions are used in both departments; however, they are named differently, making a common understanding very difficult.

The product situation was also rather positive: due to the strong focus on two market segments all future products would share major commonalities. Also, despite the functional differences among the products from the two departments, they still shared a lot of functionality, creating a sound basis for a product population.

As a result of the PuLSE-Eco analysis we found a high economic potential and a sound chance for a successful product line transition. The PuLSE-Eco analysis provided a decomposition of the product line functionality in terms of technical sub-domains. The benefit and risk level is also identified relative to the individual sub-domains. This enabled us to derive an incremental transition plan, where first an overall architecture framework would be developed and then the individual sub-domains would be transitioned one by one, in the order of their benefits and potential risks (high benefit, low risk first). In total we derived the following list of measures as the basis for our incremental product line transition:

- Initial employee training regarding product line technology
- Detailed analysis of the current architecture (also as a basis to gain the necessary domain understanding)
- Definition of a reference architecture, able to support the future products
- Development of a concept for configuration management, which supports consistent version management for all product variants

After this set-up phase for each of the different sub-domains an incremental transition ought to be performed. The following steps addressed this need (sub-domains with high benefit and low risk first):

- Perform a detailed domain analysis, in order to understand all commonalities and all (most) of the future variabilities along with their likelihood
- Refine the architecture definition for the corresponding functional area(s)
- Perform a generic implementation for the corresponding sub-domain
- Test the resulting components and reuse them in further product development

We will now discuss in a phase-wise manner how we incrementally performed the product line transition. In particular, we will discuss in Section 5 our experiences with performing a product line transition in a pure consulting mode.

3.2 Identifying and Managing Variabilities

The purpose of PuLSE-CDA is to develop a domain or product line model that captures the requirements of a product line and to enable the instantiation of this product line model for the product line members. In order to support evolution, these requirements do not only have to be identified but must also be managed adequately.

We introduced the product line approach incrementally, which means that we analyzed the sub-domains of the product line step by step, not all domain models had to be done for one release. The prioritization of the domains, which gave us an ordering for analyzing the sub-domains was given by the incremental transition plan produced during scoping (c.f. section 3.1).

As there was only little documentation available on some of the domains and some of the domains are technically very sophisticated we decided to capture the requirements interactively in the form of workshops with the domain experts from Testo and the IESE support team. The domain models were not too large, as we could split the product line into manageable sub-domains. Thus, we decided not to invest in special mechanisms or tools to capture and manage the variability, like a database supported decision model, but to use the Microsoft Word™ and Excel™ for capturing the requirements and the decision model.

In order to be able to model and manage variability, the existing mechanisms for writing textual requirements had to be extended into a product line modeling approach. Following our approach for variability management [13], only the mapping of the variability types (alternative, optional, multiple selection, etc.) onto the target representation, which was in this case text, had to be adapted.

We decided to use textual constructs framed with "<<" ">>", as these are text fragments which did so far never occur in this domain. Thus, we used the following notation to capture a multiple alternative variability:

```
<<mult    decision-variable / value-1 / text1
                          / value-2 / text2
                                    .....>>
```

Optional and alternative variability and values were expressed in a similar way.

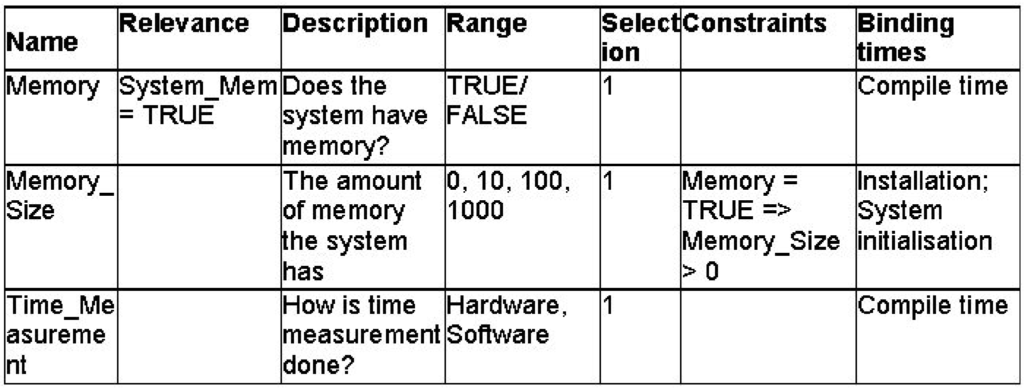
Additionally, a decision model was built that contains all decision variables used in the textual requirements, their ranges and constraints. Using this description of a decision variable, we can define a decision model simply as a set of decision variable definitions (see Figure 2).

The requirements and decision variables were captured interactively in workshops of one to two days duration, the existing requirements specifications in use case form were used as a basis, the additions and changes in these documents as well as the required variability were jointly integrated by the product line engineers and the domain experts. Additionally, general requirements like non-functional requirements for the whole sub-domain, project issues and open issues were recorded.

The product line model consists of a document for each of the sub-domains and the decision model. The requirements for a product can be identified by instantiating the sub-domain documents with specific values for the decision variables.

So far, we identified about 50 decision variables during modeling and introduced about 100 variation points into the documentation. The decision model was also used and extended during architecture development and implementation. Thus, the decision model evolved throughout the whole domain engineering process. The final domain models went through inspection and were released by the development team.

Products are instantiated by giving values to the decision variables in the decision model. By considering the restrictions and constraints in the decision model a correct and complete product model can be built. We refrained from deriving instantiated documents, as the developers accepted the generic product line documents and an instantiation would just have led to the duplication of documentation.



Name	Relevance	Description	Range	Selection	Constraints	Binding times
Memory	System_Mem = TRUE	Does the system have memory?	TRUE/ FALSE	1		Compile time
Memory_ Size		The amount of memory the system has	0, 10, 100, 1000	1	Memory = TRUE => Memory_Size > 0	Installation; System initialisation
Time_Measurement		How is time measurement done?	Hardware, Software	1		Compile time

Figure 2 Example of the decision model

3.3 Inventing a New Product Line Architecture

One of the key artifacts during product line development is the so-called product line or reference architecture. This architecture supports current as well as future products in a domain by defining common and variable components for the members of a product family. During the transition to product line development at Testo AG, a reference architecture had to be developed that provides a basis for the development of the various planned products while taking the special requirements imposed by the underlying hardware and technology into account.

As a basis for the development of a common reference architecture for the product line of flue gas and climate measurement devices, the architectures of existing products rather than domain documentation have been used. However, as no sufficient documentation of the existing software architectures was available, we started with a re-documentation of the existing systems. The re-documentation was done manually by means of a two-day workshop at the Testo site. As a basis for the re-documentation, a summary of the source code files, the file-system structure, and a short description of the purpose and provided functionality of each file has been used..

The architecture resulting from the re-documentation and the newly developed product line architecture were documented following the view model described by Hofmeister et al. [5]. Instead of the execution view defined by Hofmeister et al., however, we used a so-called behavioral view. Further, we extended each of the four views with respect to product lines and variability. As mentioned in Section 2.1.3, the architectural views were documented using an extended version of the Unified Modeling Language (UML). In total, we used the following four views for documenting architectures:

- **Conceptual View.** Depicts a system from an application domain viewpoint that is independent of solution aspects, like software or hardware techniques. This view captures the application domain by mapping the functionality of the system to conceptual components and depicting the relationships between them.
- **Module View.** Describes the static structure of a system in terms of layers, subsystems, and modules, the interfaces provided by them, and the relationships among the various elements.
- **Behavioral View.** Illustrates using sequence and/or collaboration diagrams how the architectural elements interact in order to fulfill a certain usage scenario.
- **Code View.** Shows how the elements from the module view are mapped to entities of the development environment such as code files, directories, libraries, etc.

During the re-documentation, first the code view was created based on information about the source files and their structuring. As a next step, the module view was defined. In retrospective, this approach was very effective. Once the module view had been captured to a certain level of detail, typical usage scenarios were defined and for each of them a behavioral view was created. This was used to check the module view for consistency and completeness. Finally, the conceptual view, which had been created during the planning of the product line introduction, was checked for completeness and soundness. It was also refined using the results obtained during the creation of the module and behavioral views.

The re-documentation revealed inconsistency and unintentional differences among the two product development departments. In addition, it helped to get a better understanding of the products and their architecture and provided a good starting point for the design of a new, common architecture.

The development of the new reference architecture was performed in joint workshops of Fraunhofer IESE and Testo AG. Contrary to traditional product line approaches, the design of the product line architecture was started even though no complete domain analysis had been performed beforehand. This was possible since already enough information from the pilot study and the re-documentation was available and a refinement of the architecture would be done at a later stage.

As a basis for the first workshop, two sketches of design alternatives were prepared. Both sketches had been developed based on an evaluation of the architecture resulting of the re-documentation activity and taking into account the product line requirements. These alternatives provided a basis for the development of the reference architecture. However, still additional design work had to be done. Next, the work focused on the creation of the module and behavior views. For the latter one, scenarios for the most important usages of a system were identified and defined. The behavioral view was created to check the consistency of the architecture and to refine the architecture.

In order to fulfill the specified functional and non-functional requirements for the product line and to address the required variations the following design decisions were made for the architecture:

- **Layered Architecture.** The system was organized in terms of several distinct layers, each encapsulating a certain subset of the overall system's functionality and placed on top of each other. By default, layers can only access the layer directly below it. For performance reasons, however, we had to relax this strict layering somewhat.
- **Publish-/Subscribe.** As the main communication and control mechanism among modules, the publish-/subscribe pattern was applied. Modules can register for an event-based notification on data changes or other specific conditions.
- **Repository.** Data exchange among modules is done using an active data repository. Modules making use of data in the repository can register via a publish/subscribe mechanism to be notified of data changes.

- **Model-View-Controller.** The user interface was structured according to the well-known model-view-controller pattern. In our architecture, the data repositories represent the model. The controller communicates with the repository and the views via the event mechanism and is independent of the actual input hardware (e.g. keyboard, touch screen, etc.) used for a particular product.

Figure 3 shows an example of a behavioral view for the developed architecture. This view depicts the scenario "Printing measurement data". Note that the figure shows only a simplified version of the view.

The resulting product line architecture was evaluated with respect to quality requirements and its suitability as a basis for the development of the envisioned products of the software product line. In particular, the architecture was evaluated with respect to maintainability (e.g. changing existing functionality, testability), extensibility (e.g. adding new functions, adding support for new input and output devices), reusability, performance (i.e. time behavior, resource usage), and reliability.

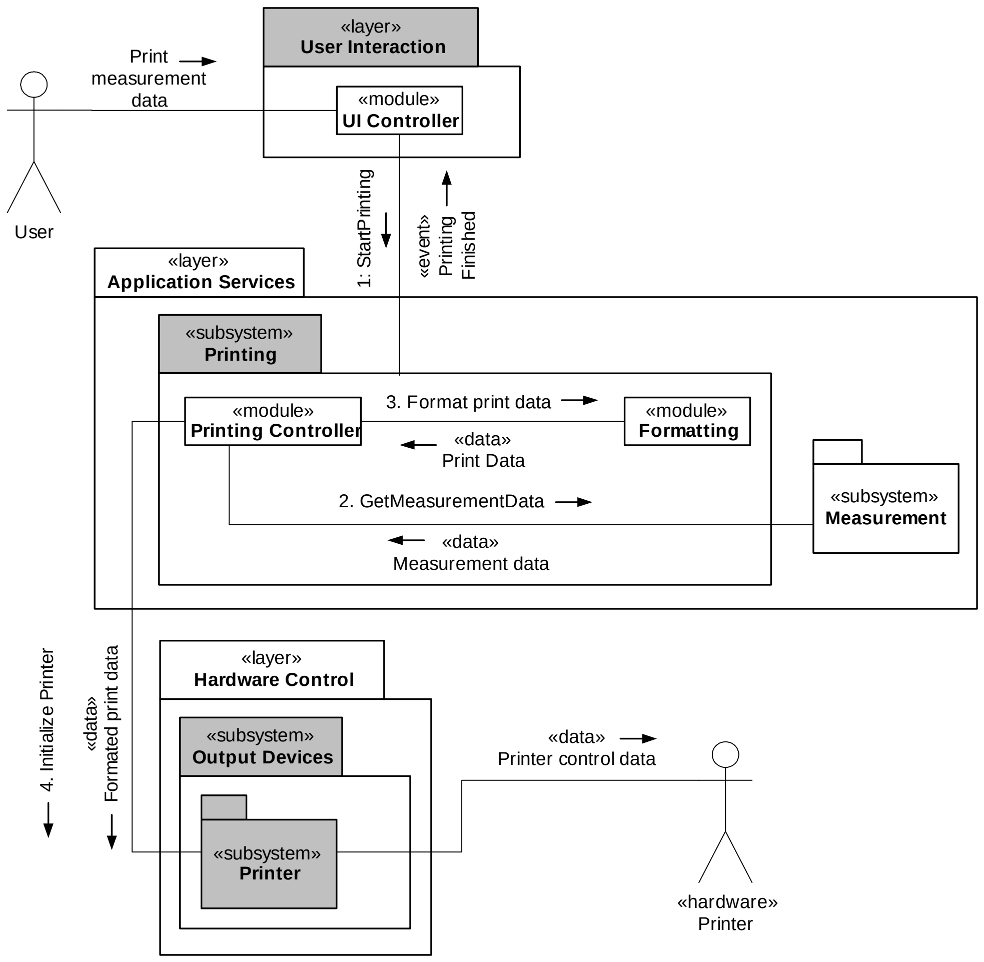


Figure 3. Example of a behavioral view

The Architecture Tradeoff Analysis Method (ATAM) [4] was used for the evaluation. Rather than using the standard out-of-the-box method, however, we used a slightly modified version that was tailored to the specific project context. Contrary to ATAM, the evlauation scenarios were prioritized regarding their importance for project success as well as the influence of the scenario on the architecture. As a result of the architecture evaluation, identified risks were documented and an overall estimation of the architecture and its quality was provided in a report. The architecture evaluation showed that the developed product line architecture provides a sound basis for the development of the product line and its members. Nevertheless, there was the general judgment that there might occur problems during implementation that cannot be foreseen and depend on how the architecture and its components will be realized using the specific programming language and implementation technologies.

As mentioned earlier, architecture development did start without a formal, prior domain analysis. Thus, some information required for defining interfaces and the behavior of architectural elements was missing. Therefore, the evaluated architecture was not detailed enough to provide a sound basis for implementation. Thus, as a last step of the architecture development activity, the architecture was refined as soon as domain models were available as the result of a domain analysis performed for a sub-domain.

3.4 Designing and Implementing Generic Assets

Once the architecture development was finished, developers of Testo AG started to implement the various components defined in the architecture. For one generic component of the architecture, however, Fraunhofer IESE was contracted for the design and implementation according to the defined product line architecture. The goal of this outsourcing was not only to develop a particular component, but also to show how a software component can be generically designed and implemented.

The component selected by Testo AG for this was the printing component. This is mainly responsible for controlling the different printers that can be connected to a measurement device as well as for providing users functions for printing various measurement data and information about the device including its status and settings.. The generic printing component must provide printer drivers for three different printers, which are planned for use with the products under development. Besides the different printers, variations to be covered by the component occur in terms of the provided functions for printing data. In a flue gas product, for example, additional information regarding the measurement conditions must be printed.

Before starting the implementation of the component, we made a detailed design for the component. As a starting point we used the product line architecture and the domain analysis document for the printing subsystem. The design of the component was done using the KobrA approach [1]. According to KobrA, the design of a component consists of two basic parts: the specification, which describes the externally visible properties of the component, and the realization, which describes how these properties are realized. The specification consists of three different models which were modeled using the Unified Modeling Language (UML): the structural model, which describes the information needed to use a component in terms of class and object diagrams, the behavioral model, which describes the externally visible states of the component in terms of a state chart diagram, and the functional model which describes the effects of the component's operations in terms of textual pre and post conditions. In case of the printing component, the specification structural model is equivalent to the architecture.

The realization also consists of three models: the structural model which is a refinement of the specification structural model describes all types and structural information related to how the component works, the interaction model which describes how instances of the component interact with other component instances in order to realize the component's operations, and the activity model which describes the algorithms used to implement the operations. The models of both specification and realization potentially contain variability. Therefore, we also created a decision model, which is actually a refinement of the decision model created and maintained as part of the domain analysis and architecture creation activities. Once the design of the generic printing component was finished, the implementation work started. The implementation language used by Testo is standard C. In order to implement the variabilities of the printing component, we had to select one or more suitable variability mechanisms. A variability mechanism [6] is a way of implementing varying characteristics of a component at the implementation level. As there is actually only very limited variability for the printing component at the implementation level, we decided to mainly use conditional compilation for realizing optional and alternative variabilities. This variability mechanism was used for variabilities with a binding time of build time. Binding time refers to the point in time when decisions are bound for the variations, after which the behavior of the final software product is fully specified [8]. In the project context, we had to deal with the following binding times: static code instantiation time, production time, start-up time, and runtime. Therefore, we applied also directory naming [8], dynamic settings, and inclusion polymorphism using function pointers for realizing variability at the implementation level. The latter was used to implement the support for multiple printers and the switching of printers at runtime.

The implementation of the generic printing component also included its validation by means of code inspections and testing. Developers from Testo AG inspected the code of the component and Fraunhofer IESE incorporated the feedback from the inspection into the implementation. As the component is used in two different products only and has very limited variability, testing was done just as in traditional single software development. We created instances of the component for the particular product and performed unit and integration tests.

4 Experiences in Consulting a Product Line Effort

In this section, we will discuss our main experiences and lessons learned from this effort with a focus on lessons that can be easily transferred to other contexts. We will structure our discussion mainly based on the phases of the product line effort: prestudy and planning, domain analysis, architecting, implementation and the overall technology transfer.

4.1 Prestudy and Planning

Prior to the actual introduction of the product line we performed a prestudy that aimed at analyzing the potential cost savings and development optimizations that could be gained from a product line effort. As part of this effort the PuLSE-Eco approach was applied with a strong focus on the Product Line Mapping and the assessment component [9]. As a result of this a domain characterization of the technical subdomains and a product line introduction plan was derived.

As a lot of experience in the product domain existed, we decided to identify the main technical domains by a simplified architecture workshop. This had the dual purpose of acquainting the stakeholders with architectural concepts. This proved in our context a successful replacement of a more detailed domain analysis, however, this can only be done, if a lot of experience exists with the domain.

We found that we could do this analysis easily with a single person from our side and that it provided us with a lot of base information that enabled us to systematically derive an introduction plan. This introduction plan could not only be much better estimated in terms of effort for both organizations, we could also identify certain risks and benefits that we could not have identified without the underlying systematic evaluation approach. Thus it directly informed our introduction plan and should be used also in other introduction efforts. It also had the additional benefit of acquainting both partners and thus building the necessary foundation of mutual trust.

4.2 Domain Analysis

The domain analysis focused on developing individual domain specifications for the different technical domains. This was only done after we derived a basic architecture of the system. This proved very successful in our context, however, we expect this approach (architecting without an overall domain analysis, but using domain analysis to refine the basic functionality) to work only in settings where the overall domain is rather mature and the products are not too innovative.

For introducing novel functionality in the domain model, we used the approach of discussing first whether the functionality would be really relevant and second whether the relevant functionality would be required in the first two to three systems. This way we identified a lot of functionality, which might be required in further products (thus being aware to this in the further refinement of the architecture) while explicitly excluding it from the components that should be implemented initially. This two-step characterization went very well and provided an additional evolution dimension, which was captured in the decision model. When deriving a basic notation for the domain analysis, we built on the existing use case notation. We found it very useful to rely on a notation already in use. Using the approach we developed [13], such an augmentation can always be performed and can be expected to considerably raise the level of acceptance by the developers. We found that the documents in their generic form (i.e., without instantiation) were widely accepted by the developers after some time and a learning phase. However, the training and the strong integration of the domain experts in the document development were key in the acceptance of the notation.

4.3 Architecture Development

In our effort we based the architecture development on the analysis of the existing architecture, instead of basing it on an overall domain analysis as the prototypical approach would be. Here, this was a viable approach, as the previous recovery of the architecture lead to a good understanding of the domain and the future systems would not include fundamentally different functionally.

We developed the final architecture in close cooperation, where Fraunhofer IESE contributed through overall product line and architecture knowledge and Testo focused on the specific requirements, especially those for potential future systems. This strong cooperation together with a jointly performed ATAM-assessment lead to a feeling by Testo employees that the resulting architecture was really their architecture and not just an architecture imposed on them. It also lead to a better understanding of the architecture and its implications on behalf of the developers, although we found the results of the ATAM-assessment were rather hard to judge from their point of view. From our experience this kind of pairing and cooperative architecture development can be recommended in general. We also found in architecture development that it is better to have a list with design alternatives together with rationales as well as expected benefits and drawbacks rather than two elaborated architecture alternatives.

In our experience we found scenarios to be particularly helpful in achieving an understanding of the architecture and communicating the architecture, identifying inconsistencies and defining the semantics of modules. Due to the embedded context we found it pretty common that we had to discuss implementation possibilities and the effect of design choices on non-functional aspects like performance. While digging into the implementation aspects during architecting is not a common thing to do, we found it key for a successful embedded systems architecture.

4.4 Design and Implementation

We used the KobrA notation and approach as the basis for design. While this provides a systematic technology, we found it sometimes too comprehensive for our purposes and had difficulties fitting it together with the architecture and the decision model. We also found it useful to start implementation already while the design was only coarsely defined. The parallel activities lead to a faster convergence of the results.

In implementation, many different technologies can be used for implementing variabilities (e.g., conditional compilation, inclusion polymorphism, etc.). Using the decision model with the different required binding times helped a lot in deciding on the most appropriate technique at any point. A careful selection of the optimal variability technique was particularly important in our embedded context, as the wrong decision could easily result in performance problems.

While often condemned, we found conditional compilation as a rather useful technique for our application. A precondition for its successful application is, however, that the number of variabilities is not too high and the variabilities are not too fine-grained. It was also very helpful to use as condition variables the names given by the decision model (e.g., the decision PGM_MEASUREMENT is implemented using the expression #ifdef PGM_MEASUREMENT. As this name is also used in the other documents (e.g., domain models, architecture, etc.) we achieve traceability of variability basically for free.

Regarding the testing of the generic components we found that for a small number of instances, testing can be performed as with traditional systems/components. However, we expect that for a larger number of instances along with a large number of variabilities optimized test strategies will be necessary.

Additionally to introducing variation points, the printing component was implemented using a generative implementation approach. We identified about 40 different types of printing-blocks (e.g. header, measurement data, maintenance data etc), that can now be instantiated by combining calls to the block-functions in the concrete printer implementation. This concept was seen as a large improvement by the developers. The resulting implementation additionally contained about 20 variation points that can be differently combined for the three printers realized so far. By using preconditions, the code size and complexity of the resulting implementation stayed the same for the new implementation.

4.5 Product Line Technology Transfer

Introducing product line concepts from an external position is a very problematic endeavor. We started this effort with a systematic training of the development engineers at the customer site. This has to be regarded only as partially successful as it was just too early in the overall project. The developers focused very strongly on technical matters (implementation, configuration management), while issues like domain analysis and architecting, which are key to product line development did not (yet) interest them too much. In retrospect this understandable do to the early time.

However, we did also use the seminar to identify and address their needs and fears (e.g., fear of unemployment, due to overall effort reduction). In this regard the seminar was very important. As a result we would recommend to step-wise integrate (and train) the different engineers on the basis of their overall work involvement [10].

We used for domain analysis and architecting an approach that was highly interactive: we held meetings that usually involved two persons from Fraunhofer IESE and 2-8 persons from Testo AG (typically 4). During these meetings we partially developed the domain analysis documents and architectural documentation. During the meetings typically one person from Fraunhofer IESE focused on the role of the scribe, while the other person focused on the role of a moderator. We also used two projectors: one typically showing the decision model (or other key information), the other showing the current artifact under development. For introducing product line technologies we recommend this interactive approach, as here it is possible to incrementally introduce concepts and to directly react on misunderstandings.

Also for design and implementation, we had to introduce a process change. The developers had to use the architecture and the predefined interfaces in order to keep the product line healthy. So, a much stronger development process awareness and awareness of the architecture (as opposed to ad-hoc implementations) had to be fostered.

5 Conclusions

Product line development is an up-and-coming technology with a large potential to reduce cost and effort and improve quality in product development. This potential is even increased by addressing a complete product population. In this paper we discussed our experiences with a product line transition in an embedded environment.

The basis for this transition was provided by a systematic pre-study and it was actually conducted using the PuLSE™ approach. While this project was unusual in its approach (e.g., applying architecting without systematic domain analysis, based on reengineering), we discussed the key strengths and weaknesses of our approach and described under which circumstances a similar approach will probably be successful.

As a key contribution our paper provides a large number of lessons learned, which can be transferred (under the described circumstances) to other environments. In this way we contribute to the increasing body of knowledge on product line introduction.

References

[1] C. Atkinson, J. Bayer, C. Bunse, E.Kamsties, O. Laitenberger, R. Laqua, D. Muthig, B. Paech, J. Wüst, and J. Zettel. Component-based Product Line Engineering with UML. Component Software Series. Addison-Wesley, 2001.

[2] J. Bayer, O. Flege, P. Knauber, R. Laqua, D. Muthig, K. Schmid, T. Widen, and J.-M. DeBaud. *PuLSE: A Methodology to Develop Software Product Lines.* Proceedings of the Fifth ACM SIGSOFT Symposium on Software Reusability (SSR'99), pp. 122–131, 1999.

[3] J. Bayer, O. Flege, and C. Gacek. Creating Product Line Architectures. Third International Workshop on Software Architectures for Product Families, Frank van der Linden (ed.), Springer LNCS 1951, pp. 210–216, 2000.

[4] P. Clements, R. Kazman, M. Klein. *Evaluating Software Architectures: Methods and Case Studies*. Addison-Wesley, 2002.

[5] C. Hofmeister, R. Nord, D. Soni. *Applied Software Architecture.* Addison-Wesley, 1999.

[6] I. Jacobson, M. Griss, P. Jonsson. *Software Reuse: Architecture, Process and Organization for Business Success*. Addison-Wesley, 1997.

[7] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, November 1990

[8] C. W. Krueger. *Towards a Taxonomy for Software Product Lines*. Proceedings of the Fifth Workshop on Product Family Engineering (PFE-5), Sienna, Italy, November 2003.

[9] K. Schmid. *A Comprehensive Product Line Scoping Approach and Its Validation.* In Proceedings of the 24th International Conference on Software Engineering (ICSE'02), pp. 593–603, May 2002.

[10] K. Schmid. *People Management in Institutionalizing Product Lines.* In Proceedings of Netobject.days 2003 (NODe'03), pp. 175–189, September 2003.

[11] K. Schmid and T. Widen. *Customizing the PuLSE Product Line Approach to the Demands of an Organization*. Software Process Technology, 7th European Workshop, (EWSPT'2000), Reidar Conradi (Ed.), pp. 221-238, LNCS 1780, Springer, 2000.

[12] K Schmid. *An Assessment Approach To Analyzing Benefits and Risks of Product Lines.* The 25th Annual International Computer Software and Applications Conference (Compsac'01), pp. 525-530, 2001.

[13] K. Schmid and I. John. A Customizable Approach To Full-Life Cycle Variability Management, Journal of Science of Computer Programming, 2004, to appear.

Document Information

Title:	Introducing the PuLSE Approach to an Embedded System Population at Testo AG
Date:	February 2004
Report:	IESE-015.04/E
Status:	Final
Distribution:	Public