Accepted for Panel
Discussion at SEKE'99,
Kaiserslautern

How to do Software Product Lines Right

Peter Knauber
Fraunhofer Institute for
Experimental Software Engineering (IESE)
Sauerwiesen 6
D-67661 Kaiserslautern, Germany
+49 (0) 6301 707 242
knauber@iese.fhg.de

Abstract
Product lines pay in traditional industries. In this position paper, their success factors are analyzed to see if they can be transferred to software development. Then, the PuLSE method is presented, which covers the whole product line lifecycle, thus addressing all critical tasks involved in development, usage, and evolution of software product lines. Experience from PuLSE application in several industrial projects is presented.

1 Introduction

Product lines are a very successful approach in traditional industries. Up to now, some successful examples have been published of product line (or domain engineering) concepts having been applied to software production [2]. But what is the basis for their success and can it be repeated systematically? After analysis of the main product line success factors in traditional industries in section 2, section 3 points out what software industry can learn from these successes. In section 4, PuLSE is described, the product line method developed at Fraunhofer IESE that covers the complete product line lifecycle. Some lessons learned from the application of PuLSE are highlighted in section 5.

2 Product Lines in Traditional Industries

Product lines have been established in various industries for a long time. Successful examples are clothing industry and especially the car manufacturing industry with many small and mid-sized companies supplying specific car parts. The reason for introducing product lines in these traditional industries was the availability of manufacturing machines that could be used to automate routine tasks. These machines typically led to faster production (once the product line was established), cheaper products (due to rationalization effects), and better product quality (because machines are more reliable than humans as far as routine tasks are concerned).

On the other hand, machines are less flexible than humans. That fact very soon led to the development of standard product platforms as basis for a variety of similar but different products. Using standard product platforms, it is possible to build the common core of a product and then finish it by integrating components sharing a standardized interface with the core (and sometimes with each other). That sort of production using modular components based on a common core has several advantages: Components can be produced by different companies and later be integrated with the common core of a product line. This leads to competition among the component suppliers and thus, decreases the component prices, as well as increasing the variety of component alternatives. For example, in Germany, there exist about 1500 companies developing and producing different parts for the automobile industry. As another major advantage, the needs of different customer groups can be addressed using alternative components to produce a diversity of products that is still based on the same core platform. Sony Corporation, for example, developed a variety of 232 different walkman variants based on one core product.

Summarized, the three main advantages of product lines in non-software industries are

1. effective, that is, fast and cheap production,
2. the definition of standards (i.e., the interfaces of the product platform) that enable the development and use of (third-party) components, and
3. the chance to derive a variety of products based on the same product line infrastructure to address different customer groups.

Can these product line advantages be transferred to software production? With respect to product lines, what can software industry learn from traditional disciplines? Which benefits can we expect?

3 Product Lines in Software Industry

The production of software is different from the production in other industries in that the *pure* production

process is just copying existing software, that is, application binaries. The challenge for the software industry is the development of the *master copy*. (This is why the term *software production* usually refers to the *development* of software.) What does that mean with respect to the advantages of product lines in traditional industries described in the previous section?

1. As described in the previous paragraph, the pure software *production* is no problem. The problem is to scope and define a good product platform.
2. Product line concepts in software development are applied to determine a standard product platform called domain-specific software architecture (DSSA) or reference architecture. Once such an architecture and its interfaces have been defined, components can be developed (also by third-party companies) to modify existing or add new functionality.
3. Software product lines allow to address the needs of different customer groups by defining a standard platform on which individual product variants can be based.

In software production, especially, a reference architecture offers another advantage:

4. Based on a reference architecture, software variants with a common core can be developed and maintained more effectively, that is, faster and cheaper than a set of single (unrelated) products. This results from the fact that common parts are developed and maintained once and for all [5]. Over time it can also be expected that product quality increases, since more and more (tested and thus reliable) components can be reused.

To summarize the second and third item, the reference architecture as a platform for a set of similar products is the major motivation to invest in software product lines. From the fourth item it becomes clear that product lines are worth the investment. The first item points out that the challenge in software development is to define, develop, and evolve an appropriate reference architecture.

At Fraunhofer IESE, we have developed a methodology called PuLSE (Product Line Software Engineering) that meets exactly that challenge. PuLSE covers the whole product line lifecycle, centered around the concept of a reference architecture. The method is described very briefly in the following section.

4 The PuLSE™ Methodology

The PuLSE method [1] consists of three main elements: the deployment phases, the technical components, and the support components (see Figure 1).

- The *deployment phases* are logical stages a product line goes through: Initialization of PuLSE, construction, usage, and evolution of the product line infrastructure. The phases describe the activities performed to set up, use, and evolve the product line.
- The *technical components* provide the technical know-how needed to operationalize the product line development. They are used throughout the deployment phases to cover the whole product lifecycle, but can also be applied separately to address specific product line-related aspects.
- The *support components* package guidelines on how to introduce, deploy, and evolve product lines in specific contexts. They are used by the other PuLSE elements.

In the given context of deciding when it is worth to introduce a software product line, only the technical components are of interest. They are described roughly in section 4.1 to section 4.6.

4.1 Baselining and Customization: PuLSE-BC

PuLSE-BC's role is to instantiate PuLSE and to customize it to a specific enterprise context. In a baselining phase, factors characterizing specific product line situations are elicited, then PuLSE is customized accordingly. Customization comprises the derivation of a complete process, including the definition of workproducts used, their relations, and their representations depending on the elicited factors.

4.2 Scoping the Domain: PuLSE-Eco

PuLSE-Eco is the first of three components used in the construction phase of the product line infrastructure. Within PuLSE, PuLSE-Eco is used to determine an economically viable scope for the product line. Moreover, it provides valuable information for distinguishing

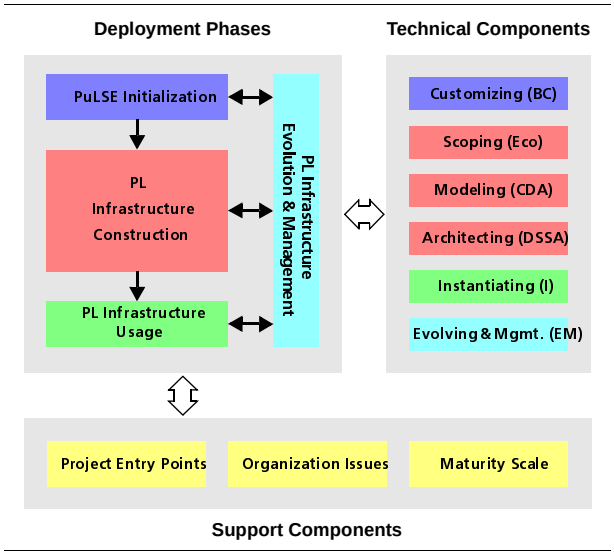

Figure 1. PuLSE Overview

domain-related from application-specific components during the development of a product line reference architecture (see PuLSE-DSSA in Section 4.4).

First, anticipated product line members are determined and their characteristics are mapped out. Then the benefit of having each of the characteristics as part of the reusable infrastructure is evaluated with regard to the specific business objectives that motivated the move towards a product line approach. This is done in a systematic manner to enable traceability of the results [3].

4.3 Modeling the Product Line: PuLSE-CDA

In PuLSE-CDA, information about the product line scoped by PuLSE-Eco is elicited and modeled using the workproducts and the process as defined by PuLSE-BC. The different, often domain-specific, workproducts (e.g., workflow diagrams, message sequence charts, data models) capture complementary views on the product line.

Populating the workproducts combines elicitation of information about single systems and consolidation of this knowledge into a generic product line model that captures variability. To derive the specification of a single product line member from that generic model, a domain decision model is created that contains a structured set of decisions corresponding to variabilities in the workproducts. To define a specific product, the decisions are resolved (see PuLSE-I in Section 4.5).

4.4 Developing a Reference Architecture: PuLSE-DSSA

PuLSE-DSSA supports the definition of a domain-specific software architecture, which covers current and future applications as described by the product line model produced by PuLSE-CDA.

Task scenarios drive the incremental development of the architecture. These scenarios are derived from the different generic workproducts produced by PuLSE-CDA; they represent major functional requirements and quality-related aspects. First, they are ranked according to their architectural significance. The most important ones are used to create an initial architecture. Then, using an iterative process, the others are applied one by one to refine or extend the current architecture in a way such that it supports the scenarios. Through the definition of this architecture the software components that will form the basis for reuse are identified. Information from PuLSE-Eco is used to differentiate between components implementing application-specific functionality and those implementing common, i.e., domain-related functionality. The latter are part of the reuse infrastructure, the former are planned for in the form of architectural alternatives and options.

During the creation of the reference architecture, implementation-specific decisions are collected that will have to be resolved during instantiation. These decisions and their possible resolutions are captured in the architecture decision model that extends the domain decision model. This architecture decision model supports the tailoring of the reference architecture to a specific situation, but it is also used to select among different implementations of a software component (interface), in those cases where more than one implementation is possible.

4.5 Instantiating Product Line Members: PuLSE-I

PuLSE-I aims at specifying, instantiating, and validating single product line members. This encompasses the instantiation of the product line model and the reference architecture using the decision models from PuLSE-CDA and DSSA, the creation and/or reuse of products that constitute the instance, and validation of the resulting product.

For the implementation, existing product line assets are reused, new assets that are within the scope of the product line but have not been designed and implemented yet are developed, and assets are created to fulfill specific requirements of the current instance only.

4.6 Evolving and Managing the Product Line: PuLSE-EM

PuLSE-EM monitors and controls the evolution of the product line infrastructure and coordinates the activities of the other PuLSE components over time. It gets various workproducts from the different components, consolidates them, and takes care of their evolution and maintenance. Configuration management is also one main task of this component.

5 Software Product Lines: How to do it right

Currently, we are applying (parts of) PuLSE in several industrial projects. Examples of domains we are currently addressing include merchandise information systems, civil engineering, desktop publishing, and stock exchange.

What we have seen so far meets our expectations:

- The definition of the “right” reference architecture is the critical issue in developing software product lines. Nevertheless, to achieve that goal, the determination of the appropriate scope and the availability of a good domain model are essential.
- For the small and medium-sized enterprises (SME), especially, that we are working with in the domains listed above, the definition of the right scope is critical. If it is defined wrong or too small, it will not fulfill its purpose, that is, we get an inappropriate domain model and thus, an unusable reference architecture. On the

other hand, SMEs, in particular, cannot afford the investment of the resources necessary to build a model for a complete domain, develop a reference architecture based on that model, and implement that architecture if the scope (i.e., the domain) is too big. The scope has to be based on the most important business objectives and take into account current and anticipated products only (i.e., not the episthemic domain definition) to serve the needs of the company.

- Having an appropriate domain model is prerequisite for deriving the reference architecture. That means, the notation(s) to be used for the model has to be chosen and/or defined carefully and the right parts of the model have to be generic, covering the concrete products but keeping the complete domain in mind.
- Parts of the reference architecture covering certain parts of the functionality must be traceable to the respective parts of the domain model. Since most domains are not stable but evolving, it must be possible to evolve the product line infrastructure as well, that is, the scope, the model, the architecture, and the implementation. For that purpose, traceability between these main product line assets is the key.
- Another task that has to be clearly understood is the instantiation of the product line infrastructure to define and implement single products. There has to be a well defined process (and tool support) to do a systematic instantiation of the domain model (in order to define the specific product) and the reference architecture (to identify the necessary parts of the implementation with their interfaces to product specifics).

6 Conclusion

After analyzing product line success factors in traditional industries, lessons learned for software development are pointed out. The PuLSE method, developed at Fraunhofer IESE, covers the whole product line lifecycle, thus addressing the tasks necessary to do software product lines right. Experience with PuLSE and some lessons learned conclude the position paper.

References

1. J. Bayer et al. *PuLSE: A Methodology to Develop Software Product Lines*, accepted by the Symposium on Software Reusability ‘99 (SSR’99)
2. L. Brownsword, P. Clements *Case Study in Successful Product Line Development*, Technical Report CMU/SEI-96-TR-016, Carnegie Mellon, 1996
3. J.-M. DeBaud, K. Schmid *A Systematic approach to Derive the Scope of Software Product Lines*, accepted by the International Conference on Software Engineering ‘99 (ICSE’99)
4. P. Knauber, D. Rombach, K. Schmid, F. Bernauer *A Comprehensive Approach to Software Product Line Development: PuLSE*, Position Paper submitted to IEEE Computer Society Special Issue and Book Readings on “Trends in Development of Reusable Software Components and Products”
5. D. C. Rhine, R. M. Sonnemann *Investment in reusable software. A study of software reuse investment success factors*, The Journal of Systems and Software, Volume 41, 1998