Bayer, J., Flege, O. & Gacek, C. (2000). Creating product line architectures. Lecture Notes in Computer Science, 1951, 210 - 216. doi: 10.1007/978-3-540-44542-5_23 <http://dx.doi.org/10.1007/978-3-540-44542-5_23>

Creating Product Line Architectures[1]

Joachim Bayer, Oliver Flege, and Cristina Gacek

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

Abstract. The creation and validation of product line software architectures are inherently more complex than those of software architectures for single systems. This paper compares a process for creating and evaluating a traditional, one-of-a-kind software architecture with one for a reference software architecture. The comparison is done in the context of PuLSE-DSSA, a customizable process that integrates both product line architecture creation and evaluation.

1 Introduction

Product line engineering is an approach to improve development efficiency for families of systems by facilitating large-scale reuse. It typically focuses on building a reuse infrastructure that can then be used to derive the product line members. A core asset of such a reuse infrastructure is a product line architecture (also known as reference architecture or domain-specific software architecture). Software architectures are a set of components, connectors, constraints imposed on the components, connectors, and their composition, and a supporting rationale. They are presentable in various ways — different views supporting different needs [4]. A product line architecture is a software architecture for supporting a complete product family, it reflects common parts as well as variabilities among the various products. Product line architectures define the essential parts of the reuse infrastructure and thus ensure that shared and instance-specific components fit together for all product line members.

In this position paper, we illustrate the specific aspects of architecting for product lines in the context of PuLSE-DSSA, the reference architecture development component of the PuLSE™ (Product Line Software Engineering)[2] method [2]. PuLSE is a method for enabling the conception and deployment of software product lines within a large variety of enterprise contexts. To achieve this, the different PuLSE components are customizable to different situations and contexts.

[1] This work has been partially funded by the ESAPS project (Eureka Σ! 2023 Programme, ITEA project 99005).

[2] PuLSE is a registered trademark of the Fraunhofer IESE

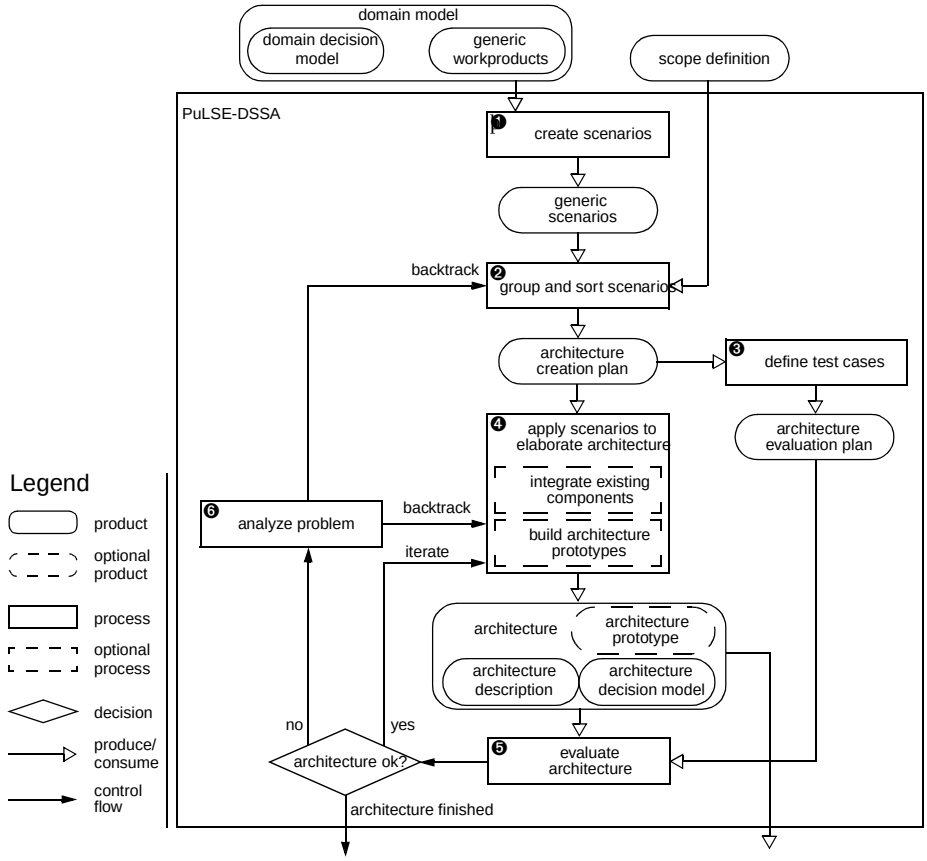

Figure 1 PuLSE-DSSA Process Overview

2 PuLSE-DSSA

PuLSE-DSSA is based on the generic architecture development process shown in Figure 1. This process is generic because it abstracts from the differences between one-of-a-kind and product line architecture development, and also from the customizations necessary to use this process in different application contexts. The only product line specific aspects shown in Figure 1 are the input and output products of PuLSE-DSSA. The input is a scope definition and a domain model, where the former defines the business case for the development of the product line and the latter describes commonalities and variations of applications within the product line. Output of PuLSE-DSSA is a product line architecture as defined in the introduction.

In the following subsections, we present for each process step a generic description and then point out the product line specific aspects, including customizations, that have to be taken into account when developing a reference software architecture.

2.1 Create Scenarios

The first step in architecture creation is to determine the most important requirements. These are captured in scenarios that describe critical use-cases (i.e., how the system is used to perform a specific task) as well as system level quality objectives (i.e., non-functional requirements) and constraints. This step is necessary for a generic process in order to decouple it from the various kinds of inputs (i.e., requirements specifications) that are available in different contexts.

Product Line Aspects. In the PuLSE product line development process, the input for this step would be a product line model consisting of generic workproducts (i.e., products describing requirements in terms of commonalities *and* variabilities) and a decision model.[3]

Conventional scenarios are described on an instance level, which makes it difficult to convey the variability information needed for product line requirements and to extract the information that applies to a particular instance. Therefore, it is beneficial to use generic scenarios that represent commonalities and variabilities like the product line model's generic workproducts, and are also instantiable via the domain decision model.

Managing traceability information is important to achieve effective maintenance and consistent change management. This is even more valid in a product line context, because a product line infrastructure is a strategic investment and will almost certainly be maintained for a long time. As a first step towards full traceability, the scenarios have to be linked to elements of the product line model.

2.2 Group and Sort Scenarios

This step yields the architecture creation plan that defines the iterations in which the architecture development is performed. The first iteration deals with the most important group of scenarios, the second one with the second most important group and so forth. The order in which scenarios are addressed is highly significant, because each iteration's design decisions impose constraints on the architecture that delimit its further evolution. Unfortunately, grouping and ordering of scenarios is also a highly subjective task, as it relies on a combination of domain knowledge (ability to judge the importance of a scenario) and system architecting experience (ability to anticipate how a scenario could affect the architecture).

Product Line Aspects. The judgement of a scenario's importance cannot be based on its expected impact on the architecture alone. A complementary factor is the overall importance a scenario has for the product line. Therefore, the information contained in the scenarios is insufficient and has to be augmented with the scope definition. Ideally, the scope definition is based on a process that seeks to estimate the economic value each distinct feature would have for the product line (e.g., PuLSE-Eco [3]). Assuming

---

[3] A *decision model* captures variability in a product line in terms of open decisions and possible resolutions. In a *decision model instance*, all decisions are resolved. As variabilities in generic workproducts refer to these decisions, a decision model instance defines a specific instance of each generic workproduct and thus specifies a particular product line member.

that economic value and importance correlate, some of the subjectivity of this process step can thus be reduced.

2.3 Define Test Cases

For each group of scenarios, test cases are defined that will be used to evaluate the architecture at the end of each iteration. It should be possible to conduct the tests automatically in order to facilitate regression testing, which is particularly important for an iterative development process. Specifying test cases *before* the actual development begins has a number of benefits, including a better understanding of the requirements and avoidance of creating tests that, due to a fixed perspective, merely support what has been developed.

It is important to note that evaluation needs have an impact on the choice of how the architecture should represented, because different notations allow for different kinds of (automatic) evaluations.

Product Line Aspects. In a product line context, instance- and family-specific test cases have to be distinguished. The former do not differ from those used in one-of-a-kind architecture development and are therefore less of a problem. The latter focus on specific kinds of system level quality objectives such as maintainability, understandability, and reusability. These qualities play a critical role for the appropriateness of a reference architecture. However, it is extremely difficult to assess the degree to which theses qualities are achieved by a given architecture. This problem is hardly addressed by any architecture evaluation method (e.g., SAAM[4] [5], ATAM[5] [6]) nor by any architecture description language [7]. Neither have – to our knowledge – any guidelines been published on how testing should be performed to ensure that a product line infrastructure will be adequate.

2.4 Apply Scenarios

The group of scenarios associated with the current iteration is used to create the initial architecture or to refine/extend an already existing, partial architecture. This step also includes the possible integration of existing components (legacy or COTS) as well as prototyping. The result is a (partial) architecture description and possibly a prototype. During the application of scenarios, it is important to capture design decisions and link them as well as architectural elements to scenarios to ensure traceability.

Product Line Aspects. During architecture development some variabilities might become apparent that are not driven by the problem domain, but rather by the solution

---

[4] SAAM assesses the impact of anticipated changes (new requirements) to predict modifiability. The idea behind product lines is to consider anticipated changes already during architecting. Therefore, we should expect that a SAAM scenario applied to a product line architecture is satisfied by a particular instance of that architecture without needing any noteworthy changes

[5] ATAM provides a framework for evaluating architectures, but does not provide the methods that are required to use the framework (e.g., once you have a method to describe and assess the understandability of an architecture, you could use ATAM to investigate the trade-offs between understandability and other quality attributes).

(e.g., two components addressing the same problem, yet implementing differing algorithms). In this case, the domain decision model is complemented by an architecture decision model. All open decisions in both of these models have to be resolved for instantiation.3w

The following issues address possible customizations of this step. First, an appropriate representation for the product line architecture has to be chosen, which is a major problem, because mainstream notations (e.g., UML) and tools do not support the description and instantiation of generic architectures. Furthermore, different parts of the architecture might require different representations as described by Perry [8].

Hand in hand with choosing a representation goes choosing (creating) an instantiation mechanism and process. This may even encompass the construction of the necessary instantiation infrastructure (tools, etc.). By defining how instantiation is performed, it is also determined how variability is actually *implemented*.

Finally, the extent to which prototyping is performed must be defined. As illustrated earlier, a theoretical groundwork for evaluating reference architectures based on models (descriptions) is almost not existing, which suggests that prototyping is even more important than for one-of-a-kind architecture development.

2.5 Evaluate Architecture

In this step, the architecture resulting from the previous step is evaluated according to the architecture evaluation plan. If the evaluation is successful (i.e., all tests are passed), the architecture development continues with the next iteration or is finished once the last group of scenarios has been applied. If, however, at least some test failed, the process continues with step 2.6 “Analyze Problem”.

Product Line Aspects. Evaluation has to address instance-specific as well as family-specific aspects and relies on a defined instantiation mechanism. First, the *ease* with which instances can be created allows to draw conclusions on critical family-specific characteristics (i.e., usability, maintainability, etc.). Second, the range of possible instantiations is used to ensure that the intended products are indeed covered by the reference architecture. Third, instantiation yields variability-free architectures that can be evaluated in the same way as one-of-a-kind ones. It is obvious that it is difficult to both get and interpret these results without relying on a prototype and on an automatic instantiation mechanism.

A lot of instance-specific tests tend to be applicable for several or even all instances. It is therefore sensible to use this process step as a starting point for the construction of an infrastructure for testing and debugging architectures of product family members as proposed by Balzer [1].

2.6 Analyze Problem

At least one of the tests for evaluating the current architecture failed. In this step, the underlying problem is examined in order to determine how the architecture development process can be continued. The examination focuses on whether the current group of scenarios could be applied successfully to the architecture that resulted from the previous iterations. If this is deemed to be the case, only the current

iteration needs to be reiterated. Otherwise, some design decisions from an earlier iteration are presumed to impose constraints that are too stringent for the current set of scenarios. Therefore, extended backtracking is needed, which may include reformulating, regrouping, and reordering of some scenarios and then reentering the process in the appropriate iteration.[6]

Product Line Aspects. The task of finding out whether a given, partial reference architecture is compatible with a new set of requirements is significantly more difficult than to judge that for a one-of-a-kind architecture. Ensuring that an addition does not break any of the possible instances of the architecture is a highly complex task, especially when it involves dealing with entities that could not be encapsulated properly. It is noteworthy that this problem is not only relevant for architecture creation but also for maintenance. We are not aware of any published guidelines on how this problem should be addressed.

3 Summary and Future Work

In this paper we have presented the PuLSE-DSSA method, while discussing its customizability and how it is to be applied for the creation and evaluation of product line architectures. This discussion was used to highlight how methods supporting product line architectures must deal with complexities non-existent in those for one-of-a-kind software architectures.

PuLSE-DSSA presents the framework for most of our current research in software architectures. Some of the issues we are currently working on include means of representing product line architectures with their commonalities and variabilities; the derivation of instance architectures from a given product line architecture; supporting traceability between architectural elements and various other assets; and the evaluation of product line architectures.

References

1. R. Balzer, “An Architectural Infrastructure for Product Families,” in *Proceedings of the Second International Workshop on Development and Evolution of Software Architectures for Product Families*, Lecture Notes in Computer Science 1429, pp. 158-160, Springer, 1998
2. J. Bayer et al., “PuLSE: A Methodology to Develop Software Product Lines,” in *Symposium on Software Reusability’99 (SSR 99)*, pp. 122-131, May 1999
3. J.-M. DeBaud and K. Schmid, “A systematic approach to derive the scope of software product lines,” in *Proceedings of the 21st International Conference on Software Engineering (ICSE 99)*, pp. 34-43, 1999

[6] In some situations backtracking to the requirements definition phase (e.g., domain modeling) may be necessary. This involves change management processes that are outside the scope of this paper but are addressed within PuLSE.

4. C. Gacek, A. Abd-Allah, B. Clark, and B. Boehm, “On the Definition of Software Architecture,” in *Proceedings of the First International Workshop on Architectures for Software Systems*, D. Garlan (ed.), Seattle, WA, pp. 85-95, 24-25 April 1995
5. R. Kazman, G. Abowd, L. Bass, P. Clements, “Scenario-Based Analysis of Software Architecture,” *IEEE Software*, vol. 13, no. 6, pp. 47-55, November 1996
6. R. Kazman, M. Barbacci, M. Klein, S.J. Carriere, and S.G. Woods. “Experience with Performing Architecture Tradeoff Analysis,” in *Proceedings of the 21st International Conference on Software Engineering (ICSE 99)*, pp. 54-63, 1999
7. N. Medvidovic, “A Classification and Comparison Framework for Software Architecture Description Languages”, *Technical Report UCI-ICS-97-02*, University of California, Irvine, CA, Feb. 1997
8. D. Perry, “Generic Architecture Descriptions for Product Lines,” in *Proceedings of the Second International Workshop on Development and Evolution of Software Architectures for Product Families*, Lecture Notes in Computer Science 1429, pp. 51-56, Springer, 1998