

# Separating Concerns in Product Lines Engineering

Joachim Bayer

*Fraunhofer Institute for Experimental Software Engineering (IESE)*

*Sauerwiesen 6*

*D-67661 Kaiserslautern, Germany*

*bayer@iese.fhg.de*

## Abstract

*Product line engineering aims at improving development and maintenance efficiency for families of related systems by building a reuse infrastructure covering all systems in the product family. Developing and maintaining such an infrastructure is a complex endeavor. Two main issues have to be addressed to make product line engineering successful. First, the handling of variability is needed to enable the infrastructure to encompass the complete product family and, second, the consistent and systematic managing of the evolution of the product line is required to assure its integrity over time.*

*A clear separation of concerns can improve the situation with respect to both issues by distinguishing common and variable aspects of product line models and by making the concerns and forces explicit that affect the creation and maintenance. In this paper, we argue that the issues must be addressed separately, show solutions to both of them, and discuss how these solutions collaborate.*

## 1. Introduction

### 1.1. Motivation

Product line engineering is an approach to improve development efficiency [1]. The two key benefits of product line engineering are the improvement of the development efficiency for families of related systems by facilitating large-scale reuse and the possibility to share the maintenance effort for the product line members. A product family is a set of systems whose functionalities overlap. The core of a product line is a product line infrastructure, a reuse infrastructure that supports the location, the evaluation, and the adaptation of reusable assets, as well as a more accurate planning of development projects. The single products are assembled from generic reusable assets, and can therefore be developed and maintained more efficiently.

The development as well as the evolution and maintenance of a product line is a difficult task. To the difficulties that arise when developing a single

application, product lines add some. Each of the products in the product line has stakeholders with their concerns on the product. The anticipations and expectations of the stakeholders are directed to their products. Encompassing a family of products, the product line infrastructure must reflect the concerns related to the different products. The concerns the different stakeholders have towards their product might lead to conflicts in the product line infrastructure. The results are inconsistencies in the product line infrastructure that can not be resolved, because all concerns have to be supported by the product line. There are also stakeholders of the product line as a whole that have concerns on the product line infrastructure directly. Another important factor is that a product line is a costly endeavor and can only pay off if it is delivering products over time. To benefit from such a long-term investment it must be kept up to date. Therefore, maintenance can not be reduced to the different products that have been built from the product line, but must encompass the product line infrastructure itself. To keep the infrastructure consistent with the derived products, the maintenance activities for the products and for the infrastructure must be coordinated [2].

Bosch and Högström investigated problems and issues arising when instantiating product lines for embedded applications [3]. The two causes that are responsible for the problems they found are variability and evolution. The variability problem consists of separating generic from specific aspects in a product line infrastructure. The evolution problem is related to the decreased quality of the product line infrastructure caused by maintenance and change. As described above the authors investigated the instantiation of product lines, but the causes they found are present at all phases of product line engineering.

In this paper, we propose to separate the solution to the two problems, even though, at a first glance, they might be addressable both in the same way: a clear separation of concerns can improve the situation with respect to both problems. The variability problem can be addressed by a separation of general and product-specific aspects of the product line infrastructure that makes commonalities and variabilities explicit. The

maintenance problem can be addressed using one of the numerous approaches for the separation of concerns that have been proposed for single system development.

## 1.2. Existing Approaches

The idea of separation of concerns is to describe and handle the important and critical aspects of a software system separately. This leads to reduced complexity in the description of the individual aspects. Separation of concerns during software development is realized by having a set of development models, in which each model describes the software system from a different perspective. The models together provide a view on the system that contains more information than the cumulative information of the individual models. The reduced complexity in the individual views results in an increased comprehensibility of the views and the overall system, and thereby supports evolution and maintenance. The explicit relations between concerns and the views enable traceability among models and facilitate reuse.

The need for separation of concerns has long been recognized in software engineering. The concept was introduced at different software development life cycle stages. In its beginning, separation of concerns was realized at code level and expressed as the need for modularization [4]. Many approaches for separation of concerns have been proposed. Recent examples are Aspect-Oriented Programming [5], Subject-Oriented Programming [6], and Multi-Dimensional Separation of Concerns [7]. Other approaches for separation of concerns include view based requirements engineering (e.g., in [8]), view based design (e.g., [9]), architectural views (the most influencing paper here was [10]), as well as object-oriented analysis and design (Catalysis [11] provides means to compose design models; in OORAM [12] concerns are modeled as roles that can be synthesized).

However, the existing approaches for separation of concerns do not take into account the special situation that product line engineering methods impose. They do not provide any support for variability management and decision modeling. Because product line engineering is concerned with the development of a reuse infrastructure that encompasses numerous products, it is a long-term investment that has numerous stakeholders over time. The stakeholders all have concerns on their products, some of which might contradict each other in the product line infrastructure. Because a product line infrastructure encompasses the full life cycle of a product line, concerns must be usable and reusable at different development stages. Additional problems arise when the set of models used in the product line infrastructure are customized, as it is the case in PuLSE, the product line engineering approach developed at IESE [13]. Its customization

includes the definition of model sets for the different development stages. Therefore, separation of concerns must provide the possibility to choose the used paradigm and set of models to model the concerns.

In principle, it would be possible to model variabilities as concerns. Yet, the fact that the variability concerns, that is, the product line as seen from the perspective of a product line member, enables reasoning about the product line, its members, and relations among the members, elevate this concerns to a higher level of abstraction. Therefore, we argue for separating the separation of commonalities from variabilities from the separation of concerns supporting the development and maintenance of a product line.

## 1.3. Overview

In the next section, our view on product line engineering and the separation of commonalities from variabilities is presented. Then, section 3 shows how a separation of concerns can improve the product line infrastructure and how the two dimensions of separation can be combined. Section 4 finally concludes.

## 2. Product Line Engineering

Product line engineering is usually split into three phases: the construction, the usage, and the maintenance of a product line infrastructure.

The initial step of the construction of the product line infrastructure is to determine its scope. That is, based on the stakeholders' business objectives and the planned products, the range of characteristics that systems in the product line should cover is identified. Then, subsequently, the requirements for the product line are documented in a product line model, a generic reference (or domain-specific) architecture is developed, and its components or modules are partially designed and implemented.

After the construction phase, the product line infrastructure consists of assets used during application engineering, when a single product line member is created. The product line usage 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.

Because a product line infrastructure is a long-term investment and concepts within the domain or other requirements on the product line change over time, the infrastructure must be continuously and systematically maintained.

A product line encompasses numerous products that are supposedly used over a long period of time. The long-term investment in an infrastructure that allows the

derivation of product line members can only pay off when, over the whole life-cycle, all product line members are based on the same common asset base, the product line infrastructure. To achieve that, the maintenance activities for the different product line members that are in production and the maintenance activities for the product line infrastructure must be coordinated.

## 2.1. Product Line Infrastructures

A product line infrastructure typically consists of two parts: firstly, a set of product line assets that are built for (re-) use during the application engineering process, and secondly, a decision model that represents the knowledge needed to build applications with the infrastructure.

**2.1.1. Product Line Assets.** A product line asset is any kind of asset that captures information about the systems of a product line. The assets are built to be reused when systems in the product line are developed. There are two types of information captured: information that is valid for all systems in the family (i.e., commonalities) and information that varies from system to system (i.e., variabilities).

The variation points in a product line infrastructure must be resolved when a particular product line member is specified. Resolving variation points means that parts of the asset are specialized, included, or excluded. The variation of system characteristics is at a higher level of abstraction than the system characteristics themselves. This higher abstraction level is not explicitly supported by most of the commonly used assets and notations. Hence, new asset elements are defined to enable the explicit integration of variabilities into an asset. This can be done by extending the meta model of an asset.

**2.1.2. Decision Model.** There is typically a large number of variation points in the assets of a complete product line infrastructure. Consequently, it is, even for experts, hard to control the rationales for each variation point, as well as the complex interrelationships and dependencies among them. To support the intellectual control, typically, a decision model, which captures this kind of domain knowledge, is built on top of the variation points.

A decision model consists of a decision hierarchy, which is grounded on simple decisions that capture the rationale and the possible choices for a single point of variation. Dependencies among decisions are explicitly captured by constraints. Usually, additional decisions are introduced. These decisions are not directly related to a point of variation of an asset but they represent domain variability at a higher level of abstraction. This higher abstraction level is related to sets of interdependent variation points.

## 2.2. Product Line Member Creation

A product line instance is a system that is developed with reuse of product line assets for a particular customer. It is the output of application engineering, the process of developing specific product line members. The tailoring of a product line asset while reusing it is a two-step process: first, its instantiation in the space of supported variabilities and, second, its extension with aspects that are not supported by the product line asset but that are required by a particular customer.

**2.2.1. Decision Model Resolution.** The decision model drives the application engineering process, that is, while traversing the decision hierarchy, decision-by-decision is resolved. When a decision is resolved that constrains variation points of a product line asset, the asset is instantiated accordingly, that is, the variation point is removed and replaced by the concrete realization that corresponds to the selected resolution. The resolution process stops when all simple decisions are resolved. The resulting asset instance is the variant of the variants supported by the product line infrastructure that is closest to what is required in the specific context. Often, even the closest variant is not exactly what is required and, thus, further modifications are necessary. The made resolutions are stored in a decision model instance.

**2.2.2. Customer-Specific Changes.** When none of the variants supported by the infrastructure is acceptable for a customer, additional requirements must be fulfilled either via infrastructure extensions or customer-specific changes - an important, strategic decision. The asset instance, which is the closest to the required one, is input for the extension process. Hence, the required asset instance is expressed in differences to the already supported instance. This simplifies the later integration generally with the infrastructure or the maintenance activities.

## 3. View Based Product Line Development

### 3.1. Separation of Concerns

The handling of variability as described above improves the situation with respect to the first cause of problems in product line engineering. The second cause - evolution - is not addressed yet. A clear separation of concerns improves the comprehensibility of the system, supports evolution and maintenance, enables consistency checking, facilitates reuse, and enables traceability among models. These characteristics help to improve the support of systematic and consistent maintenance that encompasses the product line infrastructure and the products that have been derived from it. Introducing the concept of separating concerns

and thereby moving towards view based product line development is the topic of this section.

As described above, a product line infrastructure contains all assets that are used to construct, use, and evolve the product line. These assets contain aspects that are common to all product line members and aspects that vary among them. An approach to create and use such an infrastructure coherently is to model all known forces that influence the product line infrastructure in an explicit way.

### 3.2. Introducing Views to Product Line Engineering

The separation and explicit capturing of all known relevant concerns and forces influencing a product line during its creation, usage, and maintenance results – in our opinion – in a view based product line development. View based software development approaches could be defined as approaches that allow taking different viewpoints on the software system under development and enables the explicit modeling of all of those viewpoints as views on the system. This is done to separate the different concerns the system stakeholders have. The views that have been separated must be related to each other or composed in order to retrieve the complete picture describing the system under development as a whole. Applied to product line engineering this means that a view based product line development method provides means to identify relevant viewpoints on the product line members and on the product line itself, to separate the appropriate views related to these viewpoints, and to compose the separates views.

Such a method must have the following four properties. First, it must be able to handle variants. Each view that is modeled potentially contains variability. Therefore, the method must be compatible with the variability handling presented in section 2.

Second, it must be able to handle unresolvable inconsistencies that may arise during product line engineering.

Third, it must be uniformly applicable to the complete product line life cycle. This includes that the viewpoint and view definitions can be reused in different life cycle phases and also that different definitions can be used in different phases.

The forth property is customizability. Product lines are developed for very diverse application areas. Consequently, the models that are produced are very different and also are based on different paradigms. We see the concept of view based product line engineering as an extension to existing and applied product line engineering methods. It can be added to a possibly large number of product lines developed using different ways of modeling. To achieve that, the method must be independent of the used model set and the underlying paradigm.

The approach we are proposing takes as its input the currently used set of models that are used to model the different phases in the product line engineering life cycle. These models are generic and can be instantiated to make them specific. The different concerns the stakeholders have on their product and thereby on the product line are elicited next. The concerns are made explicit in viewpoints that are used to cover the respective concern. Viewpoints can be, for example, role-related, that is the product line is seen from the perspective of the different roles that interact with the systems derived from the product line, or related to non-functional requirements.

The next step is to determine the models that are used to express the views related to the identified viewpoints. A view consists of a number of models that together are able to express how the product line and the derived product line members looks like from the respective viewpoint. This step is performed on the meta model describing the models. Each view is made up of interrelated models. Like the models that interact to form a view, the views interact and overlap. Two models or views are said to overlap if they designate common aspects. This overlap can be model elements being present in more than one model or transformation relations (e.g., methods are derived from use cases). This overlap is captured at the meta level and provides a basis for consistency checks and traceability. A life cycle phase is then defined as a set of interrelated views. The phases are also interrelated. The result is a meta model capturing the complete product line life cycle.

Using the meta level definition of the product line, the different stages of product line development can be modeled in a view based way. This means that a modeler can only use the elements defined in the meta model.

Based on the overlap definition, relations among views can be investigated and views can be combined. Inconsistencies can be revealed this way.

### 3.3. Variability Handling

The view based approach to product line development introduced is compatible with the approach for variability handling described in section 2. The generic models are combined into views. This enables views to be generic and specific, too.

There is clearly a relation between the variants in a product line and the realizations of a concern. That is, a variant potentially imposes different requirements on a viewpoint. To satisfy the different requirements, the views might be different. But there are more aspects than variants (e.g., aspects induced by roles or non-functional requirements). Another important point is that variability handling and views are at a different level of abstraction that should not be mixed.

## 4. Conclusions

Variability handling and consistent evolution of the infrastructure are key to successful product line engineering. In this position paper, we argued for addressing them separately and proposed a view based approach to product line engineering as a solution.

## 5. References

- [1] P. Donohoe (ed.). *Software Product Lines. Experience and Research Directions*. Proceedings of the First Software Product Lines Conference (SPLC1), Kluwer Academic Publishers, August 2000.
- [2] J. Bayer and D. Muthig, Maintenance Aspects of Software Product Lines, In *Proceedings of 1. Deutscher Software-Produktlinien Workshop. DSPL-1*, Kaiserslautern, Germany, 2000.
- [3] J. Bosch and M. Höglström, Product Instantiation in Software Product Lines: A Case Study, In *Proceedings of Net.ObjectDays 2000*, pp. 342-357, Erfurt, Germany, 2000.
- [4] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053-1058, December, 1972.
- [5] G. Kiczales, J. Lamping, A. Mendhekar, C. Medea, C. Lopes, J.-M. Loingtier, and J. Irving. Aspect-oriented Programming. In *Proceedings of the 11th European Conference on Object-Oriented Programming (ECOOP'97)*, pp. 220-242, Jyväskylä, Finland, June 1997.
- [6] W. Harrison and H. Tarr. Subject-Oriented Programming (A Critique of Pure Objects). In *Proceedings of the 8th Conference on Object-Oriented Programming: Systems, Languages, and Applications (OOPSLA'93)*, pp. 411-428, Washington, D.C., September 1993.
- [7] P. Tarr, H. Ossher, W. Harrison, S. Sutton. N Degrees of Separation: Multi-Dimensional Separation of Concerns. In *Proceedings of the 21st International Conference on Software Engineering*, Los Angeles, CA, May 1999.
- [8] A. Finkelstein, J. Kramer, B. Nuseibeh, L. Finkelstein, and M. Goedicke. Viewpoints: A Framework for Integrating Multiple Perspectives in System Development. In *International Journal of Software Engineering and Knowledge Engineering*, Vol. 2, No. 1, pp. 31-57, 1992.
- [9] S. Clarke, W. Harrison, H. Ossher, and P. Tarr. Subject-Oriented Design: Towards Alignment of Requirements, Design and Code. In *Proceedings of the 14th Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'99)*, pp. 325-339, Denver, CO, USA, November 1999.
- [10] P. Kruchten. The 4+1 View Model of Architecture. *IEEE Software*, November 1995, pp. 42-50.
- [11] D. F. D'Souza and A. C. Wills. *Objects, Components, and Frameworks with UML: The Catalysis Approach*. Addison-Wesley, 1999.
- [12] T. Reenskaug, P. Wold, and O. A. Lehne. *Working with Objects: The OOram Software Engineering Method*. Manning Publications Co., 1995.
- [13] 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. In *Proceedings of the Symposium on Software Reuse (SSR'99)*, May 1999.