

# Adopting Software Product Lines: Approaches, Artefacts and Organization

Jan Bosch  
University of Groningen  
Department of Computing Science  
PO Box 800, 9700 AV, Groningen  
The Netherlands

[Jan.Bosch@cs.rug.nl](mailto:Jan.Bosch@cs.rug.nl) <http://www.cs.rug.nl/~bosch>

## Abstract.

Software product lines have received considerable adoption in the software industry and prove to be a very successful approach to intra-organizational software reuse. Contemporary literature on the subject, however, often presents only a single approach towards adopting a software product line. In this paper, we present an overview of different adoption approaches, different maturity levels for product line artefacts and different organizational models.

## 1. Introduction

Software product lines have achieved substantial adoption by the software industry. A wide variety of companies has substantially decreased the cost of software development and maintenance and time to market and increased the quality of their software products. The product line approach is basically the first intra-organizational software reuse approach that has proven successful.

Contemporary literature on product lines often presents one particular approach to adopting a product line, suggesting a particular process model, a particular organizational model and a specific approach to adopt the product line approach. For instance, most existing literature assumes the organization to have adopted a domain engineering unit model, where this unit develops reusable artefacts while the product engineering units develop concrete products based on these reusable assets.

However, in our experiences with software companies that have adopted a product line approach, we have learned that the actually available alternatives are generally much more than the particular approach presented in traditional literature. The adoption of a product line, the product line processes and the organization of software development have more freedom than one may expect.

In the remainder of this paper, we present the various approaches that are available when working with software product lines. In the next section, we discuss four types of product line adoption. In section 3, we discuss the maturity levels for each main product line artefacts that we have identified. The organizational models one may adopt are discussed in section 4. The paper is concluded in section 5.

## 2. Product Line Adoption

Software product lines do not appear accidentally, but require a conscious and explicit effort from the organization interested in employing the product line approach. Basically, one can identify two relevant dimensions with respect to the initiation process. First, the organization may take an evolutionary or a revolutionary approach to the adoption process. Secondly, the product line approach can be applied to an existing line of products or to a new system or product family that the organization intends to use to expand its market with. Each case has an associated risk level and benefits. For instance, in general, the revolutionary approach involves more risk, but higher returns compared to the evolutionary approach. In table 1, the characteristics of each case are briefly described.

	Evolutionary	Revolutionary
Existing set of products	Develop vision for product line architecture based on the architectures of family members Develop one product line component at a time (possibly for a subset of product line members) by evolving existing components	Product line architecture and components are developed based on super-set of product line member requirements and predicted future requirements.

New product line	Product line architecture and components evolve with the requirements posed by new product line members	Product line architecture and components developed to match requirements of all expected product line members
------------------	---	---

Table 1. Two dimensions of product line initiation

The advantage of the evolutionary approach is that risk is minimized in two ways. First, since the up-front investment is decomposed into small steps for each component, the pay-off of sharing components gives a quick return of investment and the investment itself is relatively small. Second, the products in the family continue their normal evolution, albeit at a somewhat slower pace, whereas the revolutionary approach generally stops all but cosmetic evolution of the products in the family until the product line architecture and components are in place. The disadvantage is that the total amount of investment until the product line architecture and components are completely in place is larger than when using the revolutionary approach. This because much work on components done during the conversion is only performed to support temporary requirements.

The advantages of replacing an existing set of products with a product line, i.e. a revolutionary approach, are especially the shorter conversion time and the generally smaller total investment required for developing the architecture and component set for the family, when compared to the evolutionary approach. The primary disadvantages are the increased risk level, due to the large initial investment that may prove useless if important requirements change, and the delayed time-to-market of the first products developed based on the product line architecture.

Finally, an important factor in the decision to either evolve or replace a set of systems is the presence of mechanical and hardware parts. If systems, in addition to software contain considerable pieces of mechanics and hardware, the product line approach needs to be synchronized for all three aspects. Especially if considerable differences exist in the mechanical and hardware parts of the products, it is generally harder to employ an evolutionary approach and replacing the existing systems may simply be the only alternative.

### 3. *Product Line Artefacts*

One can identify three types of artefacts that make up a software product line, i.e. the product line architecture, shared components and the products derived from the shared artefacts. For each of these artefacts, one can identify three maturity levels, depending on the level of integration achieved in the product line. Below, we discuss each artefact in more detail.

The software architecture of the product line is the artefact that defines the overall decomposition of the products into the main components. In doing so, the architecture captures the commonalities between products and facilitates the variability. One can identify three levels of maturity:

- **Under-specified architecture:** A first step in the evolutionary adoption of a software product line, especially when converging an existing set of products, is to first define the common aspects between the products and to avoid the specification of the differences. This definition of the architecture gives existing and new products a basic frame of reference, but still allows for substantial freedom in product specific architectural deviation.
- **Specified architecture:** The next maturity level is to specify both the commonalities and the differences between the products in the software architecture. Now, the architecture captures most of the domain covered by the set of products, although individual products may exploit variation points for product specific functionality. The products still derive a product specific architecture from the product line architecture and may consequently make changes. However, the amount of freedom is substantially less than in the under-specified architecture.
- **Enforced architecture:** The highest maturity level is the enforced architecture. The architecture captures all commonality and variability to the extent where no product needs, nor is allowed, to change the architecture in its implementation. All products use the architecture as-is and exploit the variation points to implement product specific requirements.

The second type of artefact is the product line component, shared by some or all products in the product line. Whereas the product line architecture defines a way of thinking about the products and rationale for the structure chosen, the components contribute to the product line by providing reusable

implementations that fit into the specified architecture. Again, one can identify three levels of maturity for product line components:

- **Specified component:** The first step in converging a set of existing products towards a product line is to specify the interface of the components defined by the architecture. A component specification typically consists of a provided, a required and a configuration interface. Based on the component specifications, the individual products can evolve their architecture and product specific component implementations towards the product line thereby simplifying further integration in the future.
- **Multiple component implementations:** The second level of maturity is where, for an architectural component, multiple component implementations exist, but each implementation is shared by more than one product. Typically, closely related products have converged to the extent that component sharing has become feasible and, where necessary, variation points have been implemented in the shared components.
- **Configurable component implementation:** The third level is where only one component implementation is used. This implementation is typically highly configurable since all required variability has been captured in the component implementation. Often, additional support is provided for configuring or deriving a product specific instantiation of the component, e.g. through graphical tools or generators.

The third artefact type in a software product line is the products derived from the common product line artefacts. Again, three levels of maturity can be distinguished:

- **Architecture conformance:** The first step in converging a product towards a product line is to conform to the architecture specified by the product line. A product can only be considered a member of the product line if it at least conforms to the under-specified architecture.
- **Platform-based product:** The second level is the minimalist approach where only those components are shared between products that capture functionality common to all products. Because the functionality is so common, typically little variability needs to be implemented.
- **Configurable product base:** The third level of maturity is the maximalist approach, where all or almost all functionality implemented by any of the product line members is captured by the shared product line artefacts. Products are derived by configuring and (de-)selecting elements. Often, automated support is provided to derive individual products.

## 4. *Organizational Models*

In this section, we discuss a number of organizational models that can be applied when adopting a software product line based approach to software development. Below, we briefly introduce the models. For a more extensive discussion, we refer to [1].

- **Development department:** When all software development is concentrated in a single development department, no organizational specialization exists with either the product line assets or the products in the product line. Instead, the staff at the department is considered to be resource that can be assigned to a variety of projects, including domain engineering projects to develop and evolve the reusable assets that make up the product line.
- **Business units:** The second type of organizational model employs a specialization around the type of products. Each business unit is responsible for one or a subset of the products in the product line. The business units share the product line assets and evolution of these assets is performed by the unit that needs to incorporate new functionality in one of the assets to fulfil the requirements of the product or products it is responsible for. On occasion, business units may initiate domain engineering projects to either develop new shared assets or to perform major reorganizations of existing assets.
- **Domain engineering unit:** This model is the suggested organization for software product lines as presented in the traditional literature, e.g. Dikel et al. [2] and Macala et al. [4]. In this model, the domain engineering unit is responsible for the design, development and evolution of the reusable assets, i.e. the product line architecture and shared components that make up the reusable part of the product line. In addition, business units, often referred to as product engineering units, are responsible for developing and evolving the products based on the product line assets.
- **Hierarchical domain engineering units:** In cases where an hierarchical product line has been necessary, also a hierarchy of domain units may be required. In this case, often terms such as 'platforms' are used to refer to the top-level product line. The domain engineering units that work with specialized product lines use the top-level product line assets as a basis to found their own product line upon.

Some factors that influence the organizational model, but that we have not mentioned include the physical location of the staff involved in the software product line, the project management maturity, the organizational culture and the type of products. In addition to the size of the product line in terms of the number of products and product variants and the number of staff members, these factors are important for choosing the optimal model.

## **5. Conclusions**

Software product lines have received wide adoption in many software companies and proven to be very successful in achieving intra-organizational reuse. Traditional approaches to software product line based development typically take one particular approach. In this paper, we discussed the different alternatives that are available when adopting a product line, the maturity levels of product line artefacts and different organizational models that are available.

We have discussed four different approaches to adopting a software product line, organized in two dimensions, i.e. evolutionary versus revolutionary adoption and replacing an existing set of products versus developing a new set of products.

In addition, we have discussed maturity levels for the main product line artefacts. The product line architecture can be under-specified, fully specified or enforced. The components can just consist of a component interface specification, multiple component implementations and one configurable component implementation. Finally, a product can conform to the product line architecture, be a platform-based product or be derived from a configurable product base.

Finally, we discussed four alternative organizational models, i.e. the development department model, the business unit model, the domain engineering unit model and the hierarchical domain engineering unit model.

## **References**

- [1] J. Bosch, *Design and Use of Software Architectures: Adopting and Evolving a Product Line Approach*, Pearson Education (Addison-Wesley & ACM Press), ISBN 0-201-67494-7, May 2000.
- [2] D. Dikel, D. Kane, S. Ornburn, W. Loftus, J. Wilson, 'Applying Software Product-Line Architecture,' *IEEE Computer*, pp. 49-55, August 1997.
- [3] J. van Gurp, J. Bosch, M. Svahnberg, 'On the Notion of Variability in Software Product Lines,' Accepted for The Working IEEE/IFIP Conference on Software Architecture (WICSA 2001), April 2001.
- [4] R.R. Macala, L.D. Stuckey, D.C. Gross, 'Managing Domain-Specific Product-Line Development,' *IEEE Software*, pp. 57-67, 1996.
- [5] D.M. Weiss, C.T.R. Lai, *Software Product-Line Engineering - A Family-Based Software Development Process*, Addison-Wesley, ISBN 0-201-69438-7, 1999.