Maturity and Evolution in 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 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. Existing literature, however, often presents only a single approach towards adopting and evolving a software product line. In this paper, we present an overview of different approaches to architecture-centric, intra-organizational reuse of software artefacts. We relate these to maturity levels for product line artefacts and 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 software product line approach can be considered to be 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 or application 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 diverse 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.

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.

Table 1 *Two dimensions of product line initiation*

	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

Independent of the adoption approach taken, the organization will evolve its approach to software development. Typically, the software product line developed by the organization typically evolves through a number of maturity levels[1]. As we will discuss later in the paper, these levels include standardized infrastructure, software platform, software product line, configurable product base, program of product lines and product populations. For a given scope in terms of features and products containing a subset of these features, each of these approaches can be applied. Thus, these approaches present different solutions to the same problem.

The approach that is most appropriate for a particular situation is a function of the maturity of the organization and the maturity of the application domain. Organizations more mature in terms of domain understanding, project organization and management and with less geographical distribution will more easily adopt more mature product line approaches that typically focus more on domain engineering. A second influencing factor is the maturity of the application domain itself. For stable domains, it is easier to maximize domain engineering and minimize application engineering, because these investments have a high likelihood of providing an adequate return. It is important to stress that the economically optimal approach is a function of the maturity of the organization and of the application domain. In highly volatile domains, it may, even for a very mature organization, be preferable to only employ a standardized infrastructure approach.

In addition to the maturity of the overall product line approach, one can also identify different maturity levels for the artefacts that are developed and used as part of the

1. Our use of the term ‘maturity level’ has no relation its use in the capability maturity model (CMM).

product-line approach. These artefacts include the architecture, the components and the products. For each of these, we discuss three maturity levels. Finally, different organizational models can be used for product line development.

The remainder of the paper is organized as follows. In the next section, we discuss the six maturity levels that we have identified for software product line approaches. In section 3, we discuss the maturity levels for the main product line artefacts, i.e. the product line architecture, the shared product line components and the products. The organizational models one may adopt are discussed in section 4. In section 5, we discuss the relation between the product line approaches, the artefacts and organizational models. Related work is discussed in section 6 and the paper is concluded in section 7.

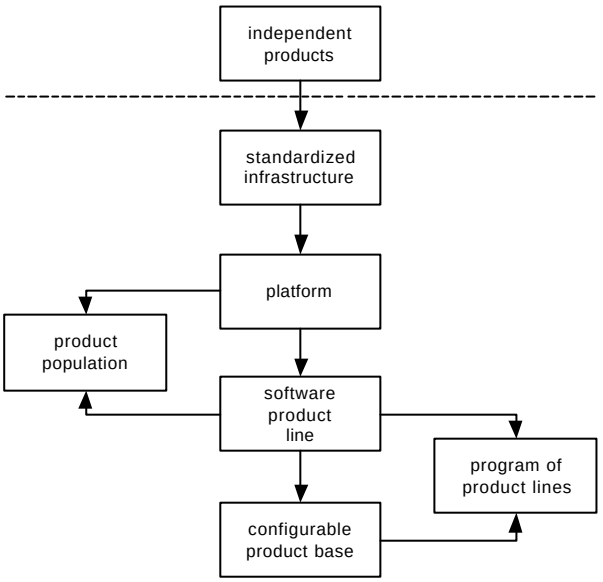

Figure 1. Maturity levels for software product lines

2 Maturity Levels of Software Product Lines

As mentioned in the introduction, architecture-centric, intra-organizational reuse takes place in various forms. In our experience, these approaches can be organized in a number of levels. The approaches that we have identified are ranging from a standardized infrastructure based on which products are created to a configurable product base that can be used to derive a variety of products. Thus, for a set of products covering a defined scope and that exhibit some commonality, an organization has a number of alternatives for developing and evolving these products, in addition to developing each product from scratch.

In figure 1, the different maturity levels are presented graphically. Starting from a situation in which each product or application is developed independently, the main maturity development path consists of a standardized infrastructure, a platform, a soft-

ware product line and finally a configurable product base. Two additional developments can be identified, i.e. product populations and a program of product lines. A product population approach is chosen when the organization decides to increase the scope in terms of the number of products. The program of product lines is selected when the scope of the product line is extended in terms of the supported features. Typically, there is an overall architecture defining a set of components where each component is again a software product line. In the sections below, the approaches are described in more detail.

2.1 Standardized infrastructure

Description. The first step that an organization takes when evolving towards exploiting commonality in its products is to standardize the infrastructure based on which the products are developed. This infrastructure typically consists of the operating system and the typical commercial components on top of it such as a database management system and a graphical user interface. In addition, the organization may acquire some domain-specific components from external sources. These components are typically integrated through some proprietary glue code.

Domain versus application engineering. Typical for this approach is that although it provides a first step towards sharing software artefacts, it requires no or very little domain engineering effort. Except for creating and maintaining the glue code, which is typically rather small compared to the size of the applications, all effort is directed to application (or product) engineering.

Variability management. The common infrastructure contains no domain specific functionality and therefore no variability management is necessary for the domain specific functionality. The infrastructure components may contain variations, but these need to be managed as in traditional software development.

Example. An example of a company exploiting this approach is Vertis Information Technology. This company develops administrative applications typically supporting some technical production systems. Vertis typically builds its applications on top of a Windows NT platform running the Oracle database system and associated tool set. They have bought a number of domain specific components specific for the Dutch taxation system. These components have been integrated with the Oracle tool set and the operating system to form a rather advanced infrastructure based on which the applications are constructed.

2.2 Platform

Description. The next level in achieving intra-organizational reuse is when the organization develops and maintains a platform based on which the products or applications are created. A platform typically includes a standardized infrastructure as discussed in

the previous section. On top of that, it captures all functionality that is common to all products or applications. The common functionality that is not provided by the infrastructure is implemented by the organisation itself, but typically the application development treats the platform as if it was an externally bought infrastructure.

Domain versus application engineering. This approach typically requires a certain amount of domain engineering effort to create and maintain the platform. The main effort, however, is still assigned to application engineering. Although we discuss organizational issues later in this paper, one can typically observe that the organizational model typically is not changed when the platform is adopted. The platform is developed as a project by a team of persons from the different product units or by a separate organizational unit, but it typically captures the functionality that is obviously common to all products. Consequently, relatively little communication between the products units and the platform developers is required. Although product units are encouraged or even told to use the platform, this does not necessarily happen.

Variability management. Since the platform only captures the common functionality of the products, there typically are relatively few variation points. A possible exception may be provided by the variations that cross-cut all products, e.g. different infrastructures, secure versus non-secure versions, etc. Such variations that are common to all products can be captured in and supported by the platform. Such variations need to be managed explicitly.

Example. An example of a company employing the platform approach is Symbian Ltd. The company develops the Symbian OS (earlier known as EPOC), an operation system, application framework and application suite for personal digital assistants and mobile phones. The Symbian OS is distributed in three device family requirement definitions (DFRDs), i.e. a landscape display communicator (640 x 200 pixels and up), a portrait display communicator (approximately 320 x 240 pixels) and a smart phone family. Each of these DFRDs is distributed to the licensees as a platform that has to be extended with device specific functionality and licensee-specific applications.

2.3 Software product line

Description. Once the benefits of exploiting the commonalities between the products become more accepted within the organization, a consequent development may be to increase the amount of functionality in the platform to the level where functionality common to several but not all products becomes part of the shared artefacts. Now we have reached the stage of a software product line. Functionality specific to one or a few products is still developed as part of the product derivation. Functionality shared by a sufficient number of products is part of the shared product line artefacts, with the consequence that individual products may sacrifice resource efficiency or development effort for being part of the product line.

Domain versus application engineering. Typically, the amount of effort required for domain engineering is roughly equal to the amount of effort needed for product development or application engineering. Also, since the scoping of the features supported by the shared product line artefacts is much more intimately connected to the features supported by the products, explicit and periodic scoping and road mapping processes are typically put in place.

Variability management. Once the product line stage has been reached, managing the variability supported by the shared artefacts becomes a real challenge. Depending on stability of the domain, the supported variation points change frequently, both in binding time (typically to a later stage) and the set of variants (typically extended). In most cases that we have studied, however, there is no automated support for variability management.

Example. An example of an organization employing this approach is Axis Communication AB, Lund. At the time that we studied the company, it developed a range of network devices such as scanner servers, printer servers, storage servers and camera servers. These products were developed by as many business units and based on a set of common software artefacts, i.e. a product line architecture and a set of more than 10 object-oriented frameworks that were used as configurable product line components.

2.4 Configurable product base

Description. Especially if the organization develops products in relatively stable domains and derives many product instances, there is a tendency to further develop the support for product derivation. The consequence is that the organization, rather than developing a number of different products, moves towards developing only one configurable product base that, either at the organization or at the customer site, is configured into the product bought by the customer. Some companies use, for instance, license key driven configuration, shipping the same code base to each and every customer. The code base configures itself based on the provided license key.

Domain versus application engineering. Once the configurable product approach has been fully adopted, all development effort has moved towards domain engineering. Very little or no application engineering is provided because the product derivation typically is supported by automated tools or techniques. As presented in figure 1, this approach represents the highest maturity level.

Variability management. Since product derivation typically is automated once this level is reached, all variation points have an explicit representation in the tool configuring the product for the particular instantiation. The variants for each variation point are part of the configurable product base and adding new variants to variation points during product derivation is most often not supported. A common characteristic for companies applying this approach is that they understand the domain exceptionally

well and, due to that, are able to provide almost all variability a customer may want as part of the delivered code base.

Example. An example of an organization applying this approach is Telelarm AB, selling fire alarm systems. After several years developing and evolving an object-oriented framework for applications in the domain, the company reached the level where each customer basically received the same code base. The persons performing the installation at the customer site are supported by a configuration tool that manages the customer specific configuration of the product.

As described in the introduction to this section, two additional directions of evolution can be identified. In these cases, the organization decides to extend the scope of the product line in terms of the features covered by the product line or the set of products. In the subsequent sections, these approaches are described in more detail.

2.5 Programme of product lines

Description. Especially for very large systems, the program of product lines approach can be used. The approach consists of a software architecture that is defined for the overall system and that specifies the components that make up the system. Several or all of the components are software product lines. The result is a system that can be configured as the configurable product described above. However, because of its size, the configuration of the components is basically performed through product line-based product derivation or by using the configurable product base approach.

Domain versus application engineering. The division of tasks between domain an application engineering is similar to the basic software product line approach or configurable product approach, depending on which approach is taken by the component developers. However, the actual work when deriving a system is of course more in this approach. First the overall architecture needs to be derived for the specific system. In some cases, the architecture is constant for all derivations, but in others cases deviations may occur, e.g. components are excluded. Subsequently, for each architectural component, a derivation from the associated software product line needs to take place. Finally, the system needs to be integrated and validated. These application engineering activities may take substantial effort, depending on the amount of derivation support that has been developed during domain engineering.

Variability management. Managing the complexity resulting from the amount of available variability is typically a real challenge, although it depends on the approach taken. The most complex cases often result from dependencies between variation points in different software product lines in the system. Also, customer specific extensions to the system may create difficulties as these may cross-cut the system's overall architecture.

Example. An illustrative example of this approach is provided by Nokia Networks. The main product of this division are telecom switches. Each system consists of several product families. These product families are developed on top of platforms that in turn consist of reusable design blocks consisting of reusable components. A system delivered to a customer requires selection and configuration of each of the aforementioned elements. Change management in this context is a highly complex endeavour. Nokia Networks employs an approach that they refer to as System Feature Management.

2.6 Product populations

Description. Whereas the approach discussed above extends the set of features covered by a single system, the product population approach extends the set of products that can be derived from the shared product line artefacts. This does not refer to the situation where the same feature scope is populated with a more fine-grained set of products, but rather to the situation where the set of covered features is extended to allow for a more diverse set of products to be derived.

Domain versus application engineering. An interesting consequence of this approach is that the extent to which a predefined architecture can be imposed on each product diminishes with this approach. Instead, the components need to be combined in more diverse ways and need to be prepared for this during domain engineering. The division between domain and application engineering is typically similar to that of software product lines.

Variability management. The main difference between this approach and the aforementioned approaches is the fact that variability is not just present within the components, but also in the different ways in which components can be composed. Different products may employ very different configurations. The component interfaces, consequently, need to be prepared for being bound to various interfaces and may need to be able to cope with smaller mismatches.

Example. Philips Consumer Electronics presents an excellent example of this approach. As discussed in [9], Philips has adopted an approach where a set of components can be used to derive a variety of different products, including analog televisions, digital televisions, video recorders and digital set-top boxes. A number of teams around the world develop components that adhere to the defined architecture and its underlying principles. Other teams create individual products by selecting, composing and configuring components.

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 architectures 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 designed 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 [1] 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 [1], 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. [4] and Macala et al. [7]. 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 Relating Maturity Levels, Artefacts and Organization

In the previous sections, we have first discussed maturity levels of software product lines. Subsequently, we discussed maturity levels for the product line artefacts, i.e. architecture, components and products. Finally, we discussed the different organisational models that can be applied when employing software product lines. However, the various maturity levels and approaches cannot be combined arbitrarily. In our experience, certain combinations work well together where others do not. In table 2, we relate the product line approaches to the artefact maturity levels and organizational models that we have discussed. Although the absence of a ‘+’ or ‘+/-’ does not necessarily indicate an incompatibility, these combinations are, in our experience, less likely or require additional effort to achieve.

Below, we discuss the combinations in the table for each product line maturity level. The standard infrastructure approach cannot impose more than an underspecified architecture. Since the infrastructure typically only provides relatively generic behaviour, it cannot fully specify a software architecture for the product line. In addition, the standard infrastructure can only specify the part of component interfaces that is concerned with the functionality provided by the infrastructure. For instance, in the case of an object-oriented database being part of the standard infrastructure, persistent compo-

nents need to support interfaces for persistence, queries and transactions. Products based on the standard infrastructure only need to conform to the architectural restrictions imposed, but are free otherwise. Since the standard infrastructure requires little effort from the organization, there is no need for a separate organizational unit taking maintaining and evolving the shared artefacts. Consequently, the development department and business unit organizational models are primarily suited for this approach.

Table 2 *Relating maturity levels to SPL artefacts and organisation*

	standard infr.	*platform*	*SPL*	*conf. prod. base*	*product popul.*	*program of SPLs*
u.spec. SA	+	+			+	
spec. SA			+		+	+
enf. SA				+		
spec. C.	+/-	+			+/-	
mult. C.I.		+/-	+		+	+/-
conf. C.			+/-	+		+
arch. conf.	+					
platf. b. P		+	+		+	
conf. P base				+		+
dev. dept.	+	+	+			
bus. units	+	+	+			
D.E. unit			+	+	+	
hier. DE units					+	+

The platform approach typically employs an underspecified architecture, because it lacks the information about specific products constructed on top of the platform. Compared to the standardized infrastructure approach, the platform architecture typically demands more conformance from products in terms of architectural rules and constraints that have to be followed. Component interfaces are typically specified at least partially and for the common behaviour, component implementations may be provided by the platform. Products are constructed on top of the platform, as indicated in the table. Platforms are typically created through the efforts of a dedicated domain engineering team, but do not necessarily require the existence of a domain engineering unit during the usage and evolution of the platform.

The software product line approach specifies a product line architecture that captures the commonalities and variabilities of the products. For each architectural component, one or more component implementations are provided. For the more stable and

well understood components, multiple implementations may have merged into one configurable component implementation. Products are based on the product line, but may deviate where necessary. We have worked with companies that employ widely different organizational models, including the development department, business units and domain engineering unit models.

The most domain-engineering centric approach centres around a configurable product base. In this case, the architecture is enforced in that no product can deviate from the commonalities and variabilities specified in the architecture. Consequently, each architectural component typically has one associated configurable component implementation and the products are derived from these artefacts by excluding those parts not needed. A configurable product base is typically developed by a domain engineering unit but since there are no associated application engineering units, the organizational model is difficult to distinguish from the development department.

The product population approach increases the scope the product line in terms of features and products. Due to the wide variety of products, the architecture cannot be enforced. In addition, depending on the type of population, the architecture may even not be fully specified, because part of the variability is achieved by creative configurations of the components. There may be multiple component implementations, although some components may need to be specifically implemented for each product. Especially in the case where the product population is organized into a number of smaller sets, a hierarchical domain engineering unit model may be applied. Otherwise, a domain engineering unit model is used.

The final approach, i.e. a program of product lines, extends the scope of individual products in terms of the features. The overall architecture is typically well-specified as it defines the interaction between the component product lines. The architectural components are software product lines in themselves and, as such, typically rather configurable. Products are derived by configuring the product line components for the specific context in which these are used. Due to the size of the overall system, the hierarchical domain engineering unit model is typically employed.

6 Related Work

Software reuse is a long standing ambition of the software engineering community, dating back to the late 1960s [8] and 1970s [10]. The increasing popularity of the object-oriented paradigm during the 1980s lead to the definition of object-oriented frameworks. As part of this work, for instance, the REBOOT project defined reuse maturity levels [6].

During the 1990s, the notion of software product lines (or software system families) was adopted and achieved increasing attention as can be judged from the software product line initiative at the software engineering institute (SEI) [2] and several large european projects, including ARES, PRAISE, ESAPS and CAFE.

Several authors published work in related to this paper. Weiss and Lai [11] present an approach that relates to the maturity level that we refer to a configurable product base. In their approach, a generator is developed that generates the specific product

derivation that is required. Czarnecki and Eisenecker [3] also employ generative techniques to minimize the amount of application engineering needed.

7 Conclusions

Software product lines have received wide adoption in many software companies and have proven to be very successful in achieving intra-organizational reuse. Traditional literature to software product line based development typically takes one particular approach. However, in our experience with industry we have identified that companies employ widely different approaches and that these approaches evolve over time. These approaches can be organized in a number of maturity levels. These maturity levels include standardized infrastructure, software platform, software product line, configurable product base, program of product lines and product populations. For a given scope in terms of features and products, each of the approaches can be applied. The optimal approach for a company depends on the maturity of the organization, but also on the maturity of the application domain.

Next to the different approaches to software product lines, we have also presented maturity levels for the product line artefacts, i.e. the architecture, components and the products. 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. 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.

In section 5, we discussed the relations between the product line approaches, the artefacts and the organizational models. We presented the more common and logical combinations of approaches, maturity levels of artefacts and organizational models.

This paper is a first attempt to categorize the different approaches to software product lines. In the future, we intend to further validate, refine and extend this taxonomy.

Acknowledgements

Many thanks to Osmo Vikman from Nokia who kindly provided information about the program of product lines approach as used within Nokia Networks and to Rob van Ommering for his information on the product populations approach.

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] P. Clements, L. Northrop, Software Product Lines - Practices and Patterns, Pearson Education (Addison-Wesley), ISBN 0-201-70332-7, 2001.

[3] K. Czarnecki, U.W. Eisenecker, *Generative Programming - Methods, Tools and Applications*, Pearson Education (Addison-Wesley), ISBN 0-201-30977-7, 2000.

[4] D. Dikel, D. Kane, S. Ornburn, W. Loftus, J. Wilson, 'Applying Software Product-Line Architecture,' *IEEE Computer*, pp. 49-55, August 1997.

[5] J. van Gurp, J. Bosch, M. Svahnberg, 'On the Notion of Variability in Software Product Lines,' Proceedings of the Working IEEE/IFIP Conference on Software Architecture (WICSA 2001), August 2001.

[6] E-A. Karlsson, Editor, *Software Reuse - a Holistic Approach*, John Wiley & Sons, 1995.

[7] R.R. Macala, L.D. Stuckey, D.C. Gross, 'Managing Domain-Specific Product-Line Development,' IEEE Software, pp. 57-67, 1996.

[8] M. D. McIlroy, 'Mass Produced Software Components,' in 'Software Engineering,' *Report on A Conference Sponsored by the NATO Science Committee*, P. Naur, B. Randell (eds.), Garmisch, Germany, 7th to 11th October, 1968, NATO Science Committee, 1969.

[9] R. van Ommering, 'Building Product Populations with Software Components,' *Proceedings of ICSE 2002* (to appear), 2002.

[10] D.L. Parnas, 'On the Design and Development of Program Families', *IEEE Transactions on Software Engineering*, Vol. SE-2, No. 1, March 1976.

[11] 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.