Software Product Lines: Organizational Alternatives

Jan Bosch
University of Groningen
Department of Computing Science
PO Box 800, NL9700 AV Groningen, The Netherlands
+31 50 363 3941
Jan.Bosch@cs.rug.nl http://www.cs.rug.nl/~bosch

ABSTRACT

Software product lines enjoy increasingly wide adoption in the software industry. Most authors focus on the technical and process aspects and assume an organizational model consisting of a domain engineering unit and several application engineering units. In our cooperation with several software development organizations applying software product line principles, we have identified several other organizational models that are employed as well. In this article, we present a number of organizational alternatives, organized around four main models, i.e. development department, business units, domain engineering unit and hierarchical domain engineering units. For each model, its characteristics, applicability and advantages and disadvantages are discussed, as well as an example. Based on an analysis of these models, we present three factors that influence the choice of the organizational model, i.e. product-line assets, the responsibility levels and the type of organizational units.

1 INTRODUCTION

Achieving reuse of software has been a long standing ambition of the software engineering industry. Every since the paper by McIlroy [8], the notion of constructing software systems by composing software components has pursued in various ways. Most proposals to achieving component-based software development assume a market divided into component developers, component users and a market place. However, this proved to be overly ambitious for most types of software. In response, there has been a shift from *world-wide* reuse of components to *organization-wide* reuse. Parallel to this development, the importance of an explicit design and representation of the architecture of a software system has become increasingly recognized. The combination of these two insights lead to the definition of software product lines. A software product line consists of a product line architecture, a set of reusable components and a set of products derived from the shared assets.

Existing literature on software product lines [1,5,6,7] tends to focus on the technology and the processes that surround product line based software development. These processes include the design of the software architecture for the product line, the development of the shared software components, the derivation of software products and the evolution of the aforementioned assets. However, generally the organizational structure of software development organizations that is needed for the successful execution of these processes is not discussed. It is, nevertheless, necessary to impose an organization on the individuals that are involved in the product line.

In this article, we discuss four primary organizational models that can be applied when adopting a product line based approach to software development. For each model, we describe in what situations the model is most applicable, the advantages and disadvantages of the model and an example of a company that employs the model. In addition we discuss some factors that influence the choice of the most appropriate organizational model. Based on an analysis of the presented models, we have identified three dimensions that are organizationally relevant, i.e. the assets that are considered product line wide, the levels of responsibility employed in the organisation and the nature of the organizational units. These dimensions are used to present a space of organizational alternatives.

The contribution of this paper is that it identifies and categorizes the organizational alternatives for companies employing software product lines. It extends considerably over existing proposals for software product line organization that assume a division in domain and application engineering units

The remainder of this paper is organized as follows. In section 2 until 5, four organizational models are discussed in more detail. Section 6 discusses four influencing factors and their effects on selecting the optimal organizational model. In section 7, we discuss the dimensions that can be used to describe a space of organizational alternatives. Finally, related work is discussed in section 8 and the article is concluded in section 9.

2 DEVELOPMENT DEPARTMENT

The development department model imposes no permanent organizational structure on the architects and engineers that are involved in the software product line. All staff members

0-7695-1050-7/01 $10.00 © 2001 IEEE

can, in principle, be assigned to work with any type of asset within the family. Typically, work is organized in projects that dynamically organize staff members in temporary networks. These projects can be categorized into domain engineering projects and application (or system) engineering projects. In the former, the goal of the project is the development of a new reusable asset or a new version of it, e.g. a software component. The goal is explicitly not a system or product that can be delivered to internal or external customers of the development department. The system engineering projects are concerned with developing a system, either a new or a new version, that can be delivered to a customer. Occasionally, extensions to the reusable assets are required to fulfil the system requirements that are more generally applicable than just the system under development. In that case, the result of the system engineering project may be a new version of one or more of the reusable assets, in addition to the deliverable system.

In figure 1, the development department model is presented graphically. Both the reusable product line assets and the concrete systems built based on these assets are developed and maintained by a single organizational unit.

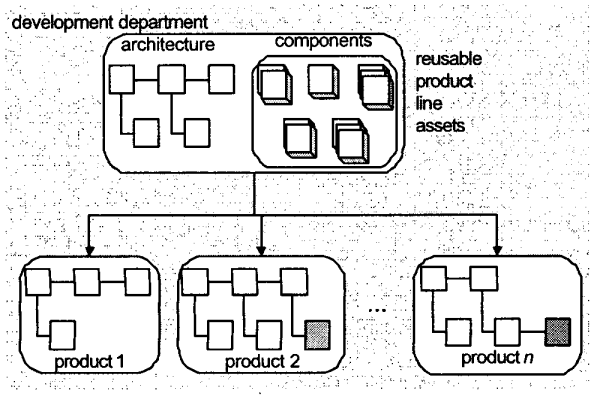

Figure 1: Development department model

2.1 Applicability

The development department model is primarily applicable for relatively small organizations and for consultancy organizations, i.e. organizations that sell projects rather than products to their customers. Based on our experience, our impression is that this model works up to around 30 software-related staff members in product-based organizations. If the number of staff members exceeds 30, generally some kind of organizational restructuring is required anyhow, independent of the use of a product line.

2.2 Advantages and disadvantages

The development department model has, as most things in life, a number of advantages and disadvantages. The primary advantage is simplicity and ease of communication. Since all staff members are working within the same organizational context, come in contact with all parts of the system family and have contact with the customers, the product line can be developed and evolved in a very efficient manner with little organizational and administrative overhead. A second advantage is that, assuming that a positive attitude towards reuse-based software development exists within the department, it is possible to adopt a software product line approach without changing the existing organization, which may simplify the adoption process.

The primary disadvantage of this approach is that it is not scalable. When the organization expands and reaches, e.g., around 30 staff members, it is necessary to reorganize and to create specialized units. A second disadvantage is that typically within organizations, staff members are, depending on the local culture, more interested in either domain engineering or system engineering, i.e. it has higher status in the informal organization to work with a particular type of engineering. The danger is that the lower status type of engineering is not performed appropriately. This may lead to highly general and flexible reusable components, but systems that do not fulfil the required quality levels, or visa versa.

2.3 Example

A company that employed this organizational model is Securitas Larm, Sweden. All their product development, i.e. hardware and software, is concentrated in a single development department. This department maintains a product line in the domain of fire-alarm systems, as we describe in [2]. The department has an engineering staff about 25 persons, so it fits our applicability requirement. In fact, up to a number of years ago, development was organized in product business units. Each product unit was responsible for sales, marketing, installation and development of the product. However, especially development did not function well in this organizational form. Generally only up to five engineers worked with the product development, which was too few to create an effective development organization. Consequently, Securitas Larm decided to reorganize development into a single development department.

3 BUSINESS UNITS

The second organizational model that we discuss is organized around business units. Each business unit is responsible for the development and evolution of one or a few products in the software product line. The reusable assets in the product line are shared by the business units. The evolution of shared assets is generally performed in a distributed manner, i.e. each business unit can extend the functionality in the shared assets, test it and make the newer version available to the other business units. The initial development of shared assets is generally performed through domain engineering projects. The project team consists of members from all or most business units. Generally, the business units most interested in the creation of, e.g. a new software component, put the

largest amount of effort in the domain engineering project, but all business units share, in principle, the responsibility for all common assets.

Depending on the number and size of the business units and the ratio of shared versus system specific functionality in each system, we have identified three levels of maturity, especially with respect to the evolution of the shared assets:

Unconstrained model. In the unconstrained model, any business unit can extend the functionality of any shared component and make it available as a new version in the shared asset base. The business unit that performed the extension is also responsible for verifying that, where relevant, all existing functionality is untouched and that the new functionality performs according to specification.

A typical problem that companies using this model suffer from is that, typically software components, are extended with too system-specific functionality. Either the functionality has not been generalized sufficiently or the functionality should have been implemented as system-specific code, but for internal reasons, e.g. implementation efficiency or system performance, the business unit decided to implement the functionality as part of the shared component.

These problems normally lead to the erosion or degradation of the component, i.e. it becomes, over time, harder and less cost-effective to use the shared component, rather than developing a system-specific version of the functionality. As we discussed in [2], some companies have performed component reengineering projects in which a team consisting of members from the business units using the component, reengineers the component and improves its quality attributes to acceptable levels. Failure to reengineer when necessary may lead to the situation where the product line exists on paper, but where the business units develop and maintain system-specific versions of all or most components in the product line, which invalidates all advantages of a software product line approach, while maintaining some of the disadvantages.

Asset responsibles. Especially when the problems discussed above manifest themselves in increasing frequency and severity, the first step to address these problems is to introduce asset responsibles. An asset responsible has the obligation to verify that the evolution of the asset is performed according to the best interest of the organization as a whole, rather than optimal from the perspective of a single business unit. The asset responsible is explicitly not responsible for the implementation of new requirements. This task is still performed by the business unit that requires the additional functionality. However, all evolution should occur with the asset responsible's consent and before the new version of the asset is made generally accessible, the asset responsible will verify through regression testing and other means that the other business units are at least not negatively affected by the evolution. Preferably, new requirements are implemented in such a fashion that even other business units can benefit from them. The asset responsible is often selected from the business unit that makes most extensive and advanced use of the component.

Although the asset responsible model, in theory at least, should avoid the problems associated with the unconstrained model, in practice it often remains hard for the asset responsible to control the evolution. One reason is that time-to-market requirements for business units often are prioritized by higher management, which may force the asset responsible to accept extensions and changes that do not fulfil the goals, e.g. too system-specific. A second reason is that, since the asset responsible does not perform the evolution him or herself, it is not always trivial to verify that the new requirements were implemented as agreed upon with the business unit. The result of this is that components still erode over time, although generally at a lower pace than with the unconstrained model.

Mixed responsibility. Often, with increasing size of the system family, number of staff and business units, some point is reached where the organization still is unwilling to adopt the next model, i.e. domain engineering units, but wants to assign the responsibility for performing the evolution assets to a particular unit. In that case, the mixed responsibility model may be applied. In this model, each business unit is assigned the responsibility for one or more assets, in addition to the product(s) the unit is responsible for. The responsibility for a particular asset is generally assigned to the business unit that makes the most extensive and advanced use of the component. Consequently, most requests for changes and extensions will originate from within the business unit, which simplifies the management of asset evolution. The other business units have, in this model, no longer the authority to implement changes in the shared component. Instead, they need to issue requests to the business unit responsible for the component whenever an extension or change is required.

The main advantage of this approach is the increased control over the evolution process. However, two potential disadvantages exist. First, since the responsibility for implementing changes in the shared asset is not always located at the business unit that needs those changes, there are bound to be delays in the development of systems that could have been avoided in the approaches described earlier. Second, each business unit has to divide its efforts between developing the next version of its product(s) and the evolution of the component(s) it is responsible for. Especially when other business units have change requests, these may conflict with the ongoing activities within the business unit and the unit may prioritize its own goals over the goals of other business units. In addition, the business unit may extend the components it is responsible for in ways that are optimized for its own purposes, rather than for the

organization as a whole. These developments may lead to conflicts between the business units and, in the worst case, the abolishment of the product line approach.

Conflicts. The way the software product line came into existence is, in our experience, an important factor in the success or failure of a family. If the business units already exist and develop their systems independently and, at some point, the software product line approach is adopted because of management decisions, conflicts between the business units are rather likely because giving up freedom that one had up to that point in time is generally hard. If the business units exist, but the product line gradually evolves because of bottom-up, informal cooperation between staff in different business units, this is an excellent ground to build a product line upon. However, the danger exist that when cooperation is changed from optional to obligatory, tensions and conflicts appear anyhow. Finally, in some companies, business units appear through an organic growth of the company. When expanding the set of systems developed and maintained by the company, at some point, a reorganization into business units is necessary. However, since the staff in those units earlier worked together and used the same assets, both the product line and cooperation over business units develop naturally and this culture often remains present long after the reorganization, especially when it is nurtured by management. Finally, conflicts and tensions between business units must resolved by management early and proactively since they imply considerable risk for the success of the product line.

In figure 2, the business unit model is presented graphically. The reusable system-family assets are shared by the business units, both with respect to use as well as to evolution.

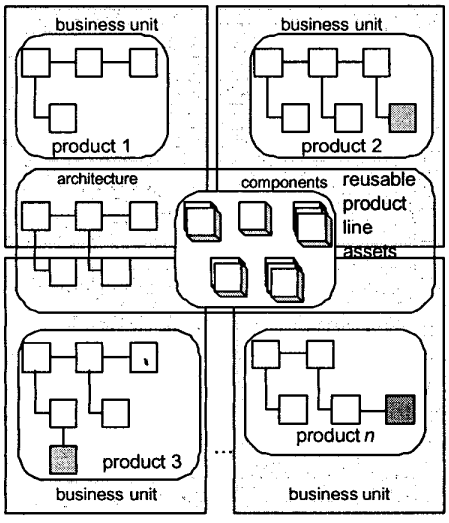

Figure 2: Business unit model

3.1 Applicability

As discussed in section 2, when the number of staff members is too low, e.g. below 30, the organization in business units is often not optimal since too few people are working together and the communication overhead over unit boundaries is too large. On the other hand, our hypothesis, based on a number of cases that we have studied, is that when the number of staff members exceeds 100, domain engineering units may become necessary to reduce the n-to-n communication between all business units to a one-to-n communication between the domain engineering unit and the system engineering units. Thus, with respect to staff size, we believe that the optimal range for the business unit model is between 30 and 100, although this, to a large extent, depends on the specific context as well.

3.2 Advantages and disadvantages

The advantage of this model is that it allows for effective sharing of assets, i.e. software architectures and components, between a number of organizational units. The sharing is effective in terms of access to the assets, but in particular the evolution of assets (especially true for the unconstrained and the asset responsible approaches). In addition, the approach scales considerably better than the development department model, e.g. up to 100 engineers in the general case.

The main disadvantage is that, due to the natural focus of the business units on systems (or products), there is no entity or explicit incentive to focus on the shared assets. This is the underlying cause for the erosion of the architecture and components in the system family. The timely and reliable evolution of the shared assets relies on the organizational culture and the commitment and responsibility felt by the individuals working with the assets.

3.3 Example

Axis Communications, Sweden, employs the business unit model. Their storage-server, scanner-server and camera-server products are developed by three business units. These business units share a common product line architecture and a set of more than ten object-oriented frameworks that may be extended with system-specific code where needed. Initially, Axis used the unconstrained model with relatively informal asset responsibles, but recently the role of asset responsibles has been formalized and they now have the right to refuse new versions of assets that do not fulfil generality, quality and compatibility requirements. The assets responsibles are taken from the business units that make the most extensive and advanced use of the associated assets. Within the organization, discussions were ongoing whether an independent domain engineering unit, alternatively, a mixed responsibility approach would be needed to guarantee the proper evolution of assets. Whenever new assets or a major redesign of some existing asset is needed, Axis has used domain engineering projects, but 'disguised' these projects as system engineering projects by developing prototype

systems. The advantage of the latter is that the integration of the new asset with the existing assets is automatically verified as part of the domain engineering project.

4 DOMAIN ENGINEERING UNIT

The third organizational model for software product lines is concerned with separating the development and evolution of shared assets from the development of concrete systems. The former is performed by a, so-called, domain engineering unit, whereas the latter is performed by product engineering units. System engineering units are sometimes referred to as application or system engineering units.

The domain engineering unit model is typically applicable for larger organizations, but requires considerable amounts of communication between the product engineering units, that are in frequent contact with the users of their products, and the domain engineering unit that has no direct contact with customers, but needs a good understanding of the requirements that the product engineering units have. Thus, one can identify flows in two directions, i.e. the requirements flow from the product engineering units towards the domain engineering unit and the new versions of assets, i.e. the software architecture and the components of system family, are distributed by the domain engineering unit to the product engineering units.

The domain engineering unit model exists in two alternatives, i.e. an approach where only a single domain engineering unit exists and, secondly, an approach where multiple domain engineering units exist. In the first case, the responsibility for the development and evolution of all shared assets, i.e. the software architecture and the components, is assigned to a single organizational unit. This unit is the sole contact point for the product engineering units that construct their products based on the shared assets.

The second alternative employs multiple domain engineering units, i.e. one unit responsible for the design and evolution of the software architecture for the product line and, for each architectural component (or set of related components), a component engineering unit that manages the design and evolution of the components. Finally, the product engineering units are, also in this alternative, concerned with the development of products based on the assets. The main difference between the first and second alternative is that in the latter, the level of specialization is even higher and that product engineering units need to interact with multiple domain engineering units.

In figure 3, the organizational model for using domain engineering unit is presented. The domain engineering unit is responsible for the software architecture and components of the product line, whereas the system engineering units are responsible for developing the systems based on the shared assets.

4.1 Applicability

Especially smaller companies are very sceptical of domain engineering units. One of the concerns is that, just because domain engineering units are concerned with reusable assets, rather than products that are relevant for users, these units may not be as focused on generating added value, but rather lose themselves in aesthetic, generic, but useless abstractions. However, based on our experience, our impression is that when the number of staff members working within a product family exceeds around 100 software engineers, the amount of overhead in the communication between the business units causes a need for an organizational unit or units specialized on domain engineering.

Multiple rather than a single domain engineering unit becomes necessary when the size of the domain engineering unit becomes too large, e.g. 30 software engineers. In that case, it becomes necessary to create multiple groups that focus on different component sets within a product line software architecture. In some cases, although component engineering units exist, no explicit product line architecture unit is present. Rather, a small team of software architects from the component engineering units assumes the responsibility for the overall architecture.

Finally, at which point the complexities of software development even exceed the domain engineering unit approach is not obvious, but when the number of software engineers is in the hundreds, the hierarchical domain engineering units model, discussed in the next section, may become necessary.

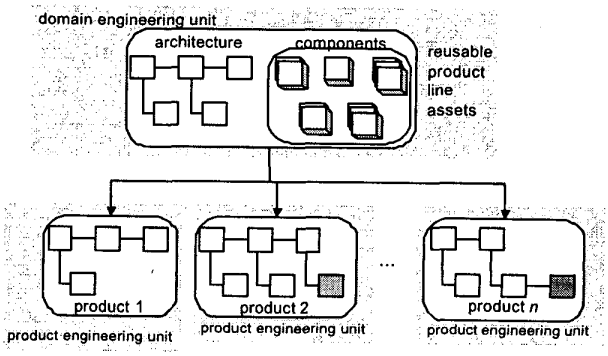

Figure 3: The domain engineering unit model

4.2 Advantages and disadvantages

Despite the scepticism in, especially smaller organizations, the domain engineering unit model has a number of important advantages. First, as mentioned, it removes the need for n-to-n communication between the business units, and reduces it to one-to-n communication. Second, whereas business units may extend components with too product-specific extensions, the domain engineering unit is responsible for evolving the components such that the requirements of all systems in the product line are satisfied. In addition, conflicts

can be resolved in a more objective and compromise-oriented fashion. Finally, the domain engineering unit approach scales up to much larger numbers of software engineering staff than the aforementioned approaches.

Obviously, the model has some associated disadvantages as well. The foremost is the difficulty of managing the requirements flow towards the domain engineering unit, the balancing of conflicting requirements from different product engineering units and the subsequent implementation of the selected requirements in the next version of the assets. This causes delays in the implementation of new features in the shared assets, which, in turn, delays the time-to-market of products. This may be a major disadvantage of the domain engineering unit model since time-to-market is the primary goal of many software development organizations. To address this, the organization may allow product engineering units to, at least temporarily, create their own versions of shared assets by extending the existing version with product-specific features. This allows the product engineering unit to improve its time-to-market while it does not expose the other product engineering units to immature and instable components. The intention is generally to incorporate the product-specific extensions, in a generalized form, into the next shared version of the component.

4.3 Example

The domain engineering unit model is used by Symbian. The EPOC operating system consists of a set of components and the responsibility of a number of subsets is assigned to specialized organizational units. For each device family requirement definition (DFRD), a unit exists that composes and integrates versions of these components into a release of the complete EPOC operating system to the partners of Symbian. The release contains specific versions and instantiations of the various components for the particular DFRD. Some components are only included in one or a few of the DFRDs.

5 HIERARCHICAL DOMAIN ENGINEERING UNITS

As we discussed in the previous section, there is an upper boundary on the size of an effective domain engineering unit model. However, generally even before the maximum staff member size is reached, often already for technical reasons, an additional level has been introduced in the software product line. This additional layer contains one or more specialized product lines that, depending on their size and complexity can either be managed using the business unit model or may actually require a domain engineering unit.

In the case that a specialized product line requires a domain engineering unit, we have, in fact, instantiated the hierarchical domain engineering units model that is the topic of this section. This model is only suitable for a large or very large organization that has an extensive family of products. If, during the design or evolution of the product line, it becomes necessary to organize the product line in a hierarchical manner and a considerable number of staff members is involved in the product line, then it may be necessary to create specialized domain engineering units that develop and evolve the reusable assets for a subset of the products in the family.

The reusable product line assets at the top level are frequently referred to as a platform and not necessarily identified as part of the product line. We believe, however, that it is relevant to explicitly identify and benefit from the hierarchical nature of these assets. Traditionally, platforms are considered as means to provide shared functionality, but without imposing any architectural constraints. In practice, however, a platform does impose constraints and when considering the platform as the top-level product line asset set, this is made more explicit and the designers of specialized product lines and family members will derive from the software architecture rather than design it.

In figure 4, the hierarchical domain engineering units model is presented graphically. For a subset of the systems in the product line, a domain engineering unit is present that develops and maintains the specialized product line software architecture and the associated components. Only the components specific for the subset in the product line are the responsibility of the specialized domain engineering unit. All other components are inherited from the overall product line asset base. The specialized domain engineering unit is also responsible for integrating the specialized with the general reusable assets.

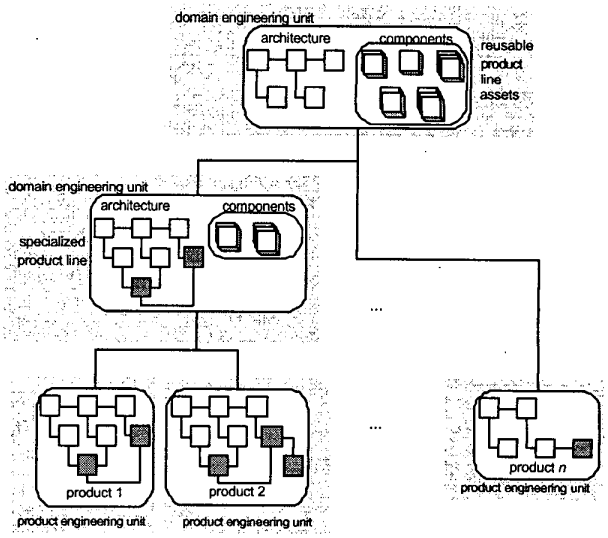

Figure 4: Hierarchical domain engineering unit model

5.1 Applicability

As mentioned in the introduction, the hierarchical domain units model becomes the preferred model when the number and variability of systems in the family is large or very large and considerable numbers of staff members, i.e. hundreds,

are involved. Consequently, the model is primarily suitable in large organizations and long-lived systems in the family, since the effort and expenses involved in building up this organizational model are substantial.

The complexities involved in the implementation and use of this organizational model are beyond the scope of this article, but a considerable maturity with respect to software development projects is required for this approach to succeed. This model is the fourth and most complex model that we discuss and if the product line cannot be captured within this model, it is reasonable to assume that the scope of the family has been set too wide.

5.2 Advantages and disadvantages

The advantages of this model include its ability to encompass large, complex product lines and organize large numbers of engineers. None of the organizational models discussed earlier scales up to the hundreds of software engineers that can be organized using this model.

The disadvantages include the considerable overhead that the approach implies and the difficulty of achieving agile reactions to changed market requirements. Typically, a delicate balance needs to be found between allowing product engineering units to act independent, including the temporary creation of product-specific versions of product line components, versus capitalizing on the commonalities between products and requiring product engineering units to use shared versions of components.

5.3 Example

An example of an organization that has successfully adopted the hierarchical domain engineering units model is Nokia Mobile Phones. This company develops and maintains a wide variety of products in the wireless information devices domain, in particular mobile phones. The company has applied a product line approach to its mobile phone development for several years. The software product line consists of two levels. The top level, i.e. a 'platform', is developed and maintained by a top-level 'infrastructure' group, and consists of a product line architecture and a set of components, that are shared by all mobile phone products and ported to different hardware platforms. For subsets of products in the product family, specialized groups exist that develop, especially, components specific for the family members in the subset. These domain engineering units have frequent contact and exchange considerable amounts of information, but are organized as independent units.

6 INFLUENCING FACTORS

Up to this point, we have presented the size of the product line and the engineering staff involved in the development and evolution of the product line as the primary factors in selecting the appropriate organizational model. Although, in our experience, the above factors indeed are the most prominent, several factors exist that should be allowed to influence the selection decision as well. Below, we present some factors that we have identified in industry as relevant in this context.

6.1 Geographical distribution

Despite the emergence of a variety of technological solutions aiming at reducing the effects of geographical location, e.g. telephone, e-mail, video conferencing and distributed document management, the physical location of the staff involved in the software product line still plays a role. It simply is more difficult to maintain effective and efficient communication channels between teams that are in disparate locations and, perhaps even, time zones, than between teams that are co-located. Therefore, units that need to exchange much information should preferably be located closer to each other than units that can cooperate with less information.

For instance, geographical distribution of the teams developing the systems in the family may cause a company to select the domain engineering unit model because it focuses the communication between the domain engineering unit and each product engineering unit, rather than the n-to-n communication required when using the business unit model.

6.2 Project management maturity

The complexity of managing projects grows exponentially with the size of the project (in virtually any measure). Therefore, the introduction of a software product line approach requires, independent of the organizational model, a relatively high level of maturity with respect to project management. Projects need to be synchronized over organizational boundaries and activities in different projects may be depending on each other, which requires experience and pro-activeness in project management.

To give an example, incorporating new functionality in a product line component at Axis Communications requires communication with the other business units at the start, the actual execution and at the end of the project. At the start because it should be verified that no other business unit is currently including the same or related functionality. During the project, to verify that the included functionality and the way in which it is implemented are sufficiently general and provide as much benefit as possible to the other business units. After the end of the project, to verify that the new version of the component provides backward compatibility to systems developed by the other business units.

6.3 Organizational culture

The culture of an organization is often considered to be a hard to use concept, which is obviously the case. However, the attitude that each engineer has towards the tasks that he or she is assigned to do and the value patterns exhibited by the informal organizational groups have a major influence on the final outcome of any project. Thus, if a kind of 'cowboy' or 'hero' culture exists in which individual achievements are valued higher than group achievements, then this attitude can

prove to be a serious inhibitor of a successful software product line approach that is highly dependent on a team culture that supports interdependency, trust and compromise.

For instance, at one company, which will remain unnamed, we discussed the introduction of a software product line approach. The company had extensive experience in the use of object-oriented frameworks and within each business unit reuse was wide-spread and accepted. However, when top management tried to implement product line based reuse, business unit managers revolted and the initiative was cancelled. The reason, it turned out, was that each business unit would have to sacrifice its lead architect(s) for a considerable amount of time during the development of the reusable product line assets. In addition, the conversion would delay several ongoing and planned projects. These two effects of adopting a product line approach would, among others, lead to highly negative effects on the bonuses received by, especially, business unit management. One explanation could be that these managers were selfish people that did not consider what was best for the company as a whole. However, our explanation is that top management had, under many years, created a culture in which business units were highly independent profit centres. This culture conflicted directly with the product line approach top management tried to introduce.

6.4 Type of systems

Finally, an important factor influencing the optimal organizational model, but also the scope and nature of the system family, is the type of systems that make up the family. Systems whose requirements change frequently and drastically, e.g. due to new technological possibilities, are substantially less suitable for large up-front investments that a wide scoped, hierarchical software product line approach may require, than systems with relatively stable requirement sets and long lifetimes. Medical and telecommunication (server-side) systems are typical systems that have reasonably well understood functionality and that need to be maintained for at least a decade and often considerably longer.

For instance, consultancy companies that typically are project based are able to adopt a software product line approach. Since subsequent projects often are in the same domain, the availability of a product line architecture and a set of reusable components may substantially reduce lead time and development cost. However, the investment taken by such a company to develop these assets can never be in the same order of magnitude as a product-based company with clear market predictions for new products. The consultancy company has a significantly higher risk that future projects are not in exactly the same domain, but an adjacent, invalidating or at least reducing the usefulness of the developed assets. Consequently, investment and risk always need to be balanced appropriately.

7 ORGANIZATIONAL DIMENSIONS

Once we had identified the four organizational models and their variants, we performed an analysis of their characteristics. Based on this analysis, we have identified three dimensions that play role in the selection of the most appropriate organization for software product line based development. These dimensions are: the assets that are considered product line wide, the levels of responsibility employed in the organisation and the nature of the organizational units. When combined, these dimensions form a space of organizational alternatives. Below, each dimension is discussed in more detail. The section ends with a discussion of the relation between the dimensions and the discussed organizational models.

7.1 Product line assets

A software product line consists of a number of assets, i.e. the product line architecture, the product line components and the product specific software. In the traditional organizational model, the product line architecture and shared components are the responsibility of the domain engineering units whereas the product specific software is the responsibility of the application engineering units.

In our experience, the way assets are treated depends on the type of products and the type of organization employing a product line approach. Therefore, the assets considered by the organization as a whole may vary from just the product line architecture to all assets including the product specific code. Below, we describe four levels:

- **Architecture**: In organizations with little integration between the various units, a first step towards achieving a software product line may be to synchronise around a common architecture for the products where possible. In this case, only the architecture is the shared asset.
- **Platform**: Once a shared architecture is a place, it becomes possible to define some basic functionality as shared components. Typically, this functionality is shared by all products. In this way, the benefits of increased integration between different units are obvious and no or few disadvantages are present. Often, this type of sharing between product units is referred to as using a platform.
- **Components**: As in traditional software product lines, the next level is to share both the product line architecture and most of the software components that are shared among two or more products. At this point, typically some products experience disadvantages of using the product line. This requires the organization to explicitly stress the overall advantages which, obviously, should outweigh the local disadvantages.
- **Product specifics**: The highest level of integration is achieved when product specific code is explicitly considered at the product line level. The reason for focusing on

product specific code is typically that much of this code, at some point in the future, is used in other products as well. Thus, by designing and developing product specific code carefully, future integration in the shared product line assets is made considerably easier.

7.2 Responsibility levels

A second dimension is the way responsibility for product line assets is handled. Again, traditionally a clear division between domain engineering and application engineering units exists. However, in many cases the actual assignment of responsibility is more fine grained. We have identified three levels of responsibility within organizations applying software product lines. Below these levels are discussed:

- **Shared**: Especially in organizations where a product line appeared in a bottom-up manner, the responsibility for the shared product line assets may, at least initially, be shared among the organizational units.
- **Responsible**: At some point, typically an individual or a small team is assigned at the responsible for the particular asset. However, this does not mean that the evolution of the asset is performed by the responsible. The responsibility of the person or team is limited to ensuring that changes do not violate the requirements of other users or decrease future modifiability.
- **Engineered**: The highest level of responsibility is achieved when a team is responsible for the development and evolution of a particular asset. This requires users of the asset to request changes from the team. The team has the responsibility to respond to change requests in a fair and timely manner.

7.3 Organizational units

The third dimension that we have identified as relevant for the organization of software product line based development is the way staff is organized into units. Our cooperation with software industry has learned us that one of four perspectives can be taken. Below, these perspectives are discussed.

- **Project**: The first model does not employ a permanent assignment of staff to units. This leads to a project-based type for organization, where staff is assigned to teams for the duration of projects.
- **Product**: The second model is to organize staff around products. Because staff is permanently assigned to a particular product, their experience leads to increased levels of efficiency. However, this experience is not shared between different products.
- **Shared components**: When considerable overlap between different products is identified, the company may decide to exploit this by identifying and defining a number of components that are assigned to units that act as service providers to the product units.
- **Architecture centric**: The final model for organizing staff centers around a software architecture that is shared among the products. This architecture defines the shared components and the product specific code and, consequently, the organizational units. Different from the previous model where the product units are in control, in this model typically the software architecture and associated staff control development.

7.4 Organizational Alternatives

The three dimensions discussed above define a three-dimensional space that can be used to categorize organizational approaches to product-line development. In section 2 through 5, four organizational models to software product line based development were discussed. To illustrate the relation between the dimensions and the organizational models, figure 5 presents this graphically. Development department and domain engineering units are presented by a single line, whereas business units and hierarchical domain engineering form planes. The latter is due to the fact that alternatives exist for these models.

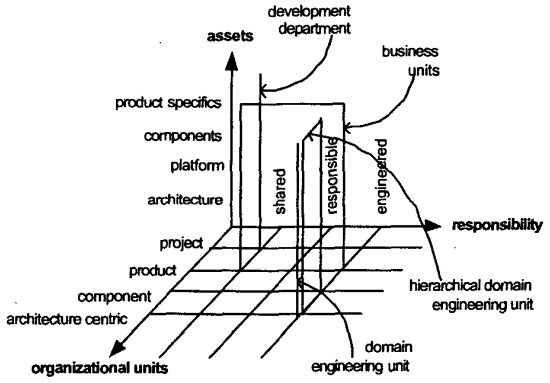

Figure 5: Dimensions organizing software product lines

What is clear from this 3D space is that several alternatives exist that we have not discussed in the earlier sections. However, this does not mean that these are no viable alternatives. We are convinced that organizations exist that employ these alternative models. When adopting a software product line, it is advisable to understand what alternatives are available and to evaluate these, rather than blindly adopting a standard model.

8 RELATED WORK

As we discussed in the introduction, most publications in the domain of software product lines address issues different from the organizational ones. Macala et al. [7] and Dikel et al. [5] were among the first publications that describe experiences from using software product lines in an industrial context. Although the authors do address organizational, management and staffing issues, both assume the domain engineering unit model and present it as the de-facto

organizational model. Jacobsen et al. [6] also discuss organizational issues, but focus on a number of roles that should be present and do not address the overall organization of software product line based development. In Clements and Northrop [4], the authors address organizational issues of software product line. The authors identify four functional groups, i.e. the architecture group, the component engineering group, the product line support group and the product development group. The authors identify that these functional groups may be mapped to organizational units in various ways. Finally, Bayer et al. [1] discuss a methodology for developing software product lines and discuss organizational guidelines, but no organizational models.

9 CONCLUSION

In this article, we have discussed four organizational models for software product lines and discussed, based on our experiences, the applicability of the model, the advantages and disadvantages and an example of an organization that employs the particular model. Below, the four models are briefly summarized:

Development department. In this model software development is concentrated in a single development department, no organizational specialization exists with either the software product line assets or the systems in the family. The model is especially suitable for smaller organizations. The primary advantages are that it is simple and communication between staff members is easy, whereas the disadvantage is that the model does not scale to larger organizations.

Business units. The second type of organizational model employs a specialization around the type of systems in the form of business units. Three alternatives exist, i.e. the unconstrained model, the asset responsibles model and the mixed responsibility model. Some of our industrial partners have successfully applied this model up to 100 software engineers. An advantage of the model is that it allows for effective sharing of assets between a set of organizational units. A disadvantage is that business units easily focus on the concrete systems rather than on the reusable assets.

Domain engineering unit. In this model, the domain engineering unit is responsible for the design, development and evolution of the reusable assets. Product engineering units are responsible for developing and evolving the products built based on the product line assets. The model is widely scalable, from the boundaries where the business unit model reduces effectiveness up to hundreds of software engineers. Another advantage of this model is that it reduces communication from n-to-n in the business unit model to one-to-n between the domain engineering unit and the system engineering units. Finally, the domain engineering unit focuses on developing general, reusable assets which addresses one of the problems with the aforementioned model, i.e. too little focus on the reusable assets. One disadvantage is the difficulty of managing the requirements flow and the evolution of reusable assets in response to these new requirements. Since the domain engineering unit needs to balance the requirements of all system engineering units, this may negatively affect time-to-market for individual system engineering units.

Hierarchical domain engineering units. In cases where an hierarchical product line has been necessary, also a hierarchy of domain units may be required. The domain engineering units that work with specialized product lines use the top-level assets as a basis to found their own product line upon. This model is applicable especially in large or very large organizations with a large variety of long-lived systems. The advantage of this model is that it provides an organizational model for effectively organizing large numbers of software engineers. One disadvantage is the administrative overhead that easily builds up, reducing the agility of the organization as a whole, which may affect competitiveness negatively.

We have discussed a number of factors that influence the organizational model that is optimal in a particular situation. These factors include geographical distribution, project management maturity, organizational culture and the type of systems.

Based on an analysis of the four organizational models, we have identified three dimensions that are organisationally relevant, i.e. the assets that are considered product line wide, the levels of responsibility employed in the organisation and the nature of the organizational units.

REFERENCES

1. J. Bayer, O. Flege, P. Knauber, R. Laqua, D. Muthig, K. Schmid, T. Widen, J.M. DeBaud, 'PuLSE: A Methodology to Develop Software Product Lines, *Symposium on Software Reuse*, 1999.
2. Jan Bosch, 'Product-Line Architectures in Industry: A Case Study', *Proceedings of the 21st International Conference on Software Engineering*, pp. 544-554, May 1999.
3. J. Bosch, *Design & Use of Software Architectures - Adopting and Evolving a Product-Line Approach*, Addison Wesley, ISBN 0-201-67494-7, 2000.
4. P. Clements, L. Northrop, 'A Framework for Software Product Line Practice - Version 1.0', *Software Engineering Institute, Carnegie Mellon*, September 1998.
5. D. Dikel, D. Kane, S. Ornburn, W. Loftus, J. Wilson, 'Applying Software Product-Line Architecture,' *IEEE Computer*, pp. 49-55, August 1997.
6. I. Jacobsen, M. Griss, P. Jönsson, *Software Reuse - Architecture, Process and Organization for Business Success*, Addison-Wesley, 1997.
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.