On the Development of Software Product Family Components

Jan Bosch

University of Groningen, Department of Computing Science,
PO Box 800, 9700 AV Groningen, The Netherlands.
Jan.Bosch@cs.rug.nl, http://segroup.cs.rug.nl

Several approaches to the development of shared artefacts in software product families exist. Each of these has advantages and disadvantages, but there is no clear framework for selecting among these alternatives. As a consequence, mismatches between the optimal approach and the one currently used by an organization may lead to several problems, such as a high degree of erosion, mismatches between product needs and shared components, organizational "noise" and inefficient knowledge management. This paper presents (1) the problems resulting from the aforementioned mismatch, (2) presents the relevant decision dimensions that define the space of alternatives, (3) discusses the advantages and disadvantages of each alternative and (4) presents a framework for selecting the best alternative for each decision dimension based on a three stage adoption model.

1. Introduction

Software product families have achieved a broad recognition in the software industry. Many organizations either have adopted or are considering to adopt the technology. One of the key items in software product families is the development, evolution and use of shared components. Being able to develop a component once and use it in several products or systems is, obviously, one of the main benefits to be achieved.

The sharing of components among multiple systems is a great concept in theory and has been quite successful in the context of inter-organizational reuse. Modern operating systems, database management systems, graphical user interfaces, component infrastructures, web servers, web browsers, etc. offer a rich infrastructure based on which products can be quickly and easily developed. Intra-organizational reuse of components, i.e. software product families, on the other hand, has experienced considerably more problems in achieving a high degree of component sharing. In many organizations, achieving effective and efficient reuse of product family components proofs, in many cases, to be not trivial at all.

In the literature, e.g. [Weiss & Lai 99] and [Clements & Northrop 01], as well as in our own publications, e.g. [Bosch 00], reasoning around the scoping of a shared software artefact in a product family is typically organized around the amount of sharing of features by the customers of the component team developing the shared component. However, although this provides an overview of the scoping of product

family components, it leaves many details to be implemented by the organization adopting software product families.

In our experience from several industrial cases, several decisions need to be taken about the scoping of product family artefacts, the way to organize for the development and evolution of these artefacts, the funding model used to finance the development of shared components, the features that are selected for implementation in shared components as well as the amount of architecture harmonisation required. However, these decisions are often taken implicitly rather than explicitly based on clearly stated objectives. The consequence is that the organization may experience a mismatch between the employed and the optimal approach. This may result in several problems, such as a high degree of erosion, mismatches between product needs and shared components, organizational "noise" and inefficient knowledge management.

The contribution of this paper is that we present a framework for deciding the optimal model for organizing the development of shared components. In particular, we present the dimensions that define the space of approaches and discuss the advantages and disadvantages. Subsequently, we present a framework for selecting the best alternative for each decision based on a three-stage adoption model. Although the paper is based on extensive industrial experience, we are unable to present any case studies due confidentiality and the lack of time for obtaining approval for publication.

The remainder of the paper is organized as follows. In the next section, we present the problems that an organization may experience when a mismatch between the chosen and optimal approach to the development of shared artefacts is chosen. Subsequently, in section 3 we discuss the dimensions for which the organization needs to take decisions when developing shared software artefacts. Section 4 presents the three stages of product family adoption and the decision framework for the development of shared components associated with each stage. Section 5 discusses related work and the paper is concluded in section 6.

2. Problem Statement

The key benefit of software product families results from the ability to share the implementation of components over multiple products, resulting in overall lower development and maintenance cost. Achieving these benefits in practice, however, proofs to be quite a challenge. Our claim in this paper is that this is, among others, caused by the mismatch between the approach used for the development of shared artefacts and the optimal approach. If such a mismatch is present, an organization may experience several problems. From the cases that we have been involved in, a common denominator for all mismatches is a shared perception among the R&D staff that "it doesn't feel right". This feeling is due to the staff experiencing too many mismatches between different roles in the organization in the process of product family centric software engineering. Below, we discuss the problems that, in our experience, are the most predominant.

- **Mismatch between shared components and product needs**: The first problem that an R&D organization may experience is that the staff experiences a mismatch

between the product requirements and the behaviour provide by the shared component. This mismatch may be of architectural nature, due to differences in assumptions about the architectural context in which the component operates. The mismatch may be concerned with interfaces provided and required by the shared component. A third reason may be concerned with the quality attributes provided by the shared component. Finally, the mismatch may be in time, i.e. between the roadmap and release schedule of the shared components and the product.

- **Design erosion of shared components**: Due to schedule or other pressures on the shared components, initial development or evolution of the shared component may be performed without sufficiently considering all aspects of the design, resulting in a high degree of design erosion. The design may be extended with too product specific functionality, additions are not sufficiently integrated in the original design or variability requirements need to be integrated that have a cross-cutting effect. The high level of design erosion increases the maintenance cost, complicates component integration in products and causes early retirement of the component.
- **Complex interface**: Delivering a component that is to be extended with product specific code, which is one approach to developing shared components, requires the availability of some kind of interface to the internals of the component. Although ideally this interface is clean and reveals minimal details of the internals of the component, in practice this is very difficult to achieve in the case where the component is extended with product specific features. This is, among others, due to the fact that features tend to be cross-cutting in nature with respect to the basic elements of software development.
- **High degree of "organizational noise"**: During an adoption process, many staff members are affected in their role and responsibilities. The organization has to learn to operate in the new role, but this is generally not achieved without problems. These problems lead to a certain amount of "organizational noise", i.e. complaints and discussions, especially in the informal organization. A limited amount of organizational noise over a limited amount of time during the early phases of adoption is normal, but if there is much "noise" over an extended period, this is a clear indicator that the approach towards the development and evolution of shared components does not fulfil the organization's needs. This may result in problems associated with productivity and predictability of schedule, but also affects softer factors, such as motivation, team spirit and the momentum for the product family initiative.
- **Inefficient knowledge management**: In the case where a shared component is delivered to several products where most product teams need to extend the component with features specific to their product, this means that, in addition to the staff in the component team, at least one person in each product team needs to maintain a sufficient, typically high, level of understanding with respect to the internals of the component. The cost of maintaining this level of expertise is an often underestimated factor and detriments the benefits of the product family approach.
- **Evolution causes ripple effects through the R&D organization**: As all software, also the shared component will evolve with, typically, a certain heartbeat. Whereas it often is feasible to maintain the existing provided interface, experience shows that it often is more difficult to maintain the extension interface. This means that

for most new releases, all product engineering units that extend the component need to re-evaluate and evolve their extensions in order to work with the new interface. Depending on the complexity of the changes, this may be quite effort consuming. In addition, the amount of effort required is often difficult to predict before the component is available, creating difficulty in the planning of software development in the unit.

- **Component value not linear**: An assumption rather wide-spread in the software industry is that a shared component that fulfils X% of the features required from it in a product context also represents X% of the value of a product-specific component. Based on our experience, we conclude that this relation is not linear, but rather exponential. Due to the problems discussed above, the value of a shared component increases exponentially with its support of product specific requirements. In the figure below, this is illustrated graphically. The figure also shows that a component that satisfies substantially more than the product requires also is less valuable, due to higher resource demands and more complex interfaces.

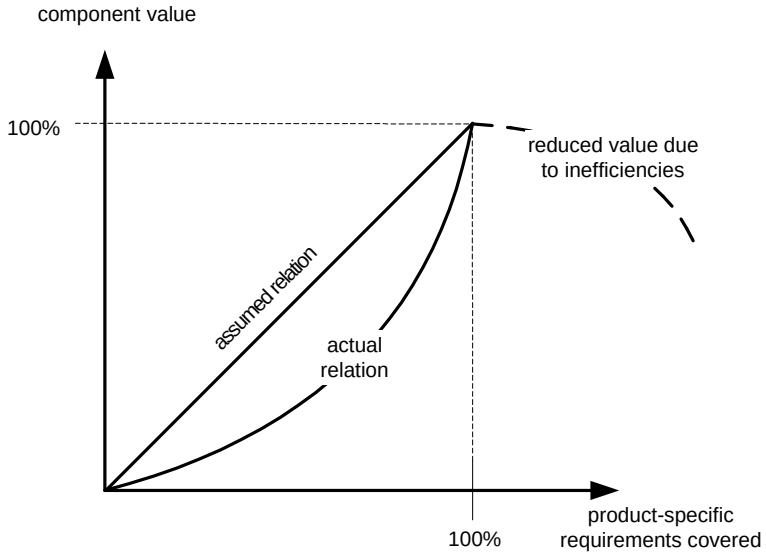

Figure 1. Relation between component value and support for product specific requirements

3. Decision Dimensions for Shared Component Development

Organizations adopting or employing software product families need to decide what shared component to develop, how to organize their development and in what order the components should be developed. As we discussed earlier in the paper, these decisions are often taken in a implicit manner, as the organization has, at the point in time when these decisions need to be made, little experience in the domain during adoption. The consequence, however, is that the development approach adopted may

be different from the approach that would be optimal for the organization, resulting in mismatches and associated problems as discussed in the previous section.

In this section, we present five decision dimensions that we, based on our industrial experiences, consider to represent the most important decisions that an organization should take. These dimensions are feature selection, architecture harmonisation, R&D organization, funding model and shared component scoping. Each dimension presents a number of alternatives and discusses the advantages and disadvantages of each of these. The dimensions are not fully orthogonal and can, as discussed in the rest of the section, not be combined arbitrarily. In the next section, we discuss the three stages of product family adoption and the typical approach associated with each stage.

Finally, we use the terms “component team” to identify a group that is responsible for the development of a shared (or product family) component. However, a component team is not necessarily an organizational unit. A component team may be part of a product unit or even a virtual team consisting of members located at different product units. The term “unit” does refer to an organizational entity.

Feature selection

An important decision that must be taken early in the adoption of a software product family approach is what features and functionality to first move from the product specific to the product family realm. As discussed in [Geppert & Weiss 2003], especially the first components typically require a careful balance between the greatest perceived benefit and the highest chance of success. One can identify three approaches to selecting the first product family components.

Oldest, most generic, lowest components

The first approach is to start from the easiest components, maximizing the chance of success. These are typically the oldest, most generic components, often located in the lower layers of the product software architectures. The product teams often experience these components as necessary, but not relevant from a competitive perspective. Consequently, there is typically little resistance to move to a model where these components become shared among the products.

Advantages. As mentioned, there is typically little resistance against share these components. Also, the variability required from the shared component is well understood. In the case of multiple, evolving infrastructures that need to be supported, there can be a real benefit from having to evolve only one shared version of the component rather than multiple product specific ones.

Disadvantages. The product-specific component versions typically have existed for a long time and are often rather eroded. Consequently, the investment required for integrating the product-specific versions into one shared version may be substantial, whereas the benefit, in terms of reduced development effort, often is limited. Finally, COTS components may be or become available, removing the need for an internal component version.

Existing, but evolving, components

As a second starting point for the selection of components, one can focus on the product-specific components that have a relatively high change rate. Typically, these components implement features that are part of the competitive advantage of the products, but due to the quick evolution, there substantial maintenance cost are associated with these components.

Advantages. Assuming the changes are orthogonal to the product portfolio, a substantial reduction in maintenance cost can be achieved by merging the product specific components into one shared version. In addition, the required behaviour, although it is evolving, is understood reasonably well, simplifying the effort required.

Disadvantages. The main disadvantage is, obviously, first several product-specific versions of a component were developed that after a relatively short time are merged into a shared component, with the associated effort requirements. In addition, in the case that the evolution of the component is rather product-specific, the effort required for maintaining the shared component implementing the superset of features may be easily underestimated.

New, but common features

Finally, one may aim to develop shared components that implement currently unavailable functionality that is present on all or most product roadmaps. Rather than being reactive, the product family effort is proactive, developing shared components before the products need them.

Advantages. The main advantage over the first two alternatives is that no product-specific versions of the component are developed, so none have to be replaced. This reduces the development as well as the maintenance effort required for the new features.

Disadvantages. The concern one may have is that the product family R&D staff has to predict future requirements that the products themselves, being close to their markets, have not yet started to implement. There is, consequently, a high degree of uncertainty. This may cause certain investments to be invalidated, due to changing requirements.

Architecture harmonisation

The second dimension that we discuss is the level of software architecture harmonisation between the products that the organization strives for. In the case where a customer may use multiple products from the portfolio, harmonization of user interfaces as well as integration of product functionality, e.g. access to data stored in one product from the other product, is of considerable market value. In addition, the integration of shared components is considerably easier if the architectures are similar. On the other hand, products that, up to now, have evolved independently may require substantial effort in order to achieve architectural harmonization and the benefits may be limited.

Component-centric
The first model that the organization may decide to use is to forego all architecture harmonisation and to only develop shared components. As the integration of these components in the products will be more effort consuming, the organization needs to find a balance between the responsibilities of the component team versus the product teams with respect to facilitating and achieving component integration.

Advantages. The obvious advantage is that no effort has to be invested in architecture harmonisation. Secondly, the product teams maintain their relative freedom and are not required to achieve consensus over architectural issues.

Disadvantages. The component integration cost per product is often relatively high, reducing the benefits of a software product family approach. This may lead to organizational tension between the component team and the product teams. The product teams may demand from the component team to better prepare their component for integration in the products. The component team is concerned with the effort associated with preparing the component for inclusion in each product and generally not inclined to give in to the demands of the product unit.

Iterative product architecture harmonisation
For most organizations that adopt software product family engineering, there is a benefit associated with harmonisation of the product architectures. However, the cost of achieving this harmonisation is so high that it cannot be achieved in one release cycle. Instead, a roadmap is defined and architecture harmonisation is pursued in an iterative fashion, taking the process one step forward with every release cycle. Prerequisite for this approach is a reference product family architecture that is used as a (long-term) goal. This reference architecture needs to be designed jointly by the product and component teams.

Advantages. The main benefit of this approach is that, by starting with the architecture harmonisation of the product around the location of the shared components, the component integration cost can be reduced.

Disadvantages. There are some drawbacks with this approach as well. First, there is a need for product teams to invest in product architecture harmonisation, which has no immediate return on investment. Second, the product teams must agree on a reference architecture, which often is a lengthy and effort consuming process.

Revolutionary architecture adoption
In earlier work, we have described the revolutionary adoption model. In this case, a product family architecture and set of shared components is developed. Products that aim to use these shared components need to perform all architecture harmonisation at once without the ability to spread it out over multiple release cycles. On the other hand, once the harmonisation is achieved, integration of shared components is trivial.

Advantages. The main advantage is the ease of integrating shared components, once the architecture harmonisation has been achieved. Also integration of multiple products at customer sites and user interface harmonisation is greatly simplified.

Disadvantages. The effort associated with performing all architecture harmonisation in one release cycle typically leaves little effort for adding new

features, potentially reducing the competitive advantage that the products have in the market.

R&D Organization

The third decision dimension is concerned with the organization of the R&D staff involved in the development of shared components. Again, we have three alternative models that we discuss.

Mixed responsibility for product teams

The first model, not requiring any changes to the formal R&D organization, is where product teams assume responsibility for developing one or more shared components in addition to evolving the product or products in their portfolio. There are two alternative approaches. A product team may evolve a shared component with functionality needed for one of their products, releasing a new version of the shared component that can, at later stage, be extended with other functionality by another product team. Alternatively, a product team may have permanent responsibility for a shared component and extend it with new functionality in response to requests from other product teams.

Advantages. Simplicity is one of the key advantages; no organizational changes, which often require support from management, are required. In addition, there is no risk of shared components being extended with functionality that has no immediate market benefit.

Disadvantages. An important disadvantage is that, in this model, shared components often suffer from a high degree of design erosion. Product teams often have a tendency to add too product specific functionality to a shared component. In the case of permanent component responsibility, a product team has to balance requests from other product teams against its own release schedule demands.

Virtual component team

Especially in the case of replacing existing, product-specific components with one shared component, the organization use a virtual component team. The members of this team originate from the involved product groups and remain formally part of the product groups. However, as part of the virtual component team, the members develop the shared version of the component.

Advantages. The members of the virtual team have, due to their experience in the product teams, a good understanding of the product requirements on the component. Also, this approach requires no changes to the formal R&D organization.

Disadvantages. The members of the team often feel split in loyalty between the product and component team that they belong to.

Component unit

When a first release of a shared component, or a set of components, is available, it frequently becomes necessary to create an explicit component unit that has

responsibility for the component(s). In the case of a virtual component team, the team may even be converted into a unit.

Advantages. The responsibilities in the organization are clearly delineated and the places where trade-offs need to be made align with the boundaries in the R&D organization.

Disadvantages. The main disadvantage is that a component team has a clear risk of focussing more on its own goals than on the goals of the product teams, especially in the time dimension. There is a tendency in component teams to build a "perfect" component, rather than satisfying the immediate needs of the product teams.

Funding

In all commercial organizations, the final metric is "the bottom line", i.e. the financial consequences of decisions. Adopting product families is no different in this respect. As the profit and loss responsibility in many organizations even exists at product team level, the development and evolution of shared components requires a suitable funding model. Below, we discuss three models for funding product family initiatives that are used in the software industry.

"Barter"

In organizations that use this model, the effort that is invested is not quantified in terms of effort or value. Instead, different product teams agree to develop shared components and to provide the components to the other product teams. Negotiations are based on a form of equal sharing of the development and evolution effort and on trust concerning the ability and willingness of the other teams to deliver their contributions in time and according to specifications.

Advantages. The main advantages are that this model requires no changes to the budget or the organization, lacks any bureaucratic overhead and can be initiated easily by a few enthusiastic people at the right places in the R&D organization.

Disadvantages. The hand-shake deals that are associated with this model require a high level of trust in the organization. One product team may run into delays in an ongoing project, causing delays in the delivery of shared components. Unforeseen events may easily put pressure on the agreements causing the initiative to fail.

Taxation

The second model is where product teams agree on a form of taxation where each product team, based on size, profit and expected use of the shared components, agrees to contribute development effort or financial support. The taxation may be implemented through the initiation of a separate component team or by product team itself committing to the development some shared component. This model is basically an investment model and is more formal than the "barter" model, i.e. the investment is quantified and delivery schedules agreed upon.

Advantages. The model provides a more formal agreement, increasing the level of trust towards the initiative. Also, it becomes feasible to develop explicit roadmaps and release plans based on the available investments.

Disadvantages. The investment in product family engineering becomes explicit in the budget and, in the case of a component team, in the organizational structure. Also, especially in the case where a taxation-based component team exists for a number of years, there is often a tendency to developing "perfect" components rather than helping product teams addressing their immediate concerns due to lacking market pressures.

Licensing/royalty

Once a usable version of a shared component is available, it becomes feasible to move to a royalty or licensing model. In this case, the component team is not funded through taxation but through the royalties that it receives based on product sales of products that contain the shared component. The component team may be in direct competition with other component teams in the organization or with external COTS components and consequently under substantial market pressure. Also, in the case of too high royalties, product teams may decide to develop the component themselves.

Advantages. This model provides the market pressures for the component team that are necessary to keep it focused on satisfying customers and on innovating and evolving their components. Also, once the team is unable to fund itself based on its royalties because lower priced COTS components have become available, it provides a natural way to dissolve the component team and to assign its members to other tasks.

Disadvantages. The model should not be applied until the shared component has achieved a level of maturity that allows the component team with a realistic budget for component evolution and innovation. In the case it is decided that the shared component represents core competences that the organization should maintain, this model is not appropriate as the market pressure is lacking.

Shared component scoping

The final decision dimension that we discuss in this paper is the scoping of the shared component. Of the features required by the products that use the shared component, the component can implement only the most common ones, the subset used by some or more products or all features, including those required by only one product.

Only common features

The starting point for most shared components is to implement only those features that are common for all products. The product teams have to extend the component with the features that their product needs.

Advantages. Assuming the common features represent a substantial subset of the features, this approach provides an efficient way to achieve early successes in the product family adoption.

Disadvantages. One concern with this approach is the often complex interface between the shared component and the product specific functionality that has to be developed on top of it. In addition, all product teams have to maintain staff with

knowledge of this interface, and often the component internals, resulting in inefficient knowledge management.

Complete component with plug-in capability

Over time, a shared component evolves by incorporating more and more features until the point is reached where only features required by only one or two products need to be added to the component. For this, the component provides a plug-in interface.

Advantages. This model resolves, to a large extent, the issues identified for the previous model concerning the complex interface and the inefficient knowledge management.

Disadvantages. The main concern often is the integration of the component in the product. Depending on the complexity of the interaction between the component and the remaining product, the integration may be a substantial challenge.

Encompassing component

The final model is where the component team aims to incorporate all product needs for the component in the component itself. In addition, the component team may take responsibility for (part of) the integration of the component in the products of their customers.

Advantages. Especially in the case where the component has a complex, e.g. real-time, interface with the rest of the product, this model is the most efficient as the component team has a better understanding of the behavioural characteristics of the component.

Disadvantages. The component team will typically need to be larger than in the earlier models, due to the service that it provides to the product teams. This may easily be viewed as inefficient by the organization.

Summary

As we discussed in the introduction to this section, organizations adopting or employing software product families need to decide what shared component to develop, how to organize their development and in what order the components should be developed. Due to the implicit manner these decisions typically are taken, organizations often experience problems due to mismatches between the actual and optimal model used.

We have presented five decision dimensions that, based on our industrial experiences, we consider to represent the most important decisions that an organization should take. These dimensions are feature selection, architecture harmonisation, R&D organization, funding model and shared component scoping. In the figure below, these dimensions as well as the alternatives on each dimension are presented graphically.

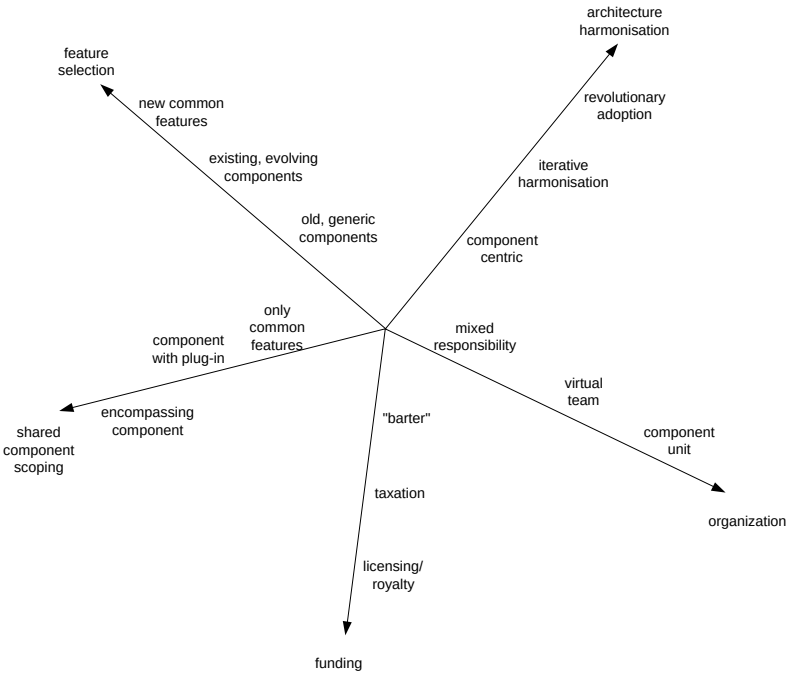

Figure 2. Five decision dimensions and associated alternatives

4. Decision Framework

When implemented successfully, software product families provide substantial benefits to the organization in terms of development effort, time-to-market of new features and products and the level of integration of the product portfolio. However, in order to achieve these benefits, the organization first has to adopt the product family approach, which is a major change process in the organization, primarily affecting the R&D organization, but also product management, marketing and other groups in the organization.

The adoption of a product family approach is primarily a business, organization and process challenge and the technical challenges are, although present, not of predominant importance. Because of the breadth of the challenge, it is important, especially early in the adoption, to select the starting point that provides maximal benefit to the organization, but also has a high likelihood of success. Based on this, one can identify three stages that product family adoption typically evolves through, i.e. the early success phase, the expanding scope phase and the increasing “maturity” phase. Once the last stage is reached, product family engineering has been embedded as the normal way of working in the organization. The business strategy of the organization then influences the evolution of the product family.

In the sections below, we discuss, for each of the decision dimensions, the alternatives that in the typical case are the preferred ones. Specific organizations may deviate from these for legitimate reasons.

Initial Adoption - Early successes

The first stage that organizations adopting software product families go through is the initial adoption. During this stage the first steps are made away from independent product teams. Although the R&D organization may have a positive attitude towards the overall initiative, the complete adoption process will initiate many changes and, especially, increase the dependencies between different R&D teams. Because of this, it is important to minimize the number of impact of the required changes, to maximize the chance of success and to maximize the benefits from the success. The principles that should guide the initial adoption are:

- Assign a champion or adoption team that champions the approach.
- Avoid organizational changes.
- Minimize the product family specific processes to a bare bone set.
- Select features that create maximal visibility and benefit.
- Failure is not an option – if the initiative fails to deliver in this stage, its termination is guaranteed.

The initial adoption phase ends with the successful integration of one or more shared components in at least two products. At this point, the product family initiative has, hopefully, achieved its first successes and there is hard evidence that the benefits of product family engineering can be achieved in the context of the organization.

In the sections below, we discuss, for each decision dimension, the preferred alternative as well as the situations in which an organization should deviate from it.

Feature selection: new, but common features

The preferred alternative for the features to be implemented as a product family are those that have not yet been implemented by the existing products, but are needed by at least one or two products in the next release.

There are cases where the products service fundamentally different markets, leading to a lack of common new features. In this case, existing product specific components with a high change rate where the changes are preferably cross-cutting the product portfolio are a suitable alternative.

Architecture harmonisation: component-centric

R&D organizations are typically hesitant to invest in changes that have no obvious, short-term return on investment. Although harmonisation of the product architectures is of importance in the long term, due to the reduced integration effort, it is also an effort-consuming activity with no immediate pay-back. Consequently, at this stage in the adoption process, effort should be directed towards creating product family components.

Organization: mixed responsibility for product teams
It is difficult to present a general preference for this dimension as it is preferable to have a component team developing the first product family components. However, obtaining the management support for creating a component team is typically difficult, due to the scarceness of R&D resources. The main risk of the recommended alternative is, obviously, the prioritization by the product teams. The aim should be to assign the responsibility for component development to product teams that have an immediate need for incorporation of the product family components in their product.

Funding: "barter"
Similar to the decisions dimensions already discussed, the main challenge is to achieve the early successes with minimal changes. In the case where different product teams decide to share responsibility for the development of shared components, it is preferable to initially achieve agreement based on estimations that can be made part of the normal product development budgets. If substantial organizational support is available, a taxation model can be considered.

Shared component scoping: only common features
Assuming product teams have shared responsibility for components and products, the least intrusive approach is to have product teams implement the features needed by their own product, leaving extension points for likely future extensions. The next product team that requires use of the product can extend the first version of the shared component with its requirements. The main challenge in this model is one of discipline, i.e. each product team is responsible for maintaining the general applicability of the shared component.

Expanding scope

Once the first product family components have been successfully integrated in two or more products and the R&D organization is experiencing the benefits of product family engineering, the product family initiative evolves to the second phase. The second phase is concerned with expanding the scope of the product family in terms functionality that is in the product family domain. R&D teams have grown accustomed to the additional dependencies and learned to trust other teams to an extent. The situation is becoming more complicated in that product family artefacts developed in the first phase need maintenance and an organizational model for this has to be established. In short, after the initial rather ad-hoc, but efficient, adoption phase, now more structure is needed to institutionalize the product family approach. The principles that guide the second phase are addressing this need:

- Preferably each product team should contain a product family champion.
- Some organizational change is necessary to embed product family engineering in the R&D organization.
- Introduce product family processes that safeguard the reliability of the shared components as well as their predictable evolution.

The second adoption phase ends at the point when no obvious extensions to the shared components can be identified by the R&D organization. At that point, the challenge shifts to increasing the "maturity" [Bosch 02] of the product family artefacts as a means to decrease the product derivation cost.

In the sections below, we discuss, for each decision dimension, the preferred alternative as well as the situations in which an organization should deviate from it.

Feature selection: existing, but evolving, components
Assuming the product family components for new, common features have already been developed in the first phase, the next step is to address the components that currently have product-specific implementations, but share common features. In particular, the components that experience a high chance rate in response to new requirements, that preferably are orthogonal to the product portfolio, are of interest. Frequently, these components are considered to be problematic within product teams, creating momentum for moving to a shared implementation. The second phase marks, in this sense, a shift from reducing development effort to reducing maintenance effort.

Obviously, if opportunities exist for the development of shared components for new, common features, these should be pursued as well.

Architecture harmonisation: iterative product architecture harmonisation
With the increasing number of shared components, the cost of integrating these components in the product architecture often starts to become a concern. This calls for the design of a product family architecture that captures the commonality and variability of the products in the product family scope. The product teams use this architecture as a reference and aim to iteratively harmonize the product architecture with the product family architecture.

Organization: virtual component team
The shared components developed in the first phase need to be maintained. In addition, the shared components created in this phase are mined based the product-specific implementations of these components. This requires, on the one hand, a team of people that is permanently assigned to a set of components. On the other hand, the team needs detailed knowledge of the product specific requirements. A virtual team, consisting of staff formally located at the product teams, provides an optimal balance between these concerns.

Funding: taxation
The effort required for the product family initiative as well as the importance of continuity of the members of the virtual component demand a strong commitment from the R&D organization. Informal agreements about development need to be replaced with a more formalized model that guarantees more continuity. The taxation model requires all units and teams, during the budget rounds, to reserve part of their resources for the product family initiative.

Shared component scoping: complete components with plug-in capability
The shared components replacing their product-specific counterparts should, preferably, be as easy to integrate as the original components. Consequently, it is necessary for these components to cover most of the features required by the products only requiring plug-ins for the features that are used by only one product.

Increasing "maturity"

Once the obvious product family components have been developed and the second phase of product family adoption has ended, the main challenge becomes to increase the "maturity" of the product family artefacts. In [Bosch 02], we present five maturity levels for product families:

- **Independent products**: Initially, the organization develops multiple products as several independent entities. These products do not share software artefacts in any planned way, only by coincidence.
- **Standardized infrastructure**: The first step towards sharing is when the organization decides to standardize the infrastructure based on which a set of products is built.
- **Platform**: A subsequent development is to extend the standardized infrastructure with internally developed software artefacts, i.e. a platform, that provide functionality that is common to all products in the scope. Each product is developed on top of the platform.
- **Software product family**: The fourth level of maturity is when the organization employs a software product family in the traditional sense of the term [Bosch00]. The shared artefacts in the product family contain functionality that is shared by all or a subset of the products. Products may sacrifice efficiency or other requirements for the benefits of being member of the product family.
- **Configurable product base**: The final level is the situation where the differences between the different products are so well understood that these can be mapped to variation points that can be bound at installation or run-time. As a consequence, individual products are derived by configuring the shared software artefacts appropriately.

The challenge in this phase is to lift the product family from the platform level at which it typically is after the second adoption phase to the product family and, if feasible, the configurable product base level.

In the sections below, we discuss, for each decision dimension, the preferred alternative as well as the situations in which an organization should deviate from it.

Feature selection: all components
With the ambition to bring all products in a configurable product base, the scope for feature selection is now extended to all components, including the old, common and low-level components. Of course, for all components considered for inclusion in the product family, a positive cost benefit evaluation should be available. The ultimate goal remains to reduce development and maintenance effort, improve time-to-market and increase integration between different products in the portfolio.

Architecture harmonisation: revolutionary architecture adoption
At this stage in the product family adoption, the products that were part of the initial adoption have evolved through several iterations of architecture harmonisation. However, new products interested in joining the product family initiative need to go through a revolutionary architecture adoption process. Reusing the product family components will require the new products to make substantial changes to their architecture.

Organization: component unit
Even if up to now, no component units were created, the product family is, at this point, so embedded in the R&D organization that it is no longer feasible to proceed without component units. However, as discussed below, after initial taxation based funding, these component units should be funding through licensing or royalty fees. The aim is to make sure that component units are exposed to market pressures from within and outside the organization.

Funding: licensing/royalty
As discussed, as the R&D organization instantiates component units, it is important for these units to be exposed to market pressures. A taxation based model easily leads to component units that are more concerned with creating a 'perfect' component than with the needs of their customers.

Shared component scoping: encompassing component
A logical consequence of the customer focus of component units is the attention towards minimizing the integration problems of the component in specific products. In this stage one can typically identify a shift from shared components extended by product teams to shared components prepared by component units for easy integration, i.e. the encompassing component model.

5. Related Work

Several authors have discussed the different aspects concerning the adoption and institutionalisation of software product families. [Weiss & Lai 99] present an approach to product family engineering that aims to achieve generation of products from the product family artefacts. Although we agree with the authors that automated generation is the long term goal for most product families, the focus of this paper is on the early phases of adoption.

[Clements & Northrop 01] presents 29 practice areas that can or should be addressed for software product family engineering. Launching and institutionalizing is discussed as one practice area and some of the issues discussed in this paper are also addressed. Different from these authors, we present the five decisions dimensions and present a three-stage adoption process.

[Böckle et al. 02] discuss the importance of adopting and institutionalizing a product family culture. Some aspects, such as the adoption approach, that are discussed have relations to this paper, but are presented much more brief. Also, [Wijnstra 02] discusses the importance of the 'weight' of introduction and supports our approach of limiting up-front investments and iteratively extending scope and maturity.

[Kang et al. 02] discuss the use of marketing and product plans to select and drive the development of product family components. The work discussed in that paper would provides valuable input to the feature selection during product family adoption. Other others discussing the process of scoping and roadmapping include [Ommering 01], [Schmid 02] and [Kishi et al. 02].

6. Conclusion

Once adopted successfully, software product families provide a considerable benefit for software R&D organizations. However, achieving the successful adoption proves, in many organizations, to be non-trivial. Our claim is that this is due to the mismatch between the optimal and applied approach to the development, evolution and use of shared product family artefacts. The mismatch may lead to several problems, such as the mismatch between shared components and product needs, high level of design erosion and complex interfaces of shared components, high degree of "organizational noise", inefficient knowledge management, ripple effects through the R&D organization due to component evolution and incorrect assumptions about the value of shared components.

To address these problems, we have presented five decision dimensions that are of relevance to the development of shared components, i.e. feature selection, architecture harmonisation, R&D organization, funding model and shared component scoping. For each dimension, we have presented three alternatives.

These dimensions are used in a decision framework that describes the preferred alternative for each stage in the adoption of software product families. We recognize three main stages in product family adoption, i.e. early successes, increasing scope and increasing maturity. In the table below, the stages and the preferred alternatives are presented.

	Early successes	**Increasing scope**	**Increasing maturity**
Feature selection	New, but common features	Existing, but evolving, components	All components
Architecture harmonisation	Component-centric	Iterative product architecture harmonisation	Revolutionary architecture adoption
R&D organization	Mixed responsibility for product teams	Virtual component teams	Component units
Funding	"Barter"	Taxation	Licensing

Shared component scoping	Only common features	Complete components with plug-ins	Encompassing component

The contribution of this paper is that we present a framework for deciding the optimal model for organizing the development of shared components. In future work, we intend to add more detail to the model in terms of advantages, disadvantages and risks. In addition, we hope to add a number of industrial cases to the presentation of the decision framework to further illustrate and validate the decision framework.

References

[Bosch 00] J. Bosch, 2000. Design and Use of Software Architectures: Adopting and Evolving a Product Line Approach, Pearson Education (Addison-Wesley & ACM Press), ISBN 0-201-67494-7.

[Bosch 02] J. Bosch, Maturity and Evolution in Software Product Lines: Approaches, Artefacts and Organization, Proceedings of the Second Conference Software Product Line Conference (SPLC2), pp. 257-271, August 2002.

[Böckle et al. 02] G. Böckle, J. Bermejo Muñoz, P. Knauber, C.W. Krueger, J.C. Sampaio do Prado Leite, F. van der Linden, L. Northrop, M. Start, D.M. Weiss, Adopting and Institutionalizing a Product Line Culture, Proceedings SPLC2, pp. 49-59, LNCS 2379, 2002.

[Clements & Northrop 01] P. Clements, L. Northrop, 2001. Software Product Lines: Practices and Patterns, SEI Series in Software Engineering, Addison-Wesley, ISBN: 0-201-70332-7.

[Geppert & Weiss 03] B. Geppert, D. Weiss, Goal-Oriented Assessment of Product-Line Domains, 9th International Software Metrics Colloquium, Sydney, Australia, 2003.

[Gurp & Bosch 02] Jilles van Gurp, Jan Bosch, 'Design Erosion: Problems & Causes', Journal of Systems and Software, 61(2), pp. 105-119, Elsevier, March 2002.

[Jansen et al. 04] Anton Jansen, Jilles van Gurp, Jan Bosch, 'The recovery of architectural design decisions', submitted, 2004.

[Kang et al. 02] K.C. Kang, P. Donohoe, E. Koh, J. Lee, K. Lee, 'Using a Marketing and Product Plan as a Key Driver for Product Line Asset Development', Proceedings SPLC2, pp. 366-383, LNCS 2379, 2002.

[Kishi et al. 02] T. Kishi, N. Noda, T. Katayama, 'A Method for Product Line Scoping Based on a Decision-Making Framework', Proceedings SPLC2, pp. 348-365, LNCS 2379, 2002.

[Ommering 01] R. van Ommering, 'Roadmapping a Product Population Architecture', Proceedings of the 4th International Workshop (PFE 2001), LNCS 2290, pp. 51-63, 2001.

[Schmid 02] K. Schmid, 'A comprehensive product line scoping approach and its validation', Proceedings of the 24th International Conference on Software Engineering (ICSE 2002), pp. 593-603, 2002.

[Weiss & Lai 99] Weiss D. M., Lai, C. T. R., 1999. Software Product-Line Engineering: A Family Based Software Development Process, Addison-Wesley, ISBN 0-201-694387.

[Wijnstra 02] J.G. Wijnstra, 'Critical Factors for a Successful Platform-Based Product Family Approach', Proceedings SPLC2, pp. 68-89, LNCS 2379, 2002.