# 3 Using Costing Information as Decision Support in Variability Management

Holger Schackmann, Horst Lichter

*Decisions on the scope of a software product line and the selection of variability realization techniques have a large impact on costs and therefore substantially affect the economic viability of a product line approach. But there is a lack of information to guide these decisions, since there is no clarity on the cost implications of variability. To close this gap, we propose an approach to costing that enables a more accurate allocation of incurred costs to the features within the software product line. More accurate costing information provides guidance for future scoping decisions and enables an assessment of the cost implications of different variability realization techniques as a first step towards a cost model for the reasoned selection of appropriate techniques.*

## 3.1 Introduction

The principal decisions in managing variability within a software product line (SPL) are made on two levels. On the requirements level the scope of the SPL must be defined, i.e. it must be identified which features should be variable within the SPL and decided which variants should be supported. Secondly, it has to be decided how to implement the variability by selecting appropriate variability realization techniques [1]. These decisions have to be taken during the initial development as well as during the further evolution of an SPL.

Decisions on the two levels are rather intertwined. The planned variability within the SPL influences the choice of variability realization techniques. These can in turn limit or facilitate adjustments of the scope.

The remainder of this paper is organized as follows. Section 2 depicts the challenges of managing the variability during evolution of a SPL. Section 3 discusses influencing factors for the selection of variability realization techniques. Section 4 depicts deficiencies of current costing approaches in gathering the costs implications of variability and proposes a costing approach based on features as cost objects. Section 5 then describes how this information can be used to guide variability management decisions. Section 6 provides a brief summary.

## 3.2 Scoping during SPL evolution

There exist several approaches for the initial definition of the scope [3]. But the problem of scoping during maintenance and evolution of an SPL is rather unexplored. In practice we encounter that SPLs are introduced gradually by exploiting commonalities between products that had been developed in customer

specific projects. Thus an exhaustive scoping was not performed and scoping decisions may not be documented or lost during further development.

Typical risks associated with initial scoping are a too big or too small definition of the scope [4]. During evolution of a product line we rather face the problem of an increasing variability within the scope. Several reasons may lead to the introduction of new variants of features. Due to deadline pressure it may often be faster to produce a customer specific solution instead of spending additional effort on investigating chances for reuse and coordinating necessary changes with other product developments. Fulfilling new customer requests with new customer specific features may be motivated by the concept of customer orientation and disregard of the follow-up costs. A large amount of variability is accompanied by an increasing technical and cognitive complexity. So the offering of features can not completely be communicated to the customer. Similarities of customer requests to existing features will be difficult to re-cognize, so similar features will unconsciously be developed several times.

Summarized, variability becomes a major cost driver, since each customer specific feature must be maintained, integrated in subsequent releases, tested separately and possibly considered during deployment, user training and customer support. A problem is that these costs are not transparent. Therefore it is not possible to balance the follow-up costs against the value that a specific feature offers to the customers.

## 3.3 Selection of variability realization techniques

Since the scoping activity analyses potential products and possible future extensions, this information is a basic guidance for the selection of variability realization techniques. For example it should be considered to how many products a specific feature will be relevant. If it is necessary in most of the products a variability mechanism should be selected that allows easy configuration of this feature during application engineering. If the feature is only needed in a few products it may suffice to provide a realization technique that requires product specific development, e.g. specializing particular classes.

Besides the scoping decisions there are some other influencing factors that will be described in the following.

## 3.4 Requirements of the application domain

There typically exist non-functional requirements of the application domain which discourage the use of certain variability realization techniques. As an example performance requirements can rule out variability realization techniques

with runtime binding. Memory constraints may require a binding before start-up time.

Furthermore there may be the requirement to allow easy reconfiguration or upgrade of the system, possibly even at runtime, which necessitates a runtime binding of the variability.

Fritsch et al. describe an approach to evaluate variability realization techniques according to qualities, like extensibility or performance [5]. But an evaluation in general is difficult since the fulfillment of a quality depends on other influencing factors like the architectural context and the sound application of the variability realization technique.

### 3.4.1 Existing architecture

The variability realization techniques that are already used in the existing product line architecture have to be taken into account. Relying on established techniques reduces the complexity within the architecture, compared to the usage of a many different realization techniques. Furthermore those techniques and their implications may be better understood by the developers.

Also the presence of legacy components within the SPL can rule out the usage of novel variability realization techniques like aspect oriented programming.

### 3.4.2 Organization

Different organizational models can be applied to SPL development, dependent on factors like the size of the product line or the physical location of the staff [6]. Choosing appropriate variability realization techniques can substantially reduce the necessary coordination efforts between different development units. For example a domain engineering unit can support a certain variant feature by providing alternative architectural components. During application engineering only one variant has to be selected. Thus the complexity visible to application engineering units is reduced and the interfaces between the development units are kept small.

Nevertheless application engineering units may need some flexibility for product specific adaptations. Choosing variability realization techniques that are open for adding variants during application engineering can help to define the points where product specific adaptations are allowed, while other variant features can be kept under control of the domain engineering unit. This can help to limit the growth of variability within the product line, which may be caused by the independent development of similar features for different products within the SPL.

### 3.4.3 Follow-up costs

The selection of variability realization techniques has a large influence on the costs of subsequent activities. Some techniques may have higher initial development costs in domain engineering, but therefore allow a fast development of new variants during application engineering.

Costs for product derivation and testing are recurring costs in each release cycle of a product. Using variability realization techniques that allow a configuration-based product derivation, these costs will probably be lower compared to variability realization techniques that require the development and maintenance of product specific assets.

Costs for deployment of a product are recurring costs for each customer site and for each product release or update. The installation at the customer site may be more or less complex, partly influenced by the applied variability realization techniques. If complex configuration steps are necessary, an expert might be needed for installation on customer site, while in other cases an automatic installation is possible.

Furthermore maintenance costs are affected, since the selection of variability realization techniques influences testability and comprehensibility. For example using preprocessor directives can lead to a scattering of variation points, which makes debugging and changing the software more difficult.

Summarized, there are many influencing factors on the follow-up costs of variability decisions, such as the number of customers, the number of products and their release cycles, as well as the amount of change requests during evolution.

### *3.4.4* Current practice

In current practice selection of variability realization techniques is rather based on ad hoc decisions or primarily influenced by the techniques already used in the architecture as well as by non-functional requirements from the application domain, as far as they have been identified. Other factors remain unconsidered due to insufficient scoping during evolution and no transparency on the follow-up costs of variability decisions. The multitude of influencing factors complicates the construction of a cost model that would enable a reasoned selection of variability realization techniques. Therefore sound decision criteria for the selection of variability realization techniques, as well as for scoping during evolution are lacking. We believe that gaining more clarity on the cost implications of variability will help to overcome this situation.

Existing cost models for SPL development rather follow a top down approach and support SPL investment decisions on a high level (see [2] for an overview).

Gathering more accurate information on the costs of variability can provide complementary input for these models as a first step towards the development of a cost model for the reasoned selection of appropriate variability realization techniques.

## 3.5 Costing approaches

### *3.5.1* Deficiencies in traditional costing methods

Traditional costing methods in software development usually allocate costs either to customer projects, maintenance projects or internal development projects. This may suffice for accounting and budget control. But with multiple reuse the relationship between the direct labor hours that went into development and the costs of software breaks down[7]. Significant costs are incurred by the reuse infrastructure. Neither the benefits of reusing assets within application engineering, nor the costs of introducing additional variabilities can therefore be gathered with these costing methods.

### *3.5.2* Activity based costing

Activity based costing was originally developed in the context of manufacturing industry, motivated by problems in managing an increasing variety within the product portfolio[8][9]. It was recognized that traditional costing systems failed to allocate the overhead costs caused by additional product variants in a reasonable way, since these costs are allocated to products on a per-unit basis. This leads to suboptimal decisions in portfolio definition and product construction, due to an unconsciously cross-subsidization of rather exotic products.

Rather than viewing work products as direct consumers of overhead resources, activity based costing introduces the activities of the production process to stand in between work products and consumed resource costs. This allows a more accurate allocation to the cost objects consuming the activities, e.g. products, services or customers. The improved cost transparency can guide decisions to establish profitable customer relationships, e.g. by changing the construction of products, changing the prices, reconfiguring or replacing products, improving production processes, or eventually abandon a product completely.

### *3.5.3* Activity based costing in software development

Analogously to the cost anomalies in manufacturing the costs of developing and maintaining an SPL are for the greater part indirect or overhead costs, which can not easily be allocated to a certain product or customer.

Can an activity based costing approach be applied to software development? Fichman and Kemerer have proposed the adoption of activity based costing to component based software development, to address the need for economic incentives for investments into reusability [7].

While the relevant activities can be identified based on defined development processes, it is not evident what should be the primary cost objects. The allocation of activity costs to customers or software components respectively raises many difficulties. It is not clear how development and maintenance costs of core assets can be distributed to customers. The use of software components as cost objects is problematic when there is no direct relation of an activity to a certain component, e.g. activities like requirements engineering or system testing.

### 3.5.4 Features as cost objects

As described before, the number of customer specific features has a large impact on costs. Therefore one can analyze which features are standard features that are relevant for most customers and which features can be seen as more exotic features, since they are only included in products for one or a few customers. Gathering the costs caused by those exotic features more accurately, would help to attain more clarity on the influence of variability on costs. Therefore we propose the use of features as the primary cost object for activity based costing in SPL development. Feature models describe the commonalities and variabilities within an SPL [10][11]. So this approach can capture the costs associated to those variabilities. Figure 3 gives an overview of the cost assignments that are described in the following.

Personnel costs are dominating the costs of software development. Other costs, like costs for tools or hardware, can be apportioned to the personnel costs. Therefore it may suffice to concentrate on personnel costs and differentiate between hourly wages according to the qualification level required for an activity. Existing change request management systems, task management systems or time registration systems provide a basis for the resource costs assignment to activities, since they basically enable a detailed gathering of labor hours for most activities.

Since features are visible in all phases of the development life cycle, most development assets, like requirements, software components, test cases, documentation, change requests and even support calls can be linked to one or more features. Thus the cost for most activities can be distributed to features as cost objects.

A catalogue of relevant activities can be developed based on a defined development process, including for example activities like requirements engineering,

design, implementation and test during the initial development of a new feature, as well as activities during evolution of a feature, like the handling of change requests and bug reports and the development of product-specific adaptations of features. To enable an analysis of the cost structures it should be differentiated between activities in domain engineering and application engineering and furthermore between activities associated to the initial development of a feature and further maintenance activities.

It will not in all cases be possible to allocate the costs of each activity to a single feature, e.g. initial requirements engineering activities which can be associated to several features. But since a feature model typically contains concept features to allow a hierarchic structuring of the model, such costs may be allocated to a concept feature that structures a set of features associated with the activity. In other cases it can be reasonable to distribute the costs of an activity to more than one feature, e.g. if the interaction of features is the subject of a maintenance activity. So this should allow the allocation of most of the former indirect costs, like maintenance costs of core assets, to features.

There will nevertheless be activities that can not be allocated to features. Either an association to a feature or a set of features can not be identified, e.g. for management activities, or gathering the related costs would be impractical or too expensive, e.g. the assignment of the costs of documentation activities to features. These remaining costs must be allocated to projects as before.

To enable a further allocation of the cost from features to customers, an assessment of the importance or value of the provided features to the different customers is necessary. It might not be possible to express this value in monetary units, but it suffices to assess the contribution of a feature to user satisfaction on a simple scale of values. Techniques like the Kano method [12] or Quality Function Deployment [13][14] may be applied to this purpose. The assignment of the costs of a feature to the customers can then be based on the different importance of the feature to its customers.

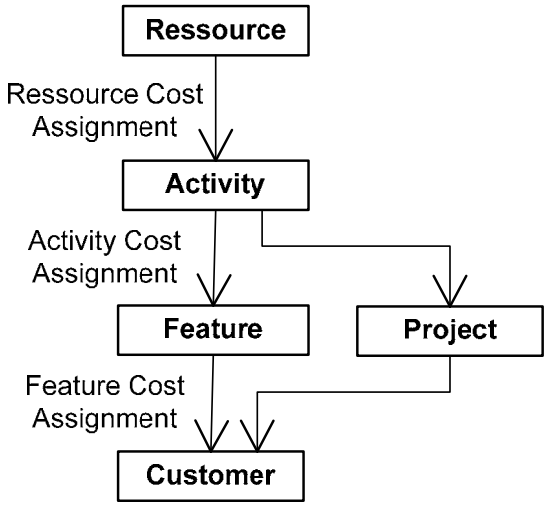


Figure 3: Features as cost objects

## 3.6 Using costing information in variability management

The depicted approach to costs enables an improved cost transparency. A better understanding of the relation between initial development costs of a feature and its follow-up costs can improve future cost estimation and guide the pricing of development and maintenance contracts. The costing information can also guide variability management decisions in several ways.

### 3.6.1 Guidance of scoping decisions

The allocation of activity costs during development and evolution to features will give a more realistic view on the costs associated to a feature. If costly features are identified, it must be analyzed which importance or business value these features provide to the customers, and which profits or competitive advantage is gained for the development organization by offering these features.

Features which are not that important for the customers but generate high additional costs can be identified. They can either be replaced by existing similar features, modified in a manner that allows a better fit to the SPL architecture, or can possibly be abandoned completely. If a feature is important for certain customers, it must be examined up to which extent the customers can be charged with the real costs.

In all cases the importance of a feature or customer for the market strategy has to be considered. It can be a reasonable decision to take a loss in order to open

or develop a market. But to decide on the strategy, there must be some estimation where and how much money is lost.

### 3.6.2 Support of selection of variability realization techniques

More accurate information on costs will enable to analyze the cost structures associated to a feature and its variants. It can be gathered which costs are spent for the initial development of a feature, for the development of new variants of this feature, for maintenance of this feature and its variants, and eventually for testing in subsequent releases. Furthermore it can be gathered how all of these costs are distributed between domain engineering and application engineering. Moreover the influence of change requests on these costs can be retrieved.

An investigation of the cost implications of different variability realization techniques can be based on comparisons of the cost structures for different features and its variants and searching for regularities or recurring patterns. For example an analysis of the distribution of maintenance costs between domain engineering and application engineering may help to identify which realization techniques require high maintenance efforts within application engineering and which techniques leave most of the maintenance work within domain engineering thereby possibly reducing overall maintenance costs. It can also be investigated, if there are variability realization techniques which facilitate the development of new variants.

Furthermore these observations may enable better estimations of future costs, e.g. the costs for developing a new variant of a feature, the costs of product derivation, or the maintenance costs of existing features.

All together, the empirical observations enabled by the described costing approach facilitate a better understanding of the cost implications of variability. Therefore this can be a first step towards the development of cost model for the reasoned selection of variability mechanisms.

## 3.7 Further work

In our further work we aim at developing a detailed model for a costing approach based on features as cost objects. Within the context of an ongoing industry cooperation project we will conduct a case study on gathering cost information in an SPL development process. To provide appropriate tool support, the collection of costing information should be integrated with our existing tool set for feature modeling [15].

## 3.8 Summary

Scoping decisions during the evolution of a software product line, as well as the selection of variability realization techniques are in current practice rather based on ad hoc decisions. The difficulties in steering the variability and selecting appropriate realization techniques lead to increasing maintenance costs. We believe that one cause of this problem is a lack of sound information on the costs implications of variability.

This resembles a problem suffered in manufacturing industry. Given an increased number of product variants, traditional costing approaches were unable to provide a sound cost allocation and therefore lead to wrong decisions in product portfolio definition and product construction. This problem was addressed with the development of new costing approaches like activity based costing.

In this paper we propose the application of activity based costing to software product line development using features as primary cost objects. Features provide a natural basis for allocating costs and are anchored in existing techniques for variability modeling.

Bringing together cost information based on features as cost objects and the assessment of the customer value of features provide a sound guidance for scoping decisions. Furthermore the cost implications of the selection of variability realization techniques can be investigated to obtain an empirical basis for the development of a cost model for the reasoned selection of variability mechanisms.

## 3.9 References

[1] Svahnberg, M., J. van Gurp, J. Bosch (2004): A taxonomy of variability realization techniques. Software – Practice and Experience. Wiley Intersciene, Chichester.

[2] Clements, P.C., J.D. McGregor, S.G. Cohen (2005): The Structured Intuitive Model for Product Line Economics (SIMPLE). CMU/SEI-2005-TR-003, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania.

[3] Schmid, K. (2003): Planning Software Reuse – A Disciplined Scoping Approach for Software Product Lines. PhD Theses in Experimental Software Engineering, vol 12, Fraunhofer IRB Verlag, Stuttgart.

[4] Clements P., L. Northrop (2001): Software Product Lines: Practices and Patterns, Addison Wesley, Boston, Massachusetts.

[5] Fritsch, C., A. Lehn, T. Strohm (2002): Evaluating Variability Implementation Mechanisms. In Schmid K., Geppert B. (Eds.) Proceedings of the Second International Workshop on Product Line Engineering PLEES'02, IESE-Report No. 056.02/E. Fraunhofer IESE, Kaiserslautern.

[6] Bosch, J. (2000): Design and Use of Software Architectures: Adopting and Evolving a Product Line Approach, Pearson Education, Harlow.

[7] Fichman, R., C. Kemerer (2002): Activity Based Costing for Component-based Software Development. Information Technology and Management, vol 3 (1/2), 137-160. Springer, Netherlands.

[8] Schuh, G. (2005): Produktkomplexität managen – Methoden, Strategien, Tools. (in German) Hanser Verlag, München.

[9] Kaplan, R.S., R. Cooper (1997): Cost and Effect. Harvard Business School Press, Boston, Massachusetts.

[10] Kang K., S. Cohen, J. Hess, W. Novak, A. Peterson (1990): Feature-Oriented Domain Analysis (FODA) Feasibility Study. CMU/SEI-90-TR-021, ADA235785, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania.

[11] Lee, K., K.C. Kang, J. Lee (2002): Concepts and Guidelines of Feature Modeling for Product Line Software Engineering. In Gacek C. (Ed.) Proceedings of the 7th International Conference on Software Reuse: Methods, Techniques, and Tools, ICSR-7, Austin, Texas. 62-77. LNCS 2319, Springer, Berlin.

[12] Kano, N., N. Seraku, F. Takahashi, S. Tsuji (1996). Attractive quality and must be quality. In J. D. Hromi (Ed.) The best on quality (Vol. 7). 165-186. ASQ Quality Press, Milwaukee, Wisconsin.

[13] Akao, Y. (1988): Quality Function Deployment QFD: Integrating Customer Requirements into Product Design. Productivity Press, Portland, Oregon.

[14] Helferich, A., G. Herzwurm, S. Schockert (2005): QFD-PPP: Product Line Portfolio Planning Using Quality Function Deployment. In Obbink, H., Pohl, K. (Eds.) Proceedings of the 9th International Software Product Line Conference, Rennes. 162-173. LNCS 3714, Springer, Berlin.

[15] von der Maßen, T., H. Lichter (2004): RequiLine: A Requirements Engineering Tool for Software Product Lines. In van der Linden, F. (Ed.) Software Product-Family Engineering, 5th International Workshop, PFE 2003, Siena. 168-180. LNCS 3014, Springer, Berlin.