Fraunhofer Institut
Experimentelles
Software Engineering

A Quantitative Model of the Value of Architecture in Product Line Adoption

Authors:
Klaus Schmid

Submitted for publication to
ICSE

In part supported by
Eureka ?! 2023 Programme,
ITEA project ip00004, CAFÉ

IESE-Report No. 103.03/E
Version 1.0
June 2003

A publication by Fraunhofer IESE

Fraunhofer IESE is an institute of the Fraunhofer Gesellschaft.
The institute transfers innovative software development techniques, methods and tools into industrial practice, assists companies in building software competencies customized to their needs, and helps them to establish a competetive market position.

Fraunhofer IESE is directed by
Prof. Dr. Dieter Rombach
Sauerwiesen 6
D-67661 Kaiserslautern

Executive Summary

Product line adoption is a key issue in product line development, as the right adoption approach is central to the overall success of product line development. Thus, this is a strongly discussed area of product line engineering. While so far, guidelines and experiences on the best approach to product line adoption have been presented, no detailed quantitative model was provided.

In this paper we present a quantitative model of the product line adoption problem. From this model we deduce general guidelines for product line adoption, particularly highlighting the role of the architecture in the cost-effective adoption of a product line.

Table of Contents

1 Introduction

While product line development is well recognized as a systematic approach to gain the benefits of large-scale reuse, the specific approach of migrating to a product line is still under discussion. The major opposing views in this context are the approaches of *proactive* and *reactive* product line adoption, as introduced in [2]. A simlar distinction is made in the *incremental vs. big-bang* development distinction [9] or the *heavyweight* vs. *lightweight* distinction of product line development [5].

While in general arguments have been exchanged that are in favor for both of the positions, so far no agreement on the respective benefits and no detailed analysis exist. Some exceptions are an experimental comparison of the two approaches [6] and some qualitative guidelines [9].

These shortcomings lead us to search for a quantitative model of product line transition. This model aims at the analysis of the respective benefits of the proactive and reactive approach and the constraints under which they can be accrued. While the overall costs of product line development are determined by a large number of factors [3], in this model, we focus specifically on the impact of modularity in a product line architecture on product line adoption as it plays a key role in the discussions.

In the following section, we will discuss in more detail the problem of product line adoption and how we will model this in this paper. In Section 3, we will briefly illustrate an example situation, that highlights our main concerns and describes the original context in which this model was developed. The core part of our contribution is given in Section 4, where we describe our valuation model for the product line adoption case. In the following section we will discuss the implications of our model for guidelines on product line adoption. Finally, in Section 6 we will conclude.

2 The ADOPTION MODEL

While commonly the potential product line approaches are discussed in terms of their extremes (e.g., the proactive vs. reactive dichotomy), these are simply a short-hand for a continuum of different approaches. This variation dimension can be characterized as the degree to which up-front development investments are made. While this is a continuum, there is one medium position, which considerably differs if compared to the other approaches. This position is proactive architecting (i.e., structuring of the software), combined with a reactive implementation approach.

Thus, we see three main approaches for dealing with variabilities, that must be distinguished in this paper:

1. If a variability is not immediately needed for any product, it is not taken into account as part of the product line infrastructure. This corresponds to the traditional reactive approach.

2. The medium position corresponds to a light-weight, architecture-centric approach. If a variability is not needed right away, the architecture should be developed in a way that takes this variation into account (see below), even if no implementation is yet provided for the variability.

3. The proactive extreme is a full analysis of the potential variabilities that are known and providing architectural measures in order to accommodate these variabilities as well as implementing the variabilities that are identified by product line scoping as part of development [8].

While the various positions are just points in a continuum, they provide pronounced key positions which we will use as the basis for our analysis. Moreover, each of these decisions must be performed for each of the potential variability points, providing us with an enormous decision space. However, as the same analysis apply for all of these individual decisions, we will focus here on deciding on the realization of a single variability.

At this point we need to describe in more detail what we mean with taking a variation into account during architecting. The development of an architecture and the availability of an architecture provide many different value contributions to a product line project. Here, we will focus on the value of *modularity* [10]. In particular for variabilities we will assume that taking into account a variability in the architecture will reduce the number of points in the software, where a varia-

tion mechanism is needed for representing this variability as opposed to an ad-hoc approach.

We define the following terms in order to describe the impact of variability:

> *Variation point (VP)*— a variation point corresponds to a consistent change in a software product due to a variability. This is independent of the number of positions in the implementation that are affected.
>
> *Variation impact point (VIP)* — a variation impact point denotes a position in the realization (independent of the binding time) where a variation point takes effect.

Consequently, there is a 1:n relationship between VPs and VIPs. An example would be a variability where web-interfaces as well as window-interfaces could be relevant to a product line. This would correspond to a single VP, but surely a large number of VIPs in the documentation, different parts of the code, etc.

Hence, we can describe our fundamental assumption as:

> Taking a VP into account during architecting will lead to a smaller number of VIPs than if it would not have been taken into account.[1]

As a basis for our model we will distinguish between the implementation of a capability *cap* and of the variability mechanism(s) *vm*. The capability in the example above could be a window-interface. This can be regarded as a single implementation, independent of the number of VIPs relevant to integrating it into the system. On the other hand, we will need one *vm*-implementation per VIP. Thus, we will regard the effort for capability implementation as independent of the software architecture, while the effort for *vm*-implementation will be proportional to the number of VIPs.

In the following section, we will discuss the cost implications of this assumption.

1 There are typically a large number of VPs in a product line. It will usually not be possible to minimize the number of VIPs simultaneously for all VP.

3 The Example

The initial motivation for this analysis came from a specific industrial project aimed at setting up a new product line in an organization. The actual parameters we are using in our discussion are derived from this example.

In this project we faced the major issue that a certain capability of the system (in our case: distribution) could not be excluded from the variabilities (meaning: while all foreseeable products would be localized, it could be possible that some future product would require to be distributed). Of course such a capability would have a strong impact on the architecture and would require a lot of added implementation effort (both for implementing the basic capabilities as well as for the variability mechanisms). The key questions we were facing were:

- Should we fully develop the necessary variability (proactive)
- Should we ignore it until a product that requires it must be built?

These and other options were evaluated. Finally, we decided to spend significant effort to ensure that the product line architecture would support the requirement of distribution (as a variant), but we delayed the implementation until such a functionality would actually be required.

The value model we describe below formalizes this situation as a basis for supporting this decision.

4 The Value Model

The key criterion we use in order to determine the optimal product line transition approach is the value a specific alternative generates or, in case this can not be resolved without uncertainty, its expected value. In this section, we will describe the basic formulas and assumptions we used for developing our value model. As our key value driver, we will use in this context a model of the overall development effort. Along the way we will discuss some conclusions that can be drawn from this model.

The key questions that the model is supposed to answer are:

- As we expect thorough architecting to reduce on average the number of VIPs for each variability, we need to ask: how does the overall cost vary as a function of the number of VIPs?
- How do costs vary depending on the probability that the capability is needed at all?

In order to answer these questions we need to introduce the following parameters.

4.1 Model Parameters

Implicit in these questions is the notion of time. Whenever we ask whether something will eventually pay off, we need to take into account the time we are looking ahead. Additionally, we need to take into account how often systems will be build during this time and a discounting rate in order to describe to what degree money is depreciated over time (this is called discounted cash-flow analysis [7]). This leads us to the following definitions:

- Number of years taken into account: y.
- Number of systems developed during this time: N.
  (we will use n to denote the number of systems that have been developed up to a certain time).
- Discount rate r.

In addition we will use the following parameters in order to describe the costs associated with the development of a variability implementation:

- Effort for an implementation of the capability (this is needed only once per product line): EC.

- Number of variation impact points for the variability in the overall infrastructure: *vip*.
- Effort for the design and implementation of the variability mechanism:[1] *Evm = Evmd + vip* Evmi*.
- In order to take into account that a specific variability might be needed not in the first system, but in a later one, we denote the probability that it is needed in a system by *p*.

Besides the basic costs of implementing the variability we also need to take into account that the maintenance of more complex code will usually lead to higher effort [4]. This complexity is increased if we add a lot of variabilities early on:

- We will use the factor *cc* (complexity cost) to denote this cost contribution that stems from the addition of variability. We will use the formula *Scc = cc*vip* to denote the cost from added complexity in developing a system.

If a variability is implemented at a later point in time (in the reactive mode), initial costs are saved, however, a different type of costs will come up at a later point in time:

- Changes that are performed at a later point in time, typically require a higher amount of effort (as this involves design changes, removing code, etc.) [1]. We call this the cost overhead *co*. *co* gives the added cost over the implementation of the variability mechanisms as part of initial development.
- If variability is added at a later point in time, the already developed systems require specific handling (e.g., adapting them to the existing infrastructure). This added cost is defined per VIP and per already existing system. We call it cost retrofit: *cr*.

With the above definitions we are able to compare the implications of the different costs for the various product line entry scenarios.

4.2 The Cost of Variability

In order to determine the cost of introducing the variability (*EVI*) into the product line there are two main cases: either we introduce it in the beginning (at *n=0*), or we introduce it a later point in time (*n>0*):

1. (n=0) *EVI(n) = EC + Evm*

1 While design costs can be expected can be expected to be independent of the number of occurrences, this will not hold for the implementation costs.

2. (n>0) *EVI(n) = EC + Evm * co + cr * n * vip*

The last term in the formula above captures the amount of effort required for retrofitting the existing infrastructure to reflect the updated variability handling mechanisms. As this is also a process over time, we also need to take the discounted cash flow approach into account. This is denoted as *EVI'(n)*:

3. *EVI'(n) = EVI(n) * (1-r)^(n/N*y)*
   (We assume the systems are evenly distributed over time)

Using these formulas, we can now analyze how the cost of variability introduction will vary depending on the specific time of variability introduction and on other parameters, like the number of *vips* that are required for this variability.

This is shown in Figure 1. Here, we used a scenario with the following parameters (y =5, r =10%, EC = 10 PW, Evm = 2PW + vip*0.2PW, cr= 0.2, co=0.4, N=10) [PW = person-week]. The numbers in this scenario have been taken from estimates from a real project, which provided the initial motivation for this model.

From Figure 1 we can see that the number *vip* has a strong impact on the overall costs of variability. In particular, we can see that the difference in costs for different *vip*-numbers is even increasing with larger numbers of systems that are built. We can easily understand the underlying functions qualitatively and how they lead to this effect:

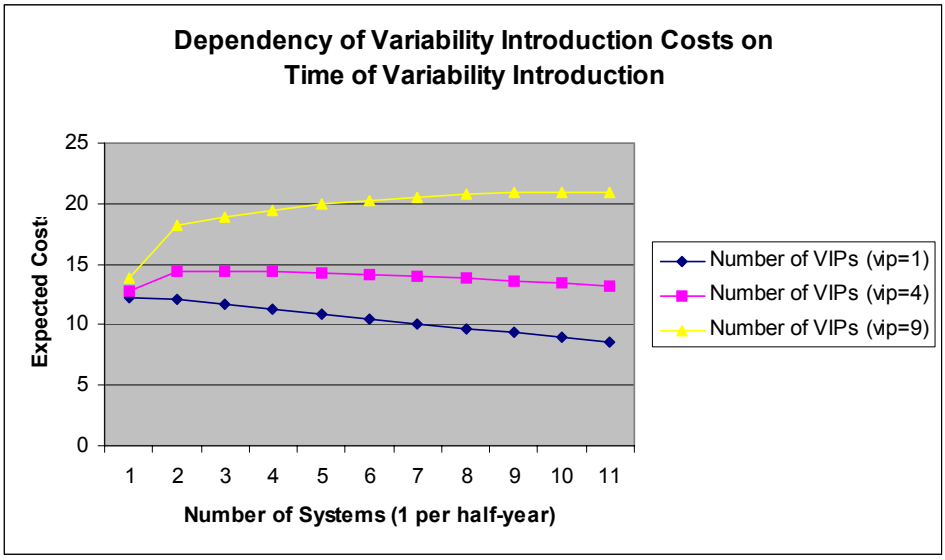

Figure 1: Variability Introduction Costs

The functions state that the number of VIPs has not a large impact if the variability mechanisms are implemented right away (cf. eq. 1 and 2). The gap that occurs to the second value for each of the graphs is motivated by the second term in equation 2. Finally, the third term leads to the fact that the maximum occurs at a large number of systems for larger values of *vip*. The fact that the values are actually going down for larger numbers of systems is motivated by the discounted cash-flow effect, which is introduced by equation 3.

While we captured in the above equations the costs from a late introduction of variability, the savings are not adequately represented as we neglected the costs of handling the more complex infrastructure that is created if we introduce the variabilities earlier.

Using *Scc,* as defined above, we can describe the costs that are incurred during the development of system *n* by an earlier introduction of variability as follows:

4. Scc'(n) = Scc * (1-r)^(n/N*y)

However, we need to take into account the cost for all systems after the introduction of the variability. This is given below:

5. $Tcc(n) = \sum_{i=n}^{N} Scc'(i)$

Consequently, if we introduce the variability with system *n*, then the total costs incurred by introducing the variability are given by:

6. *CV(n) = EVI'(n) + Tcc(n).*

The correction provided by *Tcc(n)* is very small for realistic values of *cc* (e.g., 0.05). The effect of this is shown in Figure 2. The main difference to Figure 1 is that also for a variability introduction for an earlier system the number *vip* has a stronger influence on the total costs.

From the arguments put forward so far, we can see that the number of changes (*vip*) that need to be made is a key driver for the overall costs and thus the preferability of a specific adoption approach. However, even if we know that the costs for a late introduction of a variability are very high, we will usually not implement it if it is very unlikely that we will need it. We will now turn to the impact of the probability of the need of the functionality on our total costs.

4.3 Modeling The Stochastic Process

While so far we focused purely on what happens if the variability is introduced at a certain point in time, we can now turn to the question, when will it actually be introduced?

If we assume that the variability is introduced as soon as it is needed, but not earlier, we can focus on the question: when will we need this variability?

As this is usually not known exactly for later products, it must be modelled as a stochastic process, i.e., for each new product in the product line, there is a certain probability p, that this product will need the additional capability introduced by the variability.[1]

In mathematical terms we are now asking for the expected value of the total variability costs *E(CV)*. This is given by the following formula:

$$7.\quad E(CV) = \sum_{i=1}^{N} CV(i) \times (1-p)^{i-1} \times p$$

We can now determine our expected total costs based on the varying parameter *p*.

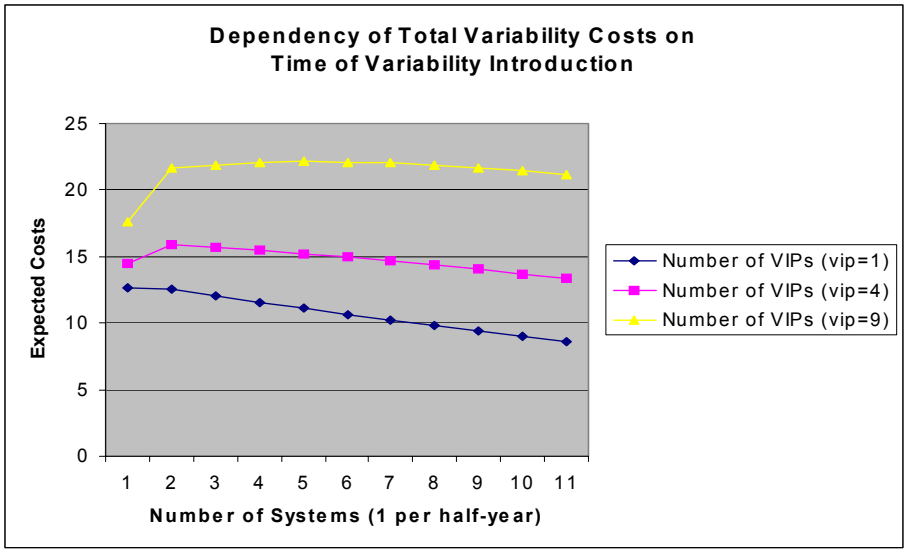

Figure 2: Total Variability Costs

1 Note, that different functionalities in the product line may have strongly different values of *p*.

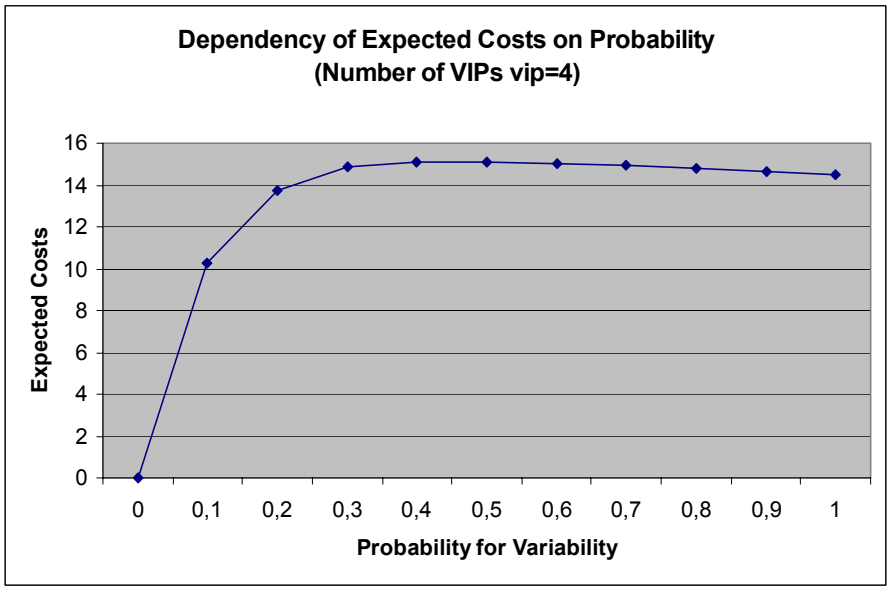

Figure 3: Cost / Probability Relationship

An analysis of this dependency is given in Figure 3. There we related the expected costs with the probability that we will need the added capability. In this figure, we used *vip=4*, but for other values of *vip*, we will get on a qualitative level the same results: the costs rise sharply, as the introduction of the functionality becomes more probable; beyond a certain threshold the costs are slightly reducing as now the probability is very high that the variability is introduced very early and the "low-cost" up-front development solution takes a stronger effect.

Thus, we can conclude that a delayed variability implementation is only beneficial from a total cost perspective if the probability of the introduction of the variability is very low (0.2 or smaller in our example).

Based on this model, we our now able to compare the situation of introducing a new variability at need with the up-front introduction of the variability and relate it with the two parameters we found particularly influential in our model: the probability *p* and the number of VIPs *vip*.

This is shown in Figure 4. This graph compares the cost figures for different numbers of modules based on different probabilities of introduction with immediate introduction. (Note, that immediate introduction corresponds in our model with a probability of 100% that we will need this capability and the variability in the first product.)

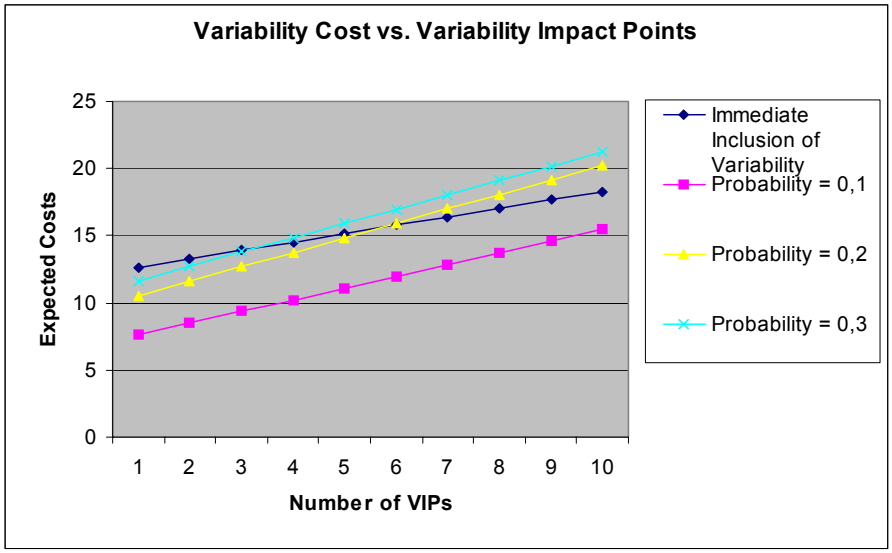

Figure 4: Immediate vs. Delayed Variability Introduction

We see here that the number of modules has a much stronger impact if we introduce the variability on a per-need basis than if it is introduced right away. Moreover, we see that even for low probabilities (e.g., 0.2) the cost for a per-need introduction can become larger than for immediate introduction, if a large number of VIPs is impacted.[1]

1 Note, that *vip* denotes the total number of positions in the implementation (including documentation) impacted by a variability, so our upper bound of 10 in the diagram is actually rather small.

5 Product Line Introduction Strategy

While, of course, the specific numbers that are given in our diagrams relate to the specific base numbers we choose, the qualitative view of the diagrams is independent of these concerns. For example, the specific point of intersection in Figure 4 between the immediate implementation and the expected value for *p=0,2*, which is a *vip* of approximately *5*, will vary depending on the specific values we select. But, the fact that there will be an intersection will always remain. This is why we focused in this paper purely on qualitative interpretations.

Based on the model we defined in the preceding section, we are now able to return to the core question of this paper:

> Which approach to building the product line infrastructure should be used (proactive, reactive, anything in between) and which criteria determine the selection?

As we saw in Section 4, the two parameters:

- Number of places impacted by the implementation of a variability (*vip*)
- The probability that the variability is actually needed (*p*)

have a major impact on the total cost of the product line introduction approach. Thus, they play a key role when deciding which approach to choose in a specific situation.

In general, we identify from Figure 4, that if the probability that the variability is actually needed is rather low, it is of course preferable to be reactive, i.e., implement the variability only on a per need basis. For higher probabilities of need of the feature, the balance usually shifts very soon. This shift is mostly driven by three parameters: *co*, *cr*, and *cc*. The higher the ratio *cr/cc* the more preferable it is to develop the variability initially. On the other hand *co* has a strong impact on the gap that occurs directly from the immediate implementation to any later implementation of the variability mechanisms. Here, we choose a value of 0.4 (implying there is a 40% overhead for late implementation) which is probably at the upper range of the spectrum. Choosing smaller values will result in a higher preference for a reactive approach.

At this point it is time to revisit our key assumption: a good product line architecture leads to an improved encapsulation of the variability. (This is of course no far-fetched assumption, as we will expect a good software architecture to encapsulate all concerns of central relevancy.) In Figure 5 we provide a zoomed

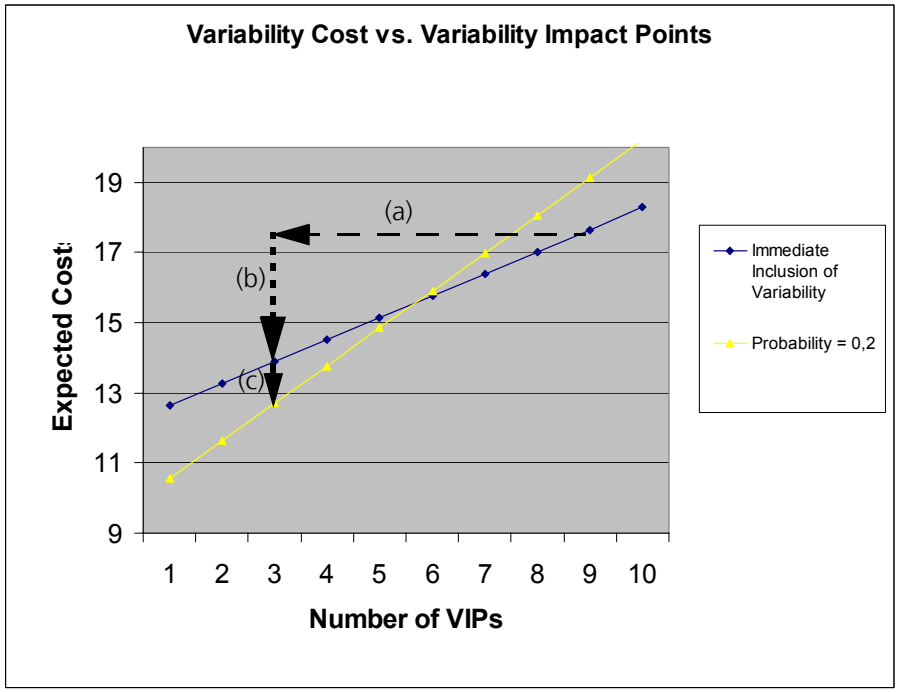

Figure 5: The Impact of Modularization in a Product Line Architecture

version of Figure 4, where we focus on the intersection of the curve for immediate variability implementation and the curve for probability *p=0.2*.

If we assume that a straightforward implementation which would take a specific variability—let's call it *v*—not into account would lead to a *vip* of 9, then the optimal approach would be the up-front implementation of the variability (even if the probability that it is actually needed is rather low, as with *p=0.2*). It is important to note that we made the assumption that in purely reactive development no *up-front* restructuring for a specific variability is made, which is in line with the general interpretation [2].

On the other hand, if we take this specific variability into account during architecture development, this will lead to a reduction of *vip*. In Figure 5 we assume a reduction to a vip of 3 (cf. arrow *(a)*). This restructuring would have a strong impact on the (variability) costs. In our example it would relate to a reduction of approximately 3 PW *(b)*, which corresponds to a saving of about 15% of the total cost. A saving that could be completely spent on architecting without shifting the balance. This architecting would have an interesting side-effect. For a *vip* of 3, the reactive implementation is actually more beneficial *(c)*. It would lead to the additional saving of about one person-week.

So, in this case the optimal strategy would actually be what we described as the intermediate or evolutionary strategy: perform a strong modularization through the development of a product line architecture that takes the variability into account, but perform the actual implementation with all the associated overhead costs only in case the variability is actually needed.

In our example situation (cf. Section 3), we estimated *p* to be somewhere between 0.1 and 0.2. At the same time the capability implementation would be rather expensive and all initial architecture sketches lead to a high value of *vip* for this variability. In accordance with this analysis we decided in our example (cf. Section 3) to use the strategy exemplified in Figure 5: we spent considerable effort on the development of a product line architecture, which can easily support this variability (i.e., with a small number of *vip*), but we did not implement the variability mechanisms nor the capability so far. This will only be done when and if the variability is actually required by a product.

6 Conclusions

In this paper we discussed a model forselecting a product line transition approach. The model development was driven from an actual industrial adoption situation and enabled us to discuss systematically the parameters impacting this decision. As we found in our discussion (and could be expected), no general answer can be given on the preference for a specific approach (proactive vs. reactive). This emphasizes the need for any product line manager, making these decisions, to have an informed opinion on the parameters that apply in his setting. Using this model a detailed analysis can be provided in a straight-forward manner.

Nevertheless, some general guidelines could be identified:

- Late implementation of a variability seems to pay off (over a large product line) only if the probability that the variability is required is very low.
- In general an up-front implementation of a variability seems to be the least costly approach if the probability that the variability is actually required by a product is not too low.
- The development of a product line architecture that modularizes the variabilities seems to play a key role in lowering overall development costs.

Of course, there are usually other aspects, like the availability of resources, that will have an impact on the decision whether to use a proactive, a reactive, or an intermediate approach.

According to our analysis, if the probability for a variability is low, it is a key strategy to proactively consider the variability in the architecture, while performing the actual implementation purely reactively, a mixed approach that — to our knowledge — has not been described so far. This strongly links architecting to scoping, as it implies that in general it seems to be the most effective approach to proactively scope the product line, perform the development of the architecture on the complete scope, but then adequately select those variabilities that are needed with a certain minimum probability for implementation.

7 References

[1] Barry W. Boehm, Chris Abts, Winsor A. Brown, Sunita Chulani, Bradford Kendall Clark, Ellis Horowitz, Ray Madachy, Donald J. Reifer, and Bert Steece. *Software Cost Estimation with COCOMO II*. Prentice Hall PTR, 2000.

[2] Paul Clements and Charles Krueger. Point — counterpoint: Being proactive pays off — eliminating the adoption barrier. *IEEE Software*, July/August 2002.

[3] Paul Clements and Linda Northrop. *Software Product Lines*. Addison–Wesley, 2001.

[4] Stephen G. Eick, Todd L. Graves, Alan F. Karr, J.S. Marron, and Audris Mockus. Does code decay? assessing the evidence from change management data. *IEEE Transactions on Software Engineering*, 27(1):1–12, 2001.

[5] John D. McGregor, Linda M. Northrop, Salah Jarrad, and Klaus Pohl. Initiating software product lines. *IEEE Software*, 19(4):24–27, 2002.

[6] Dirk Muthig. *A Light-Weight Approach Facilitating an Evolutionary Transition Towards Software Product Lines*. PhD thesis, University of Kaiserslautern, IRB Verlag, 2002.

[7] Klaus Schmid. An initial model of product line economics. In Frank van der Linden, editor, *Proceedings of the Fourth International Workshop on Product Family Engineering (PFE-4), 2001*, volume 2290 of *Lecture Notes in Computer Science*, pages 38–50. Springer, 2001.

[8] Klaus Schmid. A comprehensive product line scoping approach and its validation. In *Proceedings of the 24th International Conference on Software Engineering*, pages 593–603, 2002.

[9] Klaus Schmid and Martin Verlage. The economic impact of product line adoption and evolution. *IEEE Software*, 19(6):50–57, July/August 2002.

[10] Kevin J. Sullivan, William G. Grisworld, Yuanfang Cai, and Ben Hallen. The structure and value of modularity in software design. In *Proceedings of the Joint International Conference on Software Engineering and ACM SIGSOFT Symposium on the Foundations of Software Engineering, September 2001, Vienna*, 2001.

Document Information

Title:	A Quantitative Model of the Value of Architecture in Product Line Adoption
Date:	June 2003
Report:	IESE-103.03/E
Status:	Final
Distribution:	Public