Summarizing the poster presented
at ICSR-8, Madrid, July, 2004

Managing the Evolution of Software Product Lines

Peter Knauber
Mannheim University of Applied Sciences
Windeckstraße 110, 68163 Mannheim, Germany
p.knauber@fh-mannheim.de

Abstract. Software product lines are a very promising approach to software reuse. Because all product line members are derived from the same assets, product lines are very sensitive to evolutionary changes of these assets. Wrong decisions on how to handle these changes may lead to serious technical or economic problems that may even result in a failure of the whole product line. On the other hand, there are certain kinds of changes that product lines can cope much better with than single systems can.
The paper categorizes software evolution into different kinds and gives guidelines on how to handle these, depending on their sources and (potential) effects.

1 Introduction

Software product lines are a very promising approach to software reuse, they have proven their applicability in a broad range of situations and produced impressing results.

A software product line is a collection of software products sharing a common set of features that address the specific needs of a defined domain. These products can be derived from a common product line infrastructure or, in other words, from a common managed set of assets. In order to accomplish this, variability is key, implying that the assets that comprise the product line infrastructure are variable enough to be adaptable to different environments, user groups etc.

Due to the variability incorporated in the product line assets, the product line is able to cope with certain kinds of changes better than single systems. Nevertheless, evolution has its effects on all applications, whether they are derived from a product line or developed as single system.

The term software evolution, as used in this paper, can be characterized as the result of many (typically small) changes over time, as defined by [10][10] and used in [6]. There are two main sources of such changes:

- Sometimes a product or a product line is evolved *proactively*. This is usually the consequence of a respective strategy for the (market) introduction of a product or a product line. For single system development in this situation, a basic product is developed first that can afterwards be extended according to customer reactions and requirements. For product lines, usually an incremental adoption strategy [2] is chosen in this situation.

- The alternative to proactive development is *reactive* development. This happens to most successful systems that have a reasonable life time because over time, e.g., requirements start to shift, the products are used in new, unforeseen environments or they are combined with other, new software tools. In this situation, the "many small changes" are usually performed as maintenance tasks, either for corrective, adaptive, perfective, or preventive purposes [8].

Section 2 of this paper describes the different effects of proactive and reactive evolution on product lines vs. single systems and discusses the different sources of evolution. Section 3 gives guidelines on how to handle the different categories of reactive evolution in a product line context. Section 4 summarizes the paper. References to related work are given in the context of the respective topics.

2 Software Evolution

This section describes different kinds of software evolution and their respective sources with the focus on product lines.

2.1 Evolution of Single Systems vs. Evolution of Product Lines

The term *evolutionary* (of incremental) *development* is well-known from single system development. The idea is to either evolve a (initially rudimentary) system over time until it satisfies the requirements that often were not well-defined in the beginning or to add new components with additional functionality over time, after the initial product has been carefully integrated into a certain environment [15]. The evolution of single systems differs in at least two main aspects from the evolution of product lines:

- Typically, single systems are developed evolutionary in situations where not all requirements are known at the beginning of the development (this is also called evolutionary prototyping). Contrary to this, in the beginning of product line development a reference architecture for the infrastructure (or, at least, a predictable part of it) is created with the intention to realize it in increments over time. Contrary to single systems, the final state of the infrastructure (as far as it can be anticipated) is defined rather early. This helps to avoid typical maintenance problems often caused from the numerous changes and the resulting structural degradation in the case of single systems.
- When single systems are developed evolutionary (or incrementally), their functionality is extended step by step. In contrast to this, the term *incremental product line development* denotes a step wise extension or improvement of the infrastructure as it has been defined early in the product line life cycle, whereas the complete functionality of at least some initial products is already in place. Typically at this early phase of the life cycle, some commonalities are realized as result of separate, i.e., redundant, developments. Later on, during the development of new products, the infrastructure is extended until the potential of reuse has been reached.

These differences should illustrate that evolutionary development of single systems is not comparable to that of software product lines.

2.2 Proactive Evolution

Talking about *proactive evolution* of a (whole) product line, that is, not only of single product line members, implies that most or even all products should benefit from the evolution. This can be achieved by generalizing or extending the product line infrastructure in order to support additional requirements (*infrastructure extension*) or by *improving the infrastructure* in order to make the derivation process more efficient (examples are better documentation or higher automation, e.g., developing application generators). In practise, a combination of these two categories is most often useful.

Due to the effects of proactive evolution, software product lines can cope better with some kinds of future changes than single systems: during their development potential future requirements have been anticipated and the product line has been designed in a way that the more likely ones can easily be integrated.

2.3 Sources of Reactive Evolution

As sketched in the introduction, the evolution of software can be characterized as the result of many small changes over time. Talking about reactive evolution, these changes usually happen during maintenance of the software (product line). The [8] defines four categories of maintenance tasks:

- Corrective maintenance. Reactive modification of a software product performed after delivery to correct discovered problems.
- Adaptive maintenance. Modification of a software product performed after delivery to keep a software product usable in a changed or changing environment.
- Perfective maintenance. Modification of a software product after delivery to improve performance or maintainability.
- Preventive maintenance. Modification of a software product after delivery to detect and correct latent faults in the software product before they become effective faults.

Because corrective and preventive maintenance deal with actual or potential faults in the software, they usually do not have large-scale effects on its overall structure. This is different for the other two categories: adaptations in order to fulfill changing requirements or to make use of new (e.g., component) technology can result in large-scale modifications of the software. The same holds for perfective maintenance which includes tasks like refactoring and restructuring of the software in order to improve its future maintainability.

Software product lines are perfectly suited to deal with changes — if the variability needed to support these changes has been built in. This in turn depends on different factors:

- Was the type of change planned when the product line was architected/designed?
- Was the type of change foreseeable and, if so, was it considered to be likely enough to be incorporated into the software?
- If the type of change was anticipated, was it feasible from a cost, resource, and time-to-market perspective to incorporate it into the software?

Beyond foreseeable changes, there are often enough changes in laws, domains, or technology resulting in adaptive or perfective changes that are impossible to predict at the time when a new product line is planned. [15] distincts three categories of reasons for these changes:

- Functional reasons. Members of a product line have to be extended with new functionality, e.g., in order to support new data exchange formats or as a reaction to functionality extensions in competitor's products.
- Nonfunctional reasons. Non-functional changes often result from new or evolving technology (software or hardware). For example, faster processors, new development approaches, or changing standards for documentation can cause changes in the product line assets.
- Domain reasons. In order to support products within certain application domains, product lines assets have to react immediately to changes within these domains. Examples for such changes are new and innovative procedures, new technical developments, or changing governmental rules.

For product lines, there are two principle ways to deal with reactive evolution:

- The product line infrastructure is generalized, extended or improved in order to support the changes and the desired products are derived afterwards. This way, all products (as far as they are concerned) benefit immediately from reuse, respectively all future product line members implement the changes at once.
- One or more *reference products* are developed (evolved) first, based on the existing assets but changes are made in product-specific form and the changes are (re-) integrated into the infrastructure afterwards. Usually, changing requirements can be supported faster this way but less products benefit from reuse and there may arise problems when trying to integrate the product-specific changes into the infrastructure.

In theory, the first alternative is optimal because all future products immediately benefit from infrastructure reuse: as soon as new product line members are derived from the infrastructure they are "automatically" based on the extended functionality. Unfortunately, there is no direct advantage for existing products because, if the respective functionality is relevant to them, they already contain that functionality, developed in a product-specific way. Only future releases of these products will benefit from reduced effort for maintenance and evolution. Typically, organizations implementing this first strategy have relatively large domain engineering departments compared to their product departments [3].

In many cases, the first alternative is not feasible and organizations implement the second alternative where development of the product line infrastructure is typically done as part of the development of a new product. This way, a part of the development effort that is spent for a new developed product anyway and "only" the effort for making the new functionality reusable has to be spent additionally. This strategy is appropriate in situations where there is not enough time or money to build up the infrastructure before product development or if there exists a *reference customer* who has functional requirements that will usually later on be asked for by other customers as well.

Table 1 shows different sources of evolution and possible strategies to handle them. Filled bullets indicate recommendable strategies; strategies with non-filled bullets in the respective field may be appropriate under certain conditions; the remaining combinations should not be applied.

Sources of evolution		First development of product line infrastructure, then product development	First product development, then product line infrastructure development: Development of a reference product	First product development, then product line infrastructure development: Development of more than one product in parallel
Proactive, preplanned evolution	Infrastructure extension	●	○	
	Infrastructure improvement	●	○	
Reactive, unplanned evolution	Functional reason	○	●	●
	Non-functional reason	●		
	Domain-based reason	●	○	

Table 1: Sources of product line evolution and strategies for handling them

3 How to Manage Product Line Evolution

It is relatively easy to handle anticipated evolution of a product line: respective *variation points* [4] are planned and realized during infrastructure development. Later on, these variation points are used to add or remove optional functionality or to add new alternative variants depending on their type and binding time built in.

Compared to this situation it is much more difficult to deal with unforeseen changes of requirements. The following subsections address related problems and recommendable strategies to manage them from different angles.

3.1 Feature Shift

A clear tendency shows that new and often expensive features, that initially are only shipped with high-end products, over time become standard features. This results from the fact that development organizations try to get a unique selling proposition for their high-end software by offering such unique features that are not available with products of competitors.[1] As soon as the respective functionality becomes available in competitors' products, these features start loosing their important role and shift towards core functionality. The high-end products can be seen as kind of reference products for new features that will later on be integrated into the product line infrastructure or the new functionality might even be implemented as part of the infrastructure before it is

1. In the mobile phone or the automotive market it is very easy to observe this trend.

shipped for the first time with a high-end product. Being available with the infrastructure, the new functionality can be reused by all products directly.

One could argue that it does make sense to integrate new functionality directly into the infrastructure because this is cheaper (they are already "reused" for the high-end products) and they are then faster available for upcoming products. But there is a major argument against this: Fancy new features are much more important in fast-moving and innovative markets than in stable and settled ones. But exactly in fast-moving and innovative markets it would be risky to invest into reusable product line assets, given that it is very hard to predict customer acceptance of new features. Additionally building reusable assets is more expensive in terms of money and time-to-market (certainly the more important argument in fast-moving markets!) than realizing the functionality in a product-specific way. If customers show interest in the new features they should later on be integrated into the infrastructure. If not, unnecessary investment has been saved.

3.2 Managing unforeseen Changes of Requirements

Figure 1 presents an overview over possible sources of unforeseen changes and appropriate reactions.

- In case of new or changed functional requirements it has to be decided if they affect one or few specific product(s) only or if basically all product line members are (or will be) affected. This decision should be based on the result of a product line scoping (*product portfolio definition* and *asset scoping*, [13]); accordingly the new functionality is implemented for one ore few specific products or it is integrated into the product line infrastructure.
- If the new (functional or non-functional) requirements stem from a domain with high reuse potential they affect the infrastructure and have to be integrated there.
- In case of new or changed non-functional requirements it has to be decided if many (or all) product line members are (or will be) affected or if the requirements are product-specific. In the first situation the infrastructure has to be adapted. In the second situation where only a single product is affected, it can be appropriate or even necessary to split this product from the product line: typically, non-functional requirements are not implemented by single product components but the respective code is spread over the whole product. In order to change such a non-functional requirements it is very likely that large parts of the product have to be modified. If later on a new release of that modified product is derived from the product line infrastructure, all modifications have to be repeated. Because this task is effort-intensive and error prone, it is usually preferable to split the product from the line and evolve it separately (or even start a new line with it).

3.3 Evolution and Reintegration

In situations where new or changing requirements have non-functional origins or where they originate in the domain, is it often preferable to evolve the product line infrastructure before the products. This results from the fact that such changes are often effort-intensive and that at least part of the effort has to be spent more than once if first some products and afterwards the infrastructure is adapted. In case of changing functional requirements it is often preferable to first try out the respective features in one or few

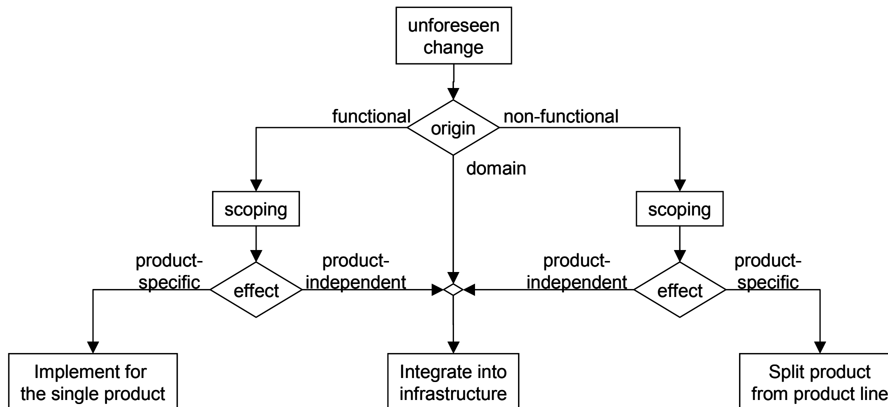

Figure 1: Handling unplanned requirement changes in software product lines

products in order to get a better understanding before they are integrated into the infrastructure (cf. Section 3.1).

Situations are rare where organizations have the option to react to changing requirements with an adaptation or extension of the infrastructure before new products are derived from it or before existing products are changed. Contrary to proactive evolution where infrastructure development and product development are at least loosely coordinated with each other in advance, in case of reactive evolution the market pressure due to competitors often forces the development of products and infrastructure in parallel.

One possibility to cope with that situation is to establish an infrastructure department in parallel to development departments. An alternative to this is to develop in cycles where the same development team first develops one or few products, then integrates their advanced functionality into the infrastructure before deriving more products from the new infrastructure release and so on.

According to economic aspects, [14] recommend the development of single reference products (*branch-and-unite*) especially in cases where new products are needed in sporadic intervals. They explicitly recommend to not develop more than one product in parallel (*bulk*) because their consolidation and the integration of parallel developments into the infrastructure is effort-intensive, makes it more difficult to judge the economic success of the product line, and may lead to quality problems. Contrary to this, [7] do recommend the development of several products in parallel, followed by explicit integration phases. In order to keep control, they restrict the development to certain layers in so-called *federated architectures*.

3.4 Measures to track Evolution

In order to keep the (reactive) evolution of a product line under control it is useful to establish certain metrics.

Many metrics that can be applied as needed, are documented for the development of single systems [11]. Beyond that, the Goal Question Metric approach by [1] allows to de-

velop new metrics depending on the specific context and goals of their application. So far, well-established metrics especially for product lines are not documented but there some suggestions can be found in literature (cf. [12], [5], [9]).

In general it can be said that metrics for product lines are not different from metrics for single systems, just the significance of attributes like maintainability, changeability etc. is higher. Interestingly enough, there are no metrics documented that explicitly address product line infrastructures.

There is another group of metrics which is interesting wrt. the evolvability of a product line: during product line scoping (potential) risk factors are identified and (if possible) quantified. Examples for such risk factors are the stability of a domain or the degree of variability among the members of a product line. These factors should not only be considered during the creation phase of a new product line but instead be monitored carefully over the complete product line life time. If they are quantified they can be used to indicate the need for activities to avoid or mitigate risks early on.

3.5 Reactive Evolution as Strategy

In principle, it is possible to choose consciously a reactive strategy to adopt and institutionalize a software product line. This leads automatically to unforeseen evolution of the product line and its infrastructure. While this approach avoids high initial investment as well as wrong decisions in which assets to invest, the benefits that can be expected from reuse are small (probably, synergy effects can only be expected from corrective and preventive maintenance). Instead, there is the risk that the infrastructure has often to be revised in order to provide an appropriate basis for current (and hopefully future) products.

In general, such an strategy can not be recommended, exceptions may be products in very small domains where it is clear from the beginning that future requirements will not match the current ones that would be implemented in the infrastructure. But that leads to the question if a product line strategy is at all appropriate in such a situation.

Choosing a reactive strategy also contradicts the perception of a product line as implementing strategic (contrary to opportunistic) reuse of software [5].

4 Summary

The paper motivates the difference between proactive and reactive evolution of software product lines. Whereas it is relatively easy for a product line to handle proactive evolution (basically, its infrastructure has been designed for this task), the management of reactive evolution is even more crucial than for non-product line systems because all product line members are derived from the same set of assets (also called product line infrastructure). Depending on the origin of changing requirements and on their (potential) effects, the paper gives guidelines on how to deal with assets and products affected by changing requirements.

References

[1] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. Goal Question Metric Paradigm. In John J. Marciniak (Editor), Encyclopedia of Software Engineering, volume 1. John Wiley & Sons, 1994

[2] Günter Böckle, Jesús Bermejo, Peter Knauber, Charles Krueger, Julio Cesar Sampaio do Prado Leite, Frank van der Linden. Adopting and Institutionalizing a Product Line Culture. 2nd Software Product Line Conference (SPLC2), San Diego, Florida, 2002

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

[4] J. Bosch, G. Florijn, D. Greefhorst, J. Kuusela, H. Obbink, K. Pohl. Variability Issues in Software Product Lines. In: Software Product-Family Engineering, Lecture Notes in Computer Science, Vol. 2290, pp. 13-21, Springer-Verlag, 2002

[5] Paul Clements, Linda Northrop. Software Product Lines: Practices and Patterns. Addison-Wesley, 2001

[6] Change management and evolution support. ESAPS Consortium Wide Deliverable, Siemens-WP3-010518, April 2001. Available at http://www.esi.es/en/Projects/esaps/public-pdf/CWD32-18-04-01.pdf

[7] D. Faust, C. Verhoef. Software Product Line Migration and Deployment. Software - Practise and Experience 2003; 33: 1-23

[8] ISO/IEC 14764: Software Engineering-Software Maintenance, 2000

[9] Susanne Johnsson and Jan Bosch. Quantifying Software Product Line Ageing, in Proceedings of 1st ICSE Workshop on Software Product Lines: Economics, Architectures, and Implications, P. Knauber, G. Succi (Editors), Limerick, Ireland, 2000, Fraunhofer IESE Report No. 070.00/E, 2000, available at http://www.iese.fhg.de/pdf_files/iese-070_00.pdf

[10] M.M Lehman. Laws of Software Evolution Revisited, EWSPT96, October 1996, LNCS 1149, Springer Verlag, 1997

[11] Manny Lehman. Rules and Tools for Software Evolution Planning and Management. Position Paper, FEAST 2000 Workshop, Imperial College, July 2000

[12] Mika Pussinen. A survey on software product-line evolution. Report 32, Institute of Software Systems, Tampere University of Technology, December 2002. Available at http://practise.cs.tut.fi/pub/index.html

[13] Klaus Schmid. Scoping Software Product Lines - An Analysis of an Emerging Technology. The First Software Product Line Conference (SPLC1), Denver, CO, USA, 2000

[14] Klaus Schmid & Martin Verlage. The Economic Impact of Product Line Adoption and Evolution, IEEE Software 19, 4; 2002

[15] Ian Sommerville. Software Engineering, Addison-Wesley, 2001