Fraunhofer Institut
Experimentelles
Software Engineering

Proceedings of the PLEES'01

International Workshop on Product Line Engineering: The Early Steps: Planning, Modeling, and Managing

September 13, 2001, Erfurt, Germany

Editors
Klaus Schmid
Birgit Geppert

IESE-Report No. 050.01/E
Version 2.0
September 13, 2001

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

Proceedings of the PLEES'01 International Workshop on Product Line Engineering: The Early Steps: Planning, Modeling, and Managing

Editors:

Klaus Schmid
Fraunhofer IESE
Sauerwiesen 6
D-67661 Kaiserslautern
Klaus.Schmid@iese.fhg.de

Birgit Geppert
AVAYA Labs -
Software Technology Research
Basking Ridge, NJ, USA
bgeppert@research.avayalabs.com

Program Committee:

Mark Ardis, USA
Günter Böckle, Germany
Jan Bosch, The Netherlands
Paul Clements, USA
Jim Coplien, USA
Krysztof Czarnecki, Germany
Ulrich Eisenecker, Germany
John Favaro, Italy
Cristina Gacek, United Kingdom
Birgit Geppert, USA
Peter Knauber, Germany
Henk Obbink, The Netherlands
David Parnas, Canada
Klaus Pohl, Germany
Klaus Schmid, Germany
Juha-Pekka Tolvanen, Finland

Table of Contents

PLEES'01: International Workshop on Product Line Engineering – The Early Steps: Planning, Modeling, and Managing

Klaus Schmid
Fraunhofer IESE
Sauerwiesen 6
D-67661 Kaiserslautern
schmid@iese.fhg.de

Birgit Geppert
Avaya Labs Research
Basking Ridge, NJ USA
bgeppert@research.avayalabs.com

In the last five years product line (PL) Engineering has become a major topic in industrial software engineering. It introduces a focus-shift from the development of single systems to the development of complete system families. This paradigm shift aims at the efficient and cost-effective development through large-scale reuse by exploiting the family members' commonalities and by controlling their variabilities. Reported results are indeed encouraging. These include error and effort reductions by a magnitude as well as strong time-to-market improvements by one third [5, 6, 7, 8].

The research focus until now has been on the technical, implementation-centered activities related to family-based development [1]. However, in order to be successful these activities need to be performed in an organizational and technical framework that is appropriately prepared to support them. This raises organizational issues, issues of PL planning, as well as issues concerning requirements modeling and management in the context of product lines. These more up-stream activities have not yet seen as much attention, but they are nevertheless critical for the successful introduction of a product line approach [2, 3].

In this workshop we focus on the issues that are particularly important when introducing a product line approach in an industrial environment. This discussion is centered around the following four key issues:

- Transition Approach
- Organizational Issues
- Product Line Management
- Variability Modeling

Transition Approach

When transitioning a running organization into a product line engineering organization, several issues must be considered that range from bridging technological gaps to solving people issues. Existing development units may not be organized to allow product line engineering, but follow an independent work style with design decisions made on a product by product basis. This leads to the duplication of featuresthat are developed by different units and for different products. An organizational re-structuring is necessary that takes into account a joint development of the common product line infrastructure and an accompanying re-assignment of responsibilities.
In most cases this transition can't be done in one step. Rather, a product line engineering approach must be introduced incrementally. This raises issues of

determining the best starting point in terms of products, functional areas, and organizational units. An important success factor in this transition is to convince and motivate people to follow the new organizational structure and accept the new assignment of responsibilities.

Organizational Issues

A key factor of successful product line engineering is an appropriate organizational structure of the software development organization. Several different organizational alternatives are possible, ranging from one single development unit that is responsible for developing and maintaining the product line infrastructure as well as deriving the single products, to a hierarchical structuring of the development units with the tasks for developing and maintaining the infrastructure as well as the single products distributed over several units [9, 10].

Identifying the organizational structure that is best suited for a given situation is a complex task that is only partially understood yet. There are many different factors that influence this decision process. Examples are the existing organizational structure, the potential for people to experience direct benefit from reuse, the strategic positioning of the product line in the market, or the size of the organization. But also technical aspects do impact the decision. For instance, the organizational structure needs to match the software architecture of the product line so that tasks and responsibilities can be assigned appropriately to the different development units. This is actually a requirement on both, the organizational structure as well as the software architecture. Consequently, both need to co-evolve.

So far, identifying and establishing the right organizational structure is not sufficiently understood. Thus, we see at this point in particular the need to further study existing and working product line organizations.

Product Line Management

When turning its attention to product line development, an organization faces important challenges that go all the way from the initial planning stage, through the early development, to the continuous evolution of the product line.

During planning the appropriate alignment with the overall product portfolio of the company and its long-term strategy needs to be established. It is already at this point that key decisions are made that will determine the overall economic benefit of the PL to the company. These decisions need to be performed in an integrated manner on several levels. First of all a commitment needs to be made on the specific systems that will be developed as part of the product line. From a strategic point of view, leaving inappropriate products out can be a key benefit to the organization, just like finding the right systems to include. Here, product line development directly interfaces with strategic management of the company. This interface needs to be appropriately managed to ensure success. Similarly, once the products have been identified the key assets that need to be developed for reuse must be identified. Again, identifying the right assets is key to the economic success of the PL. This kind of bounding decisions are in this extent unique to product line development and are addressed under the heading of scoping [4].

Once we identified the initial development plan for the product line, we need to put it into practice. This is non-trivial, as the various projects that are part of the product line need to be developed in an integrated manner as the shared asset base creates

important dependencies among the projects. Sustaining these links is crucial as otherwise the survival of the product line is at risk! For example, a disconnect between the development of the platform and the individual systems risks the overall economic benefits of product line development.
Finally, a PL needs to evolve over time. New products need to be integrated into the product line and old systems need to be phased out. This has to be supported by the development of appropriate reusable assets. Also the shift in the product line needs to be aligned with the overall strategy of the company. These topics have so far hardly been addressed. Product line evolution complicates in particular change management, as for any change decisions have to be made, about affected systems, actually changed systems, and so on.

Variability Modeling

A key principle of product line development is the codification and reuse of knowledge. In the end this boils down to the reuse of code components that are generically reusable throughout the product line. However, in order to enable and sustain this generic code development, the product line perspective needs to be pervasive throughout all artifacts across all life cycle steps. Be it requirements, design, test cases, we need to be able to describe relevancy to the product line as a whole as opposed to single systems. As not everything will be relevant to every system to the same extent, we need to make explicit the commonalities *and the variabilities* of the artifacts. While this is probably at this point the most well-studied topic within the scope of the workshop, it is still more of an art form than a daily practice in industrial environments.
At this point we are still lacking general approaches that can be applied to all forms of artifacts and that also support the efficient instantiation for product-specific situations. But even more important: how can we elicit the information that is required as a modeling basis in an efficient manner? We may explicitly not restrict ourselves to a system-specific focus as otherwise the resulting assets will be too narrow, but we may also not take on an everything-is-important attitude as this would lead to a waste of resources and an information overload that may hinder subsequent activities. Thus, a major question is: how do we focus beyond a system?

Summary

In this workshop we focus on the early steps in product line engineering, i.e., those steps that have not yet seen as much attention as the implementation specific tasks, but that are nevertheless critical for a successful product line engineering project. This comprises the transitioning of a running organization into a product line organization, issues of how to organize a development organization to best support product line engineering, tasks of managing a product line which ranges from the initial planning to an ongoing evolution, and finally the identification of a product line's reuse potential and its packaging into a product line architecture and accompanying reuse components. The intent of this workshop is to enable a discussion around the four aforementioned topics among practitioners and researchers in order to allow an exchange of industrial experience and academic results.

The proceedings of this workshop will be available as Technical Report from Fraunhofer IESE at http://www.iese.fhg.de or directly from the workshop organizers.

References

1. Software Product Lines: Experience and Research Directions; Proceedings of the First Software Product Line Conference (SPLC1). Patrick Donohoe, editor. Kluwer Academic Publishers, 2000.
2. J. Bayer et al. PuLSE: A methodology to develop software product lines. In Proceedings of the ACM SIGSOFT Symposium on Software Reusability, pages 122-131, 1999.
3. Paul Clements and Linda Northrop. A framework for software product line practice - version 3.0. http://www.sei.cmu.edu/plp/framework.html. Software Engineering Institute, Carnegie Mellon University, 2000.
4. Klaus Schmid. Scoping software product lines - an analysis of an emerging technology. In Patrick Donohoe, editor, Software Product Lines: Experience and Research Directions; Proceedings of the First Software Product Line Conference (SPLC1), pages 513-532. Kluwer Academic Publishers, 2000.
5. David M. Weiss and Chi Tau Robert Lai. Software Product-Line Engineering: A Family-Based Software Development Process, Addison-Wesley, 1999
6. James C. Dager. Cummin's Experience in Developing a Software Product Line Architecture for Real-Time Embedded Diesel Engine Controls. In: Software Product Lines - Experience and Research Directions, Proceedings of the First Software Product Lines Conference (SPLC1). Ed. Patrick Donohoe, pp. 23-46, Kluwer Academic Publishers, 2000.
7. Peter Toft, Derek Coleman, and Joni Ohta. A cooperative model for cross-divisional product development for a software product line. In Patrick Donohoe, editor, Software Product Lines: Experience and Research Directions; Proceedings of the First Software Product Line Conference (SPLC1), pages 111-132. Kluwer Academic Publishers, 2000.
8. Paul Clements, Cristina Gacek, Peter Knauber, and Klaus Schmid. Successful software product line development in a small organization. In Paul Clements and Linda Northrop, editors, Software Product Lines: Practices and Patterns, chapter 11. Addison Wesley Longman, 2001.
9. Jan Bosch. Software product lines: Organizational alternatives. In Proceedings of the 23rd International Conference on Software Engineering, pages 91-100. IEEE Computer Society Press, 2001.
10. Klaus Schmid. People issues in developing software product lines. In Proceedings of the Second ICSE Workshop on Software Product Lines: Economics, Architectures, and Implications, 2001.

Industrial Keynote — Feature Modeling in an Industrial Context

Alessandro Maccari

Nokia Research Center, Software Architecture Group
P.O. Box 407
FIN - 00045 NOKIA GROUP, Finland
Email: alessandro.maccari@nokia.com
Tel.: +35871800800

The topic of feature modelling has recently been at the centre of intense research work, particularly in the domain of telecommunication systems. A feature can simply be defined as a capability or value which the user is willing to pay for. Mobile telephones, in particular, are heavily feature-oriented products, and features drive product scoping. Mobiletelephone manufacturers have striven to diversify their products for different markets, user interface styles and user categories. The result is a product line where variability spans in several dimensions. The scene is rendered more complex by the fact that the links and interactions between different features are built into the user interface software. This is inevitable in all embedded, resource-constrained products. Since product lines are scoped mainly by mapping the various product features, the need of some kind of formal feature modelling emerges. At Nokia we tackled the problem with different approach angles, one of them being textual feature description.

Use case description seems to work best when done with the aid of textual tables. This renders use case descriptions easy to understand (even for non-experts), tool-independent and maintainable. We follow Alistair Cockburn's approach in use case modelling. I devised a textual template for feature descriptions on the wake of the one we use for use cases. The adoption of such a description method has the same advantages as the one for use cases, with the (relevant) additional one that people are used to such kind of descriptions, and little extra training is needed.

In this talk I discuss the current practice in our organization with feature modelling for product lines, and bring up a number of open research issues. In particular, I face the issue how to link feature description with product configuration. I also discuss the problems of maintaining feature descriptions for large and complex systems, such as mobile phones.

Guiding Feature Asset Mining for Software Product Line Development

Thomas Eisenbarth Daniel Simon

Universität Stuttgart
Breitwiesenstraße 20–22
70565 Stuttgart, Germany
{eisenbarth, simon}@informatik.uni-stuttgart.de

1 Introduction

Software product line architectures promise significant benefits over traditional architectures such as shorter time-to-market, shorter and cheaper development cycles, and higher exploitation of the reuse potential at hand. While the ideas and concepts of product lines are well suited for developing new products, it is not obvious if and how one can apply this technology in the presence of legacy software.

Migrating legacy software systems to a product line provides ways for extending and developing successful products and offers a chance to protect and preserve a company's former investments. The legacy artifacts have been designed under significantly different circumstances and typically for just one single application domain. Therefore turning legacy software into a software product line requires a new design aware of the old product's key assets—assuming reuse really pays. Consequently, reengineering efforts have to address a number of issues unique to product line development.

The product line development frameworks of Bosch [5] and the SEI [11] recognize a need for integration of existing assets. Bergey et al. [4, 12] discuss how to exploit the reuse potential by means of traditional reverse engineering methods and business process decisions in detail. Another effort integrating legacy assets into a product line is described by Bayer et al. [3] in the context of PuLSE [2].

Our goal is to develop techniques that respect the specific reverse engineering prerequisites of product lines. Recently, we introduced techniques that help to derive feature-component maps [9] by means of mathematically founded concept formation.

2 What's different?

A number of techniques have recently been developed that try to analyze legacy software by focusing on features of the software. They are all based on dynamic and static analyses as well as software metrics [6, 7, 8, 9, 10, 13, 14, 15].

When analyzing legacy source code in product line development context, some aspects that will be discussed in the following gain prominent importance.

Early indication whether further investigation will pay By looking at results of the early analysis steps over the system, early judgment about the lucrativeness of feature asset mining is enabled. Our techniques [7, 8, 9, 10] are incremental as they support judgment after each round whether it is benefical to take the next step or cancel the reuse effort.

No overall architecture recovery of the legacy software Our new analysis techniques need no complete recovery of the legacy code's architecture. Our techniques incorporate domain knowledge of both users and programmers rather than focusing on the source code only. Our method is opportunistic because it allows the product line engineer to analyze the old code just as far as needed. The support for partial analyses or quick and cheap ad-hoc decisions is an import aspect since complete results usually require much time. (Why should someone recover an architecture, just to decide he doesn't need it anyway?)

Analysis according to the legacy software's features We prepare legacy software for reuse in software product lines by (a) feature location and (b) connector recovery for handling the communication in the software artifact. On the one hand side, we try to grasp the functional features. On the other hand side, the communication between functional features organized via data or control structures in the source is revealed so that the communication with the other parts of the legacy system becomes explicit.

To gain the desired information, dynamic and static analyses complement each other. Further, metrics that measure the disparity, concentration, and dedication of features to program parts are computed [15]. Our domain oriented approach serves the goals of product line engineering: the analysis is not limited to classic programming paradigms but focuses on the user's needs. Indeed, even the users of the system may easily assist the product line engineer in analyzing the code assets by giving hints on how-to-use the system (this might range from typical usage to extremely rare cases).

Incremental shift towards product lines If the introduction of product line technology is too disruptive because the legacy system at hand is too large and too complex and the product is business critical to the company, then cautious and incremental shift towards product line methods is advisable.

We believe that our techniques can support the integration of development personal and avoid wrapping and decay of the competence encoded in the software. The proposed proceeding for that case is

I As long as the structure of the legacy software prevents the identification of sensible variation and commonality points, restructure according to findings of feature analysis.

II Identify variabilities and commonalities. Based upon that, identify interfaces.

3 Outlook

Currently, the Bauhaus system [1] at the Universtät Stuttgart provides means for incremental architecture recovery and code validation. To satisfy the specific needs of product line engineering, we are extending Bauhaus by the mentioned techniques for spotting software features. In conjunction with information about intra-software communication gained by connector recovery, we hope to find methods that empower software engineers to quickly find interfaces and aid their quest for reusable legacy assets. Cheap and quick reuse can help in convincing companies to switch to product line architectures initially. Further, if the legacy asset is on no account reusable, our approach will help to save much effort.

References

[1] The New Bauhaus Stuttgart. Available at `http://www.bauhaus-stuttgart.de`, 2001.

[2] Joachim Bayer, Oliver Flege, Peter Knauber, Roland Laqua, Dirk Muthig, Klaus Schmid, Tanya Widen, and Jean-Marc DeBaud. PuLSE: A Methodology to Develop Software Product Lines. In *Proceedings of the Fifth ACM SIGSOFT Symposium on Software Reusability (SSR'99)*, pages 122–131, Los Angeles, CA, USA, May 1999. ACM.

[3] Joachim Bayer, Jean-François Girard, Martin Würthner, Jean-Marc DeBaud, and Martin Apel. Transitioning Legacy Assets to a Product Line Architecture. In *Proceedings of the Seventh European Software Engineering Conference (ESEC'99)*, Lecture Notes in Computer Science 1687, pages 446–463, Toulouse, France, September 1999.

[4] John Bergey, Liam O'Brien, and Dennis Smith. Mining Existing Software Assets for Software Product Lines. Technical Report CMU/SEI-2000-TN-008, Software Engineering Institute (SEI), Carnegie Mellon University, May 2000.

[5] Jan Bosch. *Design & Use of Software Architectures*. Addison-Wesley and ACM Press, 2000.

[6] Kunrong Chen and Václav Rajlich. Case Study of Feature Location Using Dependence Graph. In *Proceedings of the International Workshop on Program Comprehension*, pages 241–249, Limerick, Ireland, June 2000. IEEE Computer Society Press.

[7] Thomas Eisenbarth, Rainer Koschke, and Daniel Simon. Herleitung der Feature-Komponenten-Korrespondenz mittels Begriffsanalyse. In *Proceedings of the 1. Deutscher Software-Produktlinien Workshop*, pages 63–68, Kaiserslautern, Germany, November 2000.

[8] Thomas Eisenbarth, Rainer Koschke, and Daniel Simon. Aiding Program Comprehension by Static and Dynamic Feature Analysis. In *Proceedings of the International Conference on Software Maintenance*, page To appear., Florence, Italy, November 2001. IEEE Computer Society Press.

[9] Thomas Eisenbarth, Rainer Koschke, and Daniel Simon. Derivation of Feature-Component Maps by Means of Concept Analysis. In Pedro Susa and Jürgen Ebert, editors, *Proceedings of the Fifth European Conference on Software Maintenance and Reengineering*, pages 176–179, Lisbon, Portugal, March 2001.

[10] Thomas Eisenbarth, Rainer Koschke, and Daniel Simon. Feature-Driven Program Understanding Using Concept Analysis of Execution Traces. In *Proceedings of the International Workshop on Program Comprehension*, pages 300–309, Toronto, Canada, May 2001. IEEE Computer Society Press.

[11] Linda M. Northrop. A Framework for Software Product Line Practice. Available at `http://www.sei.cmu.edu/plp/framework.html`, 2001.

[12] Nelson Weiderman, John Bergey, Dennis Smith, and Scott Tilley. Can Legacy Systems Beget Product Lines? *Lecture Notes in Computer Science*, 1429, 1998.

[13] Norman Wilde, Michelle Buckellew, Henry Page, and Václav Rajlich. A Case Study of Feature Location in Unstructured Legacy Fortran Code. In Pedro Susa and Jürgen Ebert, editors, *Proceedings of the Fifth European Conference on Software Maintenance and Reengineering*, pages 68–75, Lisbon, Portugal, March 2001.

[14] Norman Wilde and Michael Scully. Software Reconnaissance: Mapping Program Features to Code. *Journal of Software Maintenance: Research and Practice*, 7:49–62, January 1995.

[15] W. Eric Wong, Swapna S. Gokhale, and Joseph R. Hogan. Quantifying the Closeness between Program Components and Features. *The Journal of Systems and Software*, 54(2):87–98, October 2000.

Adopting Software Product Lines: Approaches, Artefacts and Organization

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

Abstract.
Software product lines have received considerable adoption in the software industry and prove to be a very successful approach to intra-organizational software reuse. Contemporary literature on the subject, however, often presents only a single approach towards adopting a software product line. In this paper, we present an overview of different adoption approaches, different maturity levels for product line artefacts and different organizational models.

1. Introduction

Software product lines have achieved substantial adoption by the software industry. A wide variety of companies has substantially decreased the cost of software development and maintenance and time to market and increased the quality of their software products. The product line approach is basically the first intra-organizational software reuse approach that has proven successful.

Contemporary literature on product lines often presents one particular approach to adopting a product line, suggesting a particular process model, a particular organizational model and a specific approach to adopt the product line approach. For instance, most existing literature assumes the organization to have adopted a domain engineering unit model, where this unit develops reusable artefacts while the product engineering units develop concrete products based on these reusable assets.

However, in our experiences with software companies that have adopted a product line approach, we have learned that the actually available alternatives are generally much more than the particular approach presented in traditional literature. The adoption of a product line, the product line processes and the organization of software development have more freedom than one may expect.

In the remainder of this paper, we present the various approaches that are available when working with software product lines. In the next section, we discuss four types of product line adoption. In section 3, we discuss the maturity levels for each main product line artefacts that we have identified. The organizational models one may adopt are discussed in section 4. The paper is concluded in section 5.

2. Product Line Adoption

Software product lines do not appear accidentally, but require a conscious and explicit effort from the organization interested in employing the product line approach. Basically, one can identify two relevant dimensions with respect to the initiation process. First, the organization may take an evolutionary or a revolutionary approach to the adoption process. Secondly, the product line approach can be applied to an existing line of products or to a new system or product family that the organization intends to use to expand its market with. Each case has an associated risk level and benefits. For instance, in general, the revolutionary approach involves more risk, but higher returns compared to the evolutionary approach. In table 1, the characteristics of each case are briefly described.

	Evolutionary	Revolutionary
Existing set of products	Develop vision for product line architecture based on the architectures of family members Develop one product line component at a time (possibly for a subset of product line members) by evolving existing components	Product line architecture and components are developed based on super-set of product line member requirements and predicted future requirements.

New product line	Product line architecture and components evolve with the requirements posed by new product line members	Product line architecture and components developed to match requirements of all expected product line members

Table 1. Two dimensions of product line initiation

The advantage of the evolutionary approach is that risk is minimized in two ways. First, since the up-front investment is decomposed into small steps for each component, the pay-off of sharing components gives a quick return of investment and the investment itself is relatively small. Second, the products in the family continue their normal evolution, albeit at a somewhat slower pace, whereas the revolutionary approach generally stops all but cosmetic evolution of the products in the family until the product line architecture and components are in place. The disadvantage is that the total amount of investment until the product line architecture and components are completely in place is larger than when using the revolutionary approach. This because much work on components done during the conversion is only performed to support temporary requirements.

The advantages of replacing an existing set of products with a product line, i.e. a revolutionary approach, are especially the shorter conversion time and the generally smaller total investment required for developing the architecture and component set for the family, when compared to the evolutionary approach. The primary disadvantages are the increased risk level, due to the large initial investment that may prove useless if important requirements change, and the delayed time-to-market of the first products developed based on the product line architecture.

Finally, an important factor in the decision to either evolve or replace a set of systems is the presence of mechanical and hardware parts. If systems, in addition to software contain considerable pieces of mechanics and hardware, the product line approach needs to be synchronized for all three aspects. Especially if considerable differences exist in the mechanical and hardware parts of the products, it is generally harder to employ an evolutionary approach and replacing the existing systems may simply be the only alternative.

3. Product Line Artefacts

One can identify three types of artefacts that make up a software product line, i.e. the product line architecture, shared components and the products derived from the shared artefacts. For each of these artefacts, one can identify three maturity levels, depending on the level of integration achieved in the product line. Below, we discuss each artefact in more detail.

The software architecture of the product line is the artefact that defines the overall decomposition of the products into the main components. In doing so, the architectures captures the commonalities between products and facilitates the variability. One can identify three levels of maturity:

- **Under-specified architecture**: A first step in the evolutionary adoption of a software product line, especially when converging an existing set of products, is to first define the common aspects between the products and to avoid the specification of the differences. This definition of the architecture gives existing and new products a basic frame of reference, but still allows for substantial freedom in product specific architectural deviation.
- **Specified architecture**: The next maturity level is to specify both the commonalities and the differences between the products in the software architecture. Now, the architecture captures most of the domain covered by the set of products, although individual products may exploit variation points for product specific functionality. The products still derive a product specific architecture from the product line architecture and may consequently make changes. However, the amount of freedom is substantially less than in the under-specified architecture.
- **Enforced architecture**: The highest maturity level is the enforced architecture. The architecture captures all commonality and variability to the extent where no product needs, nor is allowed, to change the architecture in its implementation. All products use the architecture as-is and exploit the variation points to implement product specific requirements.

The second type of artefact is the product line component, shared by some or all products in the product line. Whereas the product line architecture defines a way of thinking about the products and rationale for the structure chosen, the components contribute to the product line by providing reusable

implementations that fit into the specified architecture. Again, one can identify three levels of maturity for product line components:

- **Specified component**: The first step in converging a set of existing products towards a product line is to specify the interface of the components defined by the architecture. A component specification typically consists of a provided, a required and a configuration interface. Based on the component specifications, the individual products can evolve their architecture and product specific component implementations towards the product line thereby simplifying further integration in the future.
- **Multiple component implementations**: The second level of maturity is where, for an architectural component, multiple component implementations exist, but each implementation is shared by more than one product. Typically, closely related products have converged to the extent that component sharing has become feasible and, where necessary, variation points have been implemented in the shared components.
- **Configurable component implementation**: The third level is where only one component implementation is used. This implementation is typically highly configurable since all required variability has been captured in the component implementation. Often, additional support is provided for configuring or deriving a product specific instantiation of the component, e.g. through graphical tools or generators.

The third artefact type in a software product line is the products derived from the common product line artefacts. Again, three levels of maturity can be distinguished:

- **Architecture conformance**: The first step in converging a product towards a product line is to conform to the architecture specified by the product line. A product can only be considered a member of the product line if it at least conforms to the under-specified architecture.
- **Platform-based product**: The second level is the minimalist approach where only those components are shared between products that capture functionality common to all products. Because the functionality is so common, typically little variability needs to be implemented.
- **Configurable product base**: The third level of maturity is the maximalist approach, where all or almost all functionality implemented by any of the product line members is captured by the shared product line artefacts. Products are derived by configuring and (de-)selecting elements. Often, automated support is provided to derive individual products.

4. Organizational Models

In this section, we discuss a number of organizational models that can be applied when adopting a software product line based approach to software development. Below, we briefly introduce the models. For a more extensive discussion, we refer to [1].

- **Development department**: When all software development is concentrated in a single development department, no organizational specialization exists with either the product line assets or the products in the product line. Instead, the staff at the department is considered to be resource that can be assigned to a variety of projects, including domain engineering projects to develop and evolve the reusable assets that make up the product line.
- **Business units**: The second type of organizational model employs a specialization around the type of products. Each business unit is responsible for one or a subset of the products in the product line. The business units share the product line assets and evolution of these assets is performed by the unit that needs to incorporate new functionality in one of the assets to fulfil the requirements of the product or products it is responsible for. On occasion, business units may initiate domain engineering projects to either develop new shared assets or to perform major reorganizations of existing assets.
- **Domain engineering unit**: This model is the suggested organization for software product lines as presented in the traditional literature, e.g. Dikel et al. [2] and Macala et al. [4]. In this model, the domain engineering unit is responsible for the design, development and evolution of the reusable assets, i.e. the product line architecture and shared components that make up the reusable part of the product line. In addition, business units, often referred to as product engineering units, are responsible for developing and evolving the products based on the product line assets.
- **Hierarchical domain engineering units**: In cases where an hierarchical product line has been necessary, also a hierarchy of domain units may be required. In this case, often terms such as ‘platforms’ are used to refer to the top-level product line. The domain engineering units that work with specialized product lines use the top-level product line assets as a basis to found their own product line upon.

Some factors that influence the organizational model, but that we have not mentioned include the physical location of the staff involved in the software product line, the project management maturity, the organizational culture and the type of products. In addition to the size of the product line in terms of the number of products and product variants and the number of staff members, these factors are important for choosing the optimal model.

5. Conclusions

Software product lines have received wide adoption in many software companies and proven to be very successful in achieving intra-organizational reuse. Traditional approaches to software product line based development typically take one particular approach. In this paper, we discussed the different alternatives that are available when adopting a product line, the maturity levels of product line artefacts and different organizational models that are available.

We have discussed four different approaches to adopting a software product line, organized in two dimensions, i.e. evolutionary versus revolutionary adoption and replacing an existing set of products versus developing a new set of products.

In addition, we have discussed maturity levels for the main product line artefacts. The product line architecture can be under-specified, fully specified or enforced. The components can just consist of a component interface specification, multiple component implementations and one configurable component implementation. Finally, a product can conform to the product line architecture, be a platform-based product or be derived from a configurable product base.

Finally, we discussed four alternative organizational models, i.e. the development department model, the business unit model, the domain engineering unit model and the hierarchical domain engineering unit model.

References

[1] J. 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.

[2] D. Dikel, D. Kane, S. Ornburn, W. Loftus, J. Wilson, 'Applying Software Product-Line Architecture,' IEEE Computer, pp. 49-55, August 1997.

[3] J. van Gurp, J. Bosch, M. Svahnberg, 'On the Notion of Variability in Software Product Lines,' Accepted for The Working IEEE/IFIP Conference on Software Architecture (WICSA 2001), April 2001.

[4] R.R. Macala, L.D. Stuckey, D.C. Gross, 'Managing Domain-Specific Product-Line Development,' IEEE Software, pp. 57-67, 1996.

[5] D.M. Weiss, C.T.R. Lai, Software Product-Line Engineering - A Family-Based Software Development Process, Addison-Wesley, ISBN 0-201-69438-7, 1999.

Feature Scoping for Product Lines

Matthias Riebisch, Detlef Streitferdt, Ilka Philippow
Ilmenau Technical University, Ilmenau, Germany
{matthias.riebisch|detlef.streitferdt|ilka.philippow}@tu-ilmenau.de

Abstract

Product Line (PL) Engineering focuses on the development of complete system families as opposed to single systems. Systems are built of a reusable platform common to the whole family, and of specific parts extending it. The benefits of short time-to-market and lower development costs for each system within the system family are achieved by reusing the platform for each new system to be developed. Therefore the scoping of features for the reusable platform and the specific parts is crucial for PL success.
This paper proposes scoping with 4 priority levels and a decision-table based interpretation of the results. The interpretation is shown both for start and for evolution of product lines. The paper is based on experiences on large-scale reuse in industrial software projects.

1 Introduction

Reusability of software has been and is an important goal of software engineering researchers and practitioners. Experiences with different approaches have shown that both technical and organizational, economical, and psychological factors are crucial for success of s/w reuse. Product Line Engineering (PLE) focuses on the development of complete system families as opposed to single systems. By defining a reusable platform for all members of the family, a professional and planned way of software reuse is possible, based on domain analysis and other technologies. Additionally, PL offer a way for economic planning, e.g. for the return-on-invest and other key figures.

However, there are still various steps which are critical for economic success. Scoping is one of the most important steps, and one of the less formalized ones among them. Generally, scoping is the process of deciding about the effort of a software development task. This decision is influenced by the market situation, the customer needs and expectations, the strategic business goals, and the estimated effort for this task. Schmid's survey on scoping in PLE [Schmid 2000] classifies three ways of scoping:

- identifying products that should be part of a product line: product portfolio definition
- bounding the domains which are relevant: domain-centric scoping
- identifying the specific assets that should be part of the reuse infrastructure: asset-centric scoping)

In this paper, scoping focuses on the decision, which features should be implemented in the reusable platform on in a variable part of the PL. A feature is a property or a quality of a product. The set of features implemented in a PL are modeled in a so-called feature model (see [Czarnecki et al. 2000]). This way of decision is very similar to that supported by [DeBaud 2000], but extended by the decision about features which are supported by the reusable platform, even if they are outside the platform.

For the described step of scoping there are descriptions in recent publications, e.g. [DeBaud 2000] and PuLSE-Eco in [Bayer et al. 1999]. In practice however only sparse support for interpretation of scoping results as the basis for further development steps can be found.

This paper is organized as follows: After a short investigation of scoping in conventional software development there is a proposal for classifying scoping results in PLE by priority levels. An interpretation scheme of this priorities for deciding the next implementation tasks is described. In a separate section, additional propositions for further discussion during the workshop are listed.

2 Scoping In Conventional Software Development

Scoping in conventional software development – without PLs – is performed as part of the requirements engineering activities. Its predecessor activities are requirements elicitation and requirements modeling, which are followed by design and implementation phases. As mentioned above, scoping is influenced by the current market situation, the customer expectations, the strategic business goals, and the estimated effort for implementing a particular requirement resp. feature. Scoping results in the form of priorities are assigned to the list of features. Often these priorities occur in three levels:

1 - to be implemented (as part of the next development cycle),
2 - to be implemented in a later development cycle,
3 - not to be implemented (at the moment).

The task of assigning priorities consists of two main steps: the definition of business goals and their assignment to features, and the calculation of priorities. There are successful techniques for performing these two tasks: the Goal-Question-Metric technique GQM [Solingen et al. 1999], and the Quality Function Deployment QFD [Sullivan 1986].

3 Scoping In Product Line Engineering

In PLE, scoping is one of the most critical success factors.

- It influences the development effort during later changes of the reusable platform. Scoping represents the planning of reuse and therefore influences the return-on-invest.
- If performed according to the customer needs, it enables a short time-to-market and low development costs.
- By planning reusability, it leads to a higher process maturity for the reusable platform. Thus, software quality and evolvability are improved. Evolvability enables a longer usage time for the parts of the PL and therefore influences the return-on-invest, too.

In PLE, priorities are used for the assignment of features to development cycles as well as for their assignment to the reusable platform itself or to the variable parts respectively. Thus four priority levels are proposed:

1 - to be implemented for all systems in the PL, part of the reusable platform
2 - to be implemented for some systems in the PL, variable parts,
3 - to be implemented for some systems in the PL in a later development cycle, at the moment, however, no implementation
4 - not to be implemented (at the moment).

The consequences of the priority assignment depend on the stage of the PL development. Either it is just started or it is to be evolved. The next section deals with these issues.

The new criteria for deriving priorities are a refinement of those in the preceding section. Differences are described as follows:

- The market situation and the customer expectations are analyzed to rate the current situation and to predict future trends, since the future has much more impact on PLE than on single system development.
- Strategic business goals correlate with investments in the PL's reusable platform. They have to be consistent with the marketing strategy and the organization of the company.
- The effort for implementing a particular requirement resp. feature is usually estimated by experts. Due to the higher complexity of a PL such estimations are more complex, too. However, in the case of an existing PL with existing feature models, cost estimations can be partly replaced by assessing traceability links. Such links offer possibilities for automating several development steps (see [Philippow et al. 2001]).

For the calculation of priorities, methodologies similar to QFD are used. In TrueScope [DeBaud 2000] and in PuLSE-Eco [Bayer et al. 1999] these calculation schemes are called product map.

Interpreting Scoping Results For Successful Product Line Development

The results of scoping are priorities assigned to features in the feature model. To decide the next steps of development these priorities have to be interpreted. In literature, there is no systematic way for decision-making. The interpretation of the priorities depend on the development stage of the PL. In order to achieve a more systematic way of interpretation, the use of decision tables [Kohavi 1995] for architecture, design and coding decisions is proposed. Decision tables are in use in several branches of engineering, e.g. automation engineering. A decision table consists of scheme of input values as columns and resulting actions in the rows with conditions in the body of the table. In one step, all actions are performed, where the conditions are met by the input values. If in this step there is no matching action, an optional default action is performed. Using the decision table, a decision about the implementation of each feature of the PL is made. Here, we discuss three cases: the start of a PL development from scratch, the start of a PL development from existing assets, and the evolution of an existing PL.

Start of a Product Line Development

The starting of a PL development from scratch is the simplest case. The decision table is very straightforward (Tab 1): Prio-1 features are implemented within the reusable platform. Prio-2 features are

implemented by variable parts of the PL. The interfaces of these parts have to be supported by the reusable platform.

Under some conditions prio-3 features influence architectural decisions for the reusable platform: The architecture of the reusable platform is prepared for later implementation of these features, if the effort for later refactoring is significantly higher than the preparation of the reusable platform at this moment, and if the complexity of the platform's architecture is not significantly increased by this decision. This way, decisions are influenced by the intentions of the Extreme Programming approach [Beck 1999]. Although these considerations have not been discussed in current papers, they might lead to economic advantages.

All other prio-3 and prio-4 features are neither considered for decision-making nor for implementation.

Tab 1: Decision Table for the Start of a Product Line Development

...is of prio 1	…is of prio 2	…is of prio 3	…leads to significant increase of complexity	…leads to higher refactoring effort later than now	Feature in question … / Resulting action
yes	no	no	don't care	don't care	To be implemented as part of the reusable platform
no	yes	no	don't care	don't care	To be implemented as variable part
no	no	yes	no	yes	Platform architecture to be prepared for future support
					Default: no change

Start of a Product Line Development with Existing Products

In this case existing products need to be integrated in the PL. They will have to be refactored in order to divide them into parts according to the features. The refactoring effort partly leads to economically-driven decisions for re-development of large portions of the products. This effort has to be considered when estimating the total implementation effort. All other decisions are identically to the section above.

Evolution of an Existing Product Line

Here, the resulting decision table is more complex (Tab 2). Features with priorities of 3 or 4 are not implemented. Prio-1 and prio-2 features are implemented in the reusable platform or by variable parts of the PL, respectively. If the priority of a former prio-1 feature decreases resp. prio-2 feature we have to distinct, if its priority is increased or decreased, compared to the previous development cycle. In these cases a refactoring has to be done.[1]

Tab 2: Decision Table for Evolution of an Existing Product Line

…was previously part of the core	…was previously in a variable part	…was previously not implemented	...is of prio 1	...is of prio 2	Feature in question … / Resulting action
no	no	yes	yes	no	To be implemented as part of the reusable platform
no	no	yes	no	yes	To be implemented as variable part
yes	no	no	no	don't care	Reusable platform to be refactored in order to remove this feature; variable parts to be adjusted accordingly
no	don't care	don't care	yes	no	To be included in reusable platform; variable parts to be adjusted accordingly
					Default: no change

4 Propositions for Discussion

The following issues came up during discussions in our research team. Our intention is to put the given statements as "requests for comment" during the workshop.

- Scoping is one of the most critical steps for economic success in requirements engineering of PLs. While there are formalized ways for the elicitation of business goals (GQM) and for the computa-

[1] As the attentive reader will notice, the 1st and 4th row of the decision table could be joined in a normalization step, because the corresponding actions differ only slightly.

tion of scoping priorities (QFD), there is no systematic way for interpreting the results and deciding about the next steps. Decision tables offer such a way.

- Similar to the Extreme Programming (XP) philosophy all development task should be simplified as much as possible. Especially no effort for future requirements is to be invested to keep the reusable core simple and to assure a high evolvability.
- The thresholds for the priority levels 1 to 4 are defined on cooperation of management, marketing and development. They can be verified in every development cycle.
- In some cases there is a need for an additional priority level between 2 and 3 for requirements, which are to be implemented in a later product, but resources for them have to be implemented or to be prepared in the reusable platform in this development cycle.
- Effort estimation for scoping purposes is usually carried out by developers based on their expert knowledge. If the PL development is documented using feature models and traceability links, then effort for changes can be derived by tools in a more systematic way. Traceability links could be collected in a way similar to e.g. TrueScope's Increment Control List.
- Marketing and customer relations have adopt the PL philosophy. Their support for the evolution of a PL is more critical for long-term success than technical development.
- Similar to the experiences with software reusability, software quality requirements for the reusable platform are stronger than for other parts. This fact has to be considered in planning development cycles.

References

[Bayer et al. 1999] J. Bayer, O. Flege, P. Knauber, R. Laqua, D. Muthig, K. Schmid, T. Widen, J.-M. DeBaud: PuLSE – A Methodology to Develop Software Product Lines. Symposium on Software Reusability, Los Angeles, CA, USA (SSR'99), 1999, pp. 122-131.

[Beck 1999] Beck, Kent: Extreme Programming**{** XE "Extreme Programming" **}** Explained: Embrace Change. Addison Wesley Longman, Reading/Massachusetts, 1999.

[Czarnecki et al. 2000] Czarnecki, K., Eisenecker, U.W.: Generative Programming. Addison Wesley, Reading, MA, 2000.

[CQM 1993] -: Kano's Method Special Issue. Center for Quality of Management Journal, ISSN 1072-5296, Vol. 2, No. 4, Fall 1993, http://cqmextra.cqm.org/cqmjournal.nsf/issues/vol2no4

[Clements et al. 1998] Clements, Paul, Northrop, Linda M., et al.: A Framework for Software Product Line Practice – Version 1.0. SEI, CMU, Sept. 1998. http://www.sei.cmu.edu/plp

[DeBaud 2000] DeBaud, Jean-Marc: TrueScope – A Full LifeCycle Approach to Develop Software Product Lines. In: [SPLC 2000], Tutorial 6.

[Kohavi 1995] R. Kohavi: The Power of Decision Tables. In the European Conference on Machine Learning, 1995. http://robotics.stanford.edu/users/ronnyk/ronnyk-bib.html

[Philippow et al. 2001] Ilka Philippow, Matthias Riebisch: Systematic Definition of Reusable Architectures. In Proceedings of the 8th IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2001), April 2001, pp. 128-135.

[Schmid 2000] Schmid, Klaus: Scoping Software Product Lines. In: [SPLC 2000], pp513 – 532.

[Solingen et al. 1999] R.Solingen, E.Berghout: The Goal/Question/Metric Method. McGraw-Hill Publishing Company, 1999.

[SPLC 2000] Donohoe, Patrick (Ed.): Software Product Lines – Experiences and Research Directions. Proc. 1st Software Product Lines Conf. (SPLC1), Aug. 28-31, 2000, Denver, Colorado, Kluwer Acad. Publ. 2000. http://www.sei.cmu.edu/plp/conf/SPLC.html

[Sullivan 1986] L. P. Sullivan: Quality function deployment. Quality Progress, 19(6), 1986. pp. 39-50.

Combining Product Line Engineering with Options Thinking

Birgit Geppert, Frank Roessler
Avaya Labs- Software Technology Research
233 Mt. Airy Road, Basking Ridge, NJ 07920, USA
{bgeppert, roessler}@research.avayalabs.com

Abstract

Developing a product family in an uncertain environment where future family members cannot be reliably predicted generally implies high risk for a product line approach. As managers usually tend to favour short term and low risk projects, such a product family is not likely to be engineered as a product line. However, we can engineer different evolutionary paths of a product that are uncertain but have high potential of revenue growth as a product line also. One would start with developing common aspects of possible evolutionary paths and move on with variabilities while uncertainty is resolved. Because of time-to-market advantages the inherent flexibility of the resulting architecture can have great value for an organization and therefore justify a product line approach. In this context variabilities are considered as options that give you the right but not the obligation to evolve the product in one of several possible directions. This paper introduces the basic idea of *Strategic Product Line Engineering* and gives an outlook on how option-pricing theory might be applied to estimate the value of strategic product lines.

Keywords. Software Product Lines, Strategic Software Design, Real Options

1. Strategic Product Line Engineering

The concept of *software families* has been well-known for decades [8]. However, developing a family of programs is not necessarily equivalent to developing it *as* a family, i.e., systematically exploiting the family members' commonalities and controlling their variabilities. Such exploitation results in a *software product line*, which usually includes an infrastructure that enables rapid development of family members.

A common problem of product line engineering as documented in [2], [10], is that the infrastructure can require a substantial initial investment. This investment is returned through the accumulated savings we obtain by deriving the family members from the common infrastructure. Figure 1 illustrates the underlying economic model: after a certain number of family members have been produced, the investment is repaid. From this point

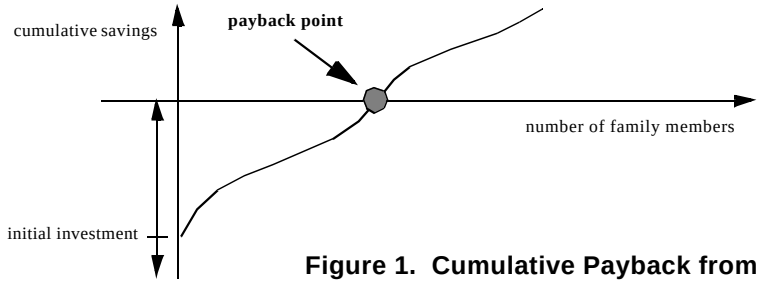

Figure 1. Cumulative Payback from Product Line Engineering

on, we expect net return on the investment in the product line. Note that the shape of the curve depends on the specific project. In particular, the amount of initial investment varies and can be influenced by different adoption strategies (in specific cases initial investment may even be zero). In this paper, we are considering projects where some initial investment is deemed to be necessary or advantageous and seek for possibilities to justify it.

The economic model of Figure 1 fits very well when we consider product families with several family members produced at the same time, possibly in regular intervals, as shown in Figure 2. In such a situation, we can reliably predict family members and can expect the payback point in the near future. Therefore an initial investment can be justified without taking too much risk. An example is a cellular phone vendor, who produces about 10 new phones every year. Old phone models are not maintained but replaced with more advanced models.

However, the situation is different if we consider an evolutionary product that is maintained and evolved over time resulting in several releases instead of new models. We can still consider the set of releases as a family, but can we justify developing them as a product line? An example for such an evolutionary product would be a PBX.

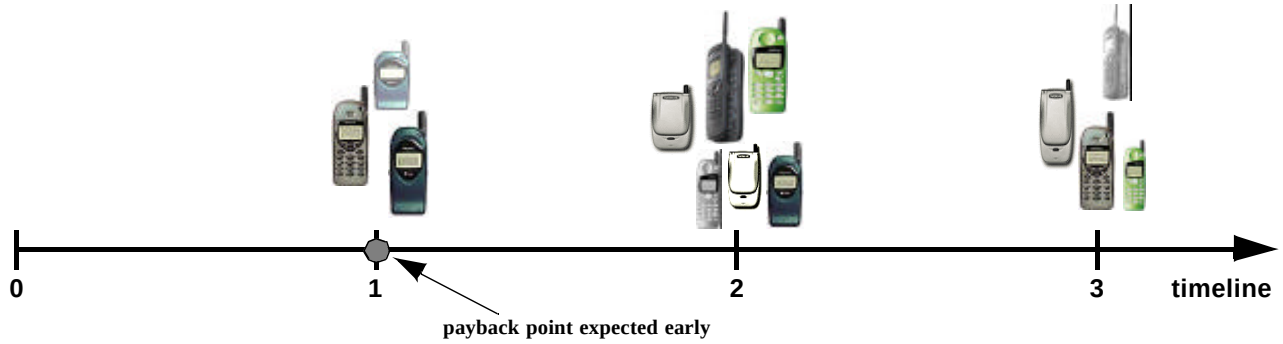

Figure 2. Product Family in Space: Cell Phones

However, because product line engineering is currently not mature enough to be applied to such complex systems at a whole, we apply it only to selected subsystems and therefore also only get part of the potential savings. Furthermore, call processing software is too complex to allow for many variants at the same time[1]. Instead, the existing product is gradually evolved as illustrated in Figure 3. Family members are spread out over a long time, and therefore a potential payback point is located further out in the future. This increases the risk when deciding on a product line approach, because we cannot well predict what family members would be needed. Since managers tend to favour short term and low risk projects, a product family in time is generally a less likely candidate for a product line.

But does this mean that product line technology should not be applied to evolutionary products at all? Definitely not! Instead of engineering the family in time as a product line we should think of possible evolutionary paths as a product line. Figure 4 illustrates this for the PBX example: assume we are at time 0. At that time, we can already reliably predict that there will be four possible variations for release 2 of our product. Which variation will eventually be most successful depends on future events that we currently cannot control or reliably predict (such as customer reaction to release 1 or announcements of competitive products). Nevertheless, it might be beneficial to provide for sufficient flexibility in the software architecture of release 1 to quickly move to one of those different variants when release 2 is due (in the example, we have chosen to only support three of the predicted variants). These variations can be provided through common product line technology, but we prefer calling this approach *Strategic Product Line Engineering*. We consider the predicted variants for future releases as a (alternate) family and create an adequate PL infrastructure for it. It is important to understand that variabilities in this context must be options that give you the right - but not the obligation - to evolve the product in one of several possible directions[2].

One might think that this is actually a waste of resources because not all considered family members are going to be developed and not all incorporated options are going to be exercised. However, we argue that having a portfolio of options in the architecture - even without guarantees that they all will eventually be exercised - can create *immediate* value and therefore justifies developing the alternate family as a product line.

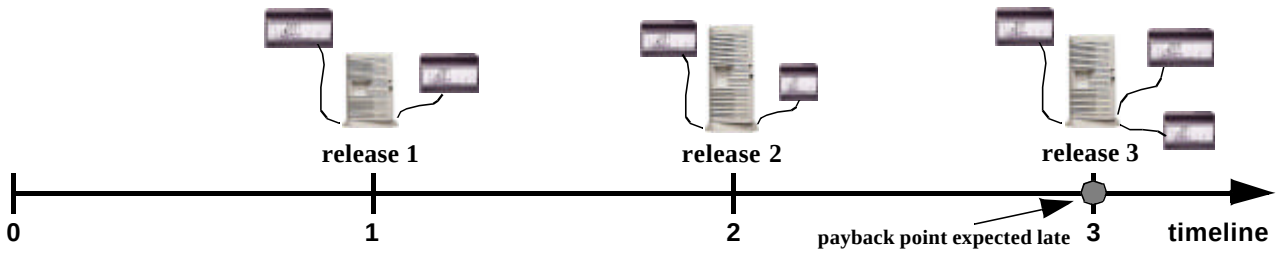

Figure 3. Product Family in Time: PBX

[1] Let us assume that there is no customization such as different line sizes, different billing systems, or country-specific needs.

[2] In particular, this means that costs for "encapsulating" the variability in the design should be low compared to the costs for implementing a variant.

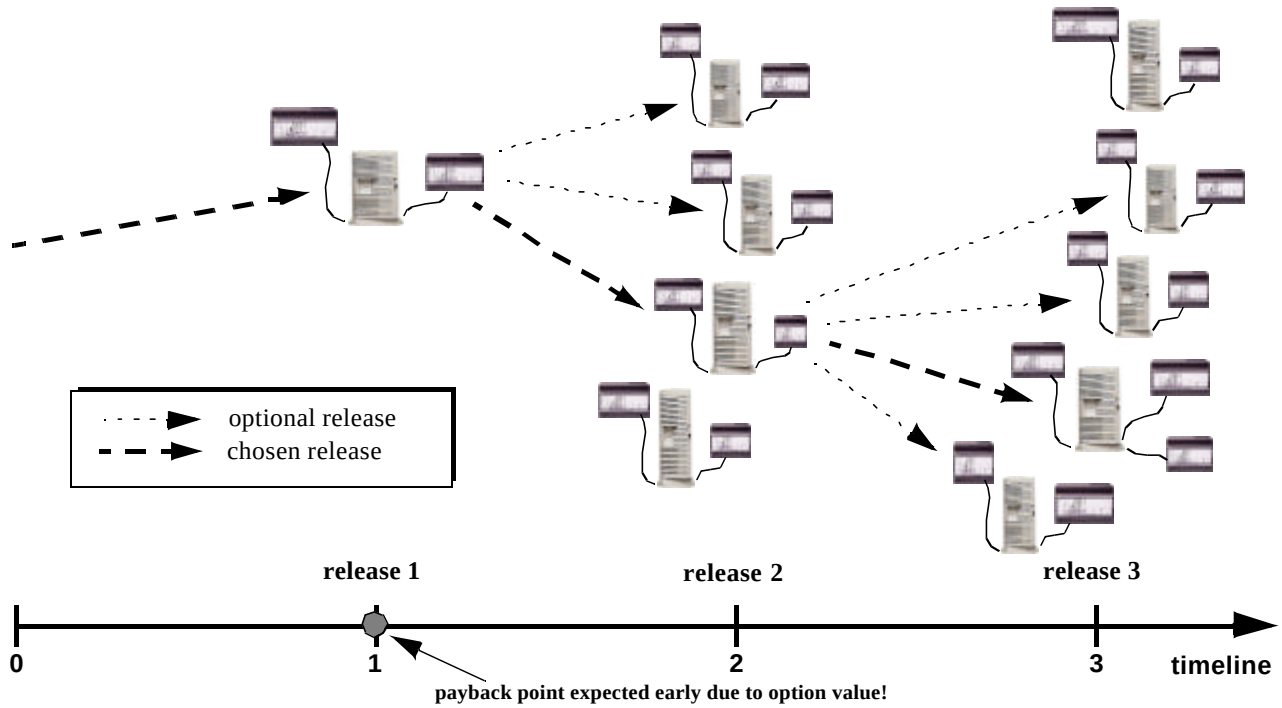

Figure 4. Strategic Product Family: Option Space for PBX

2. Value of Strategic Product Lines

The economic model of Figure 1 for estimating the value of a product line is based on a common method for valuing investments called *discounted-cash-flow* (DCF) *analysis*. Like any other valuation technique, it is a function of the three fundamental factors: cash, timing, and risk [6]. DCF analysis identifies all cash flows related to the investment, such as expected revenues from the different family members, costs for conducting the commonality analysis, or costs for building code generators. This yields a series of cash flows stretching into the future. Those cash flows are adjusted for time and risk and then summed up to yield the value of the investment. Only if the value is greater than zero, do we recommend proceeding with the project.

The main problem with DCF calculations is that they are only valid when valuing an ongoing business or an immediate investment. DCF analysis can estimate how much expected future cash flows are worth once an investment is made. Hence, as an analysis tool for managers, DCF can only support go or no-go decisions. It does not account for the ability of managers to react to new circumstances - for instance, to spend a little up front, see how things develop, and then either cancel or go full speed. However, strategic product line engineering is based on exactly this kind of flexibility in the product and project structure. In order to estimate the value of strategic product lines, we therefore need a valuation approach that considers flexibility in decision-making. In the following we briefly illustrate such an approach.

In [3], Black and Scholes proposed an analytic model for determining the fair market value of a call option, which resulted in the so-called *option-pricing theory*. Today, this is one of the most widely accepted financial models[1].

The idea of options was certainly not new. When used in the financial world, options are generally defined as a *contract between two parties in which one party has the right but not the obligation to do something, usually to buy or sell some underlying asset at a specified price on or before some future date*. In particular, call options are contracts that give the option holder the right to buy something. A call option on a share of stock gives you the right - but not the obligation - to buy that share for, say, $100 by the end of this year. If the share is currently

[1] Myron Scholes together with Robert Merton were rewarded with the Nobel Prize in economics in 1997.

worth more than $100 the option certainly is valuable. But even if it currently sells for less than $100, it might still be valuable, because stock prices can rise before the option expires. Thus, having rights without obligations has financial value, a value that option-pricing theory can calculate very accurately.

There are only a few parameters that influence the value of an option. The most important ones are the volatility of stock price and the time to expiration. The expected rate of return of the underlying asset is not one of the variables in any model for option valuation. This means that a prediction of the future price of the underlying asset is not necessary to price an option. Thus, while any two investors may strongly disagree on the expected rate of return on a stock, they will always agree on a fair price for the option, as long as they agree on volatility and the risk-free rate. Since 1973, the original Black-Scholes Option-Pricing Model has been expanded in many ways and several of its assumptions have been relaxed, resulting in amazingly accurate valuation models for stock options.

To apply an option-pricing model to (real) options that are embedded in a strategic product line we must find a mapping from project characteristics onto the variables of the option model [7]. Let us assume an option allows us to replace quickly a protocol standard used in our PBX. It is only necessary to exchange a module that implements the new protocol. This investment opportunity is like a call option, because the software organization has the right - but not an obligation - to develop the new module. If we could construct a call option sufficiently similar to the investment opportunity, the option price would tell us something about the value of the investment [6]. For instance, we can easily map the project characteristics onto the five variables of the Black-Scholes Model: spending resources for developing the module is analogous to exercising the call option, i.e., the development costs correspond to the option's exercise price. The present value of the module, including module development costs and expected savings from product line engineering, correspond to the stock price. The release date corresponds to the expiration date of the call option. The uncertainty about the future revenues related to the module corresponds to the volatility of the stock price.

The option we construct this way is certainly not a perfect substitute for the real option (i.e., the module investment opportunity), because not all assumptions of the Black-Scholes model will be met by our project. However, even with the basic Black-Scholes Model we expect to get better insights into our project than with a simple DCF analysis. However, there remain difficult questions about a reasonable estimation of certain variables of the Black-Scholes Model such as volatility, and whether certain assumptions of option models can be met in the non-financial world. The literature in the field of real options promises answers to these questions ([1], [4], [5], [9]) that must be further investigated in the context of product line engineering.

3. Summary

In this paper, we introduced the basic idea of *Strategic Product Line Engineering*, a method that engineers possible evolutionary paths of a product as a product line. In this context, variabilities are considered options that give you the right but not the obligation to evolve the evolutionary product in one of several possible directions. We also illustrated that real option theory has the potential to support strategic product line engineering by helping managers to quantify the value of active management and strategic interaction embedded in various product line investment opportunities.

References

[1] C. Y. Baldwin and K. B. Clark, *Design Rules - The Power of Modularity*, MIT Press, 2000

[2] J. Bayer et al., *PuLSE: A methodology to develop software product lines*, Symposium on Software Reusability, 1999

[3] F. Black and M. Scholes, *The pricing of options and corporate liabilities*, Journal of Political economy, 1973

[4] T. Copeland and V. Antikarov, *Real Options: A Practitioner's Guide*, Texere, 2001

[5] J. Favaro, K. Favaro, and P. Favaro, *Value based software reuse investment*, Annals of Software Engineering 5, 1998

[6] T.A.Luehrman, *What's it worth? A general manager's guide to valuation*, Harvard Business Review, May/Jun 1997

[7] T.A.Luehrman, *Investment Opportunities as Real Options:Getting Started on the Numbers*, Harvard Business Review, Jul/Aug 1998

[8] D. L. Parnas, *On the Design and Development of Program Families*, IEEE Transactions on Software Engineering, No. 1, March 1976

[9] K.J. Sullivan et al., *Software design as an investment activity: a real options perspective*, in *Real Options and Business Strategy: Applications to Decision Making*, L. Trigeorgis (Editor), Risk Books, 1999

[10] D. Weiss and R. Lai, *Software Product-Line Engineering - A Family-Based Software Development Process*, Addison Wesley, 1999

Product Line Development as a Rational, Strategic Decision

Klaus Schmid and Isabel John

Fraunhofer Institute for
Experimental Software Engineering (IESE)
Sauerwiesen 6
D-67661 Kaiserslautern, Germany
++49 (0) 6301 707 {158, 250}
{Klaus.Schmid, Isabel.John}@iese.fhg.de

ABSTRACT[1]
Product line development requires a certain amount of up front investment in order to make assets reusable. This investment often keeps organizations from planning and realizing their products in a product line. But there are also major advantages of doing product line development which are often not taken into account when deciding for or against product lines. In this paper we present problems that need to be addressed in the strategic planning of transition towards product lines. These aspects include the involvement of uncertainty, the interdependence of technical solution and decision making, and the interdependence among the decision making and the market aspects. An approach which covers these aspects will determine an objective valuation of a product line.

Key Words:
Product Line Transition, Strategic Development, Real Options, Scoping, Business Case

1 INTRODUCTION

Transforming a classical, stove-pipe development organization into a product line organization is a major shift. Performing this shift requires major investments in reuse, but may also produce major benefits to the organization [Coh01]. Obviously, a decision of such importance should be driven from a strategy and should be the result of a thorough analysis of the pros and cons of introducing product line development for a family of products. In this paper, we will discuss some problems that need to be addressed when analyzing this decision in an objective, economically adequate manner.

When doing market driven development (as opposed to customer driven development), there are no contracts when starting product development. A product is planned for a certain market, development starts at one point in time and at a later point the product is released to the now different or even decaying market. Classically, strategy development is purely driven from the business objectives of an organization and results in a business development plan, which typically stretches over a multi-year time-horizon. In the particular situation of product line development, the business objectives usually refer to aspects like cost reduction, time-to-market reduction, or quality improvement and the corresponding question is whether a product line development approach should be implemented or not. In the rational case this decision is based on answering the question whether product line development will produce more benefits than its implementation will cost.

This question is fully analogous to the scoping problem [Sch00]. The scoping problem is to identify in a rational manner the specific assets that should be developed for reuse. Basically, scoping and business case analysis differ only in terms of size of the object under study. Scoping addresses planning the amount of reuse to be made, business case analysis addresses whether to reuse at all.

The arguments we will present in this paper apply equally to the formulation of business cases (i.e., on the overall product level) — whether they are formulated in the context of the big-bang introduction or in the context of incremental product line introduction, as well as on the level of scoping a reuse infrastructure (i.e., for individual features). Thus, throughout our paper we will not further make this distinction explicit.

1. The work presented in this paper was partially supported by Eureka ?! 2023 Programme, ITEA project ip00004, Café.

In this paper we will focus on some specific key aspects impacting the valuation of a strategy for product line transition. We will not provide a comprehensive discussion of the economic aspects of product line development as this has already been given in [Sch01]. Rather, for simplicity of our discussion, we will focus on the aspects of (development) costs, time, and revenues from the products and leave out aspects like risk or quality.

In this paper, we assume the reader is familiar with the basics of reuse economics and will restrict our discussion to three key positions pertaining to the strategic decision of product line introduction:

- Valuation of uncertainty is a key ingredient of strategic planning.
- Product line development profoundly alters the decision making landscape for strategic planning.
- Software development and market aspects interact and so should be planned in an integrated manner.

We will now discuss each of the three positions in turn as they build in this order on each other.

2 VALUATION OF UNCERTAINTY IS A KEY INGREDIENT OF STRATEGIC PLANNING

The valuation of the decision to introduce product line development is composed of the investment required for the transition and the benefits that can be reaped from this transition. Typically, these are discussed solely as aspects of software engineering, i.e., how much do code assets cost to develop with a product line vs. without. If the cost of money is taken into account, typically some rough assumptions about the number of systems that can be expected are made, e.g., two per year [Coh01]. However, this approach falls short of modeling the true value of product line development. This is the case as we are discussing here multi-year planning. In industrial practice, the situation of on the market can change very rapidly. We therefore see often that what products to develop can be impossible to predict within a one year time-horizon, let alone a multi-year time-horizon as this kind of strategic decision calls for. Thus, without taking this uncertainty into account we are missing a major aspect in the valuation of the product line. On the other hand, a major reason for switching to product line development is the flexibility in product development which entails that new products can be brought to the market within a shorter time frame with less effort. This enables the organization to react more flexible to market developments. Therefore, we see that uncertainty actually comes in two disguises: opportunity (to be able to develop a product not explicitly planned for during product line planning) and risk (that a product taken into account during planning will not be developed).

For the valuation of uncertainty typical approaches are decision tree techniques or option approaches. In decision tree approaches we try to explicitly enumerate and valuate the different possibilities for the different product developments and their respective probabilities [Wit96]. Option value approaches directly address the evaluation of the value of flexibility [FFF98]. An individual option can be seen as a special case of a decision tree approach, however, the valuation approach is more sophisticated. Both techniques are integrated in the context of real options approaches [AK99, FFF98]. Therefore, these approaches are currently strongly gaining attention. Thus, we will use the option framework as a basis for our discussion of the next two positions. In order to do so, we need to introduce some terminology: an option is the possibility to buy (or sell) something at (or up to) a later point in time, the *strike date*. The something that can be traded is the *underlying*. And the fixed price at which it can be traded is called the *strike price*. The difference between the value of the underlying and the strike price is the benefit that can be acquired. Such an option has obviously a value, i.e., an *option price*.

3 PRODUCT LINE DEVELOPMENT PROFOUNDLY ALTERS THE DECISION MAKING LANDSCAPE FOR STRATEGIC PLANNING

Above we discussed that the uncertainty whether or not we are going to develop a product line will impact the valuation of the product line approach and thus the strategic decision. This is the approach typically taken in determining the value of an option [FFF98, AK99]. However, in terms of flexibility, product line development has a second major impact: the development time for a product is shortened (cf. Figure 1). If we regard the point in time when a product is supposed to be released as fixed, then we can interpret this as a delay in the point in time when the decision needs to be made. Using the terminology of options, this implies that the strike date is delayed through reuse.

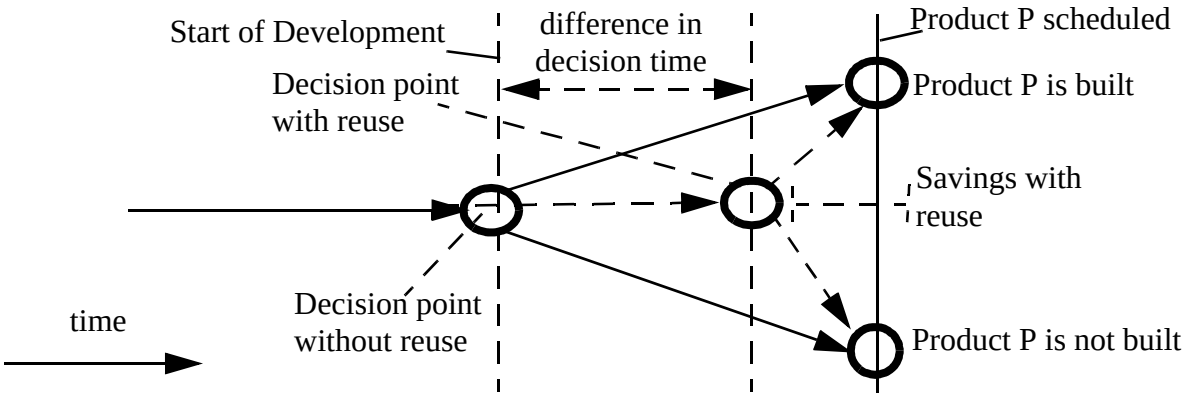

Figure 1. Decision Point in Time and Its Relation To Software Reuse

By reusing the benefit from the products (in options jargon the underlying) can be raised, by being able to release products faster. Thus, the reuse infrastructure directly influences the value of the underlying as well as the strike date.

The interpretation of faster product releases as a delay in strike date becomes particularly important when using option approaches for scoping product line development, as each addition of a feature implementation to the scope can be interpreted as exchanging an existing option (the option of developing the products with the current reuse infrastructure — and thus associated cost-savings) for an option with a delayed strike date (the option of taking more parts directly from the new reuse infrastructure).

This interpretation is symmetrical to the situation when we regard the decision point as fixed and aim at fielding the product earlier. As this is usually expected to raise higher revenues on the market, we can interpret an expansion of the reuse scope as exchanging an option against another option with a higher value of the underlying. Both possibilities of determining the value of the reuse option obviously differ, as they refer to different decision-making scenarios. In particular, in the later interpretation we introduced the market value/revenues for the product as a decision-making parameter. This is a major extension of the typical existing approaches to reuse economics [Fav96, Lim96, Pou97], which do only refer to the saving of development costs (which is basically time-independent). However, this interpretation is quite standard in the context of option approaches.

Thus, we come to the conclusion, that either way the introduction of reuse in the development scenario alters the decision making scenario, which can be either interpreted as a change in the decision time or in the decision value. This change has to be explicitly taken into account in order to arrive at an appropriate valuation of a reuse scope, respectively the decision for product line development. However, this aspect is usually missing [Coh01] and should be integrated into methods for strategic planning of software products and projects.

4 SOFTWARE DEVELOPMENT AND MARKET ASPECTS INTERACT AND SO SHOULD BE PLANNED IN AN INTEGRATED MANNER

So far we treated the reuse decision as a strict consequence of the product line, i.e., the products that ought to be developed. We will illustrate that this dependency actually goes both ways: envisioned product line and software implementation influence each other. Actually, situations where entering a reuse initiative is only beneficial because of the additional opportunities this situation implies are imaginable. The example provided in Table 1 illustrates this point. (The data was explicitly constructed to provide a simple illustration.) In this table four products, their cost for development with reuse and without reuse and the revenues that are expected for them are given.

Let us abstract for the moment from the phenomenons described in the previous sections. In our example the questions are: which products should be developed and should they be developed with or without reuse?

Product	Revenues [Million $]	Reuse Investment [Million $]	Dev. Cost with Reuse [Million $]	Dev. Cost with-out Reuse [Million $]	Reuse Savings in Product Dev. [Million $]
Reuse investment	—	3.5	—	—	—
Product 1	2.0	—	1.0	1.8	0.8
Product 2	2.4	—	1.2	2.0	0.8
Product 3	3.5	—	1.6	3.0	1.4
Product 4	1.5	—	0.8	1.8	1.0

Table 1. A Hypothetical Product Line Business Case

For the products 1 through 3 the situation is rather clear, as those products will produce higher reveneues than they cost even based on single system development. The fourth product only achieves higher revenues than the cost for development with reuse as the costs for development without reuse of $1.8 Million are higher than the revenues of $1.5 Million. Thus we will develop this product only in a reuse-based situation. Should we now go for product line development? Now, a rather interesting situation occurs: the total benefit of development without reuse is $1.1 Million (sum of the revenues of the first three products – development cost without reuse for first three products: 7.9 - 6.8 = 1.1), while the total benefit of the first three products with reuse is $0.6 Million (sum of the revenues of the first three products– development cost with reuse for the first three products – reuse investement : 7.9 - 3.8 - 3.5 = 0.6). So the reuse investment does not pay because making the products without reuse would have led to a higher revenue. Thus, we would not vote for reuse. However, if we include the fourth product, for which an opportunity only exists in the reuse-based situation, we arrive at a total benefit of $1.3 Million (revenues of the four products - development with reuse of the four products - reuse investment: 9.4 - 4.6 - 3.5 = 1.3). Thus, the scenario with four products to be built with reuse is the most preferable scenario!

This illustrates the strong interaction between market aspects and the technology choice of product line development. However, there is even a major interaction, which we dropped in the discussion above, and which leads us back to the option models discussed in the beginning. Typically, in market-driven development, the introduction of the nth product will have a profound negative impact on the revenues that can be accrued with the first n -1 products in the same market. This phenomenon is known as product line cannibalization [MM94] and should be taken into account as antipodal effect to the positive effect of reuse savings for a single product. In an option model this can be interpreted as a reduction of the value of the underlying.

5 CONCLUSIONS

In this paper we discussed problems that should be addressed by an approach to determine an objective valuation of a product line. Arriving at an objective valuation is a precondition for a rational strategy for product line introduction, for developing a business case for product line development, and for deriving an optimal product line scope. In particular, we identified three main aspects that such an approach needs to take into account: uncertainty needs to be taken into account, the repercussions that product line development itself has on the decision making situation, and the interactions between the technology aspects and the market for the product line should be explicitly modeled. Typically, even with the existing applications of real option approaches these problems are only partially addressed. Within, the Café-project we are currently focusing on the development of an approach which has said characteristics and thus supports product line transition in a rational and benefit-maximizing manner.

REFERENCES

[AK99] Martha Amram and Nalin Kulatilaka. *Real Options — Managing Strategic Investment in an Uncertain World*. Harvard Business School Press, 1999.

[Coh01] Sholom Cohen. Predicting when product line investment pays. In *Proceedings of the Second ICSE Workshop on Software Product Lines: Economics, Architectures, and Implications,* 2001.

[Fav96] John M. Favaro. A comparison of approaches to reuse investment analysis. In *Proceedings of the Fourth International Conference on Software Reuse*, pages 136–145, 1996.

[FFF98] John M. Favaro, Kenneth R. Favaro, and Paul F. Favaro. Value based software reuse investment. *Annals of Software Engineering*, 5:5–52, 1998.

[Lim96] Wayne C. Lim. Reuse economics: A comparison of seventeen models and directions for future research. In *Proceedings of the Fourth International Conference on Software Reuse*, pages 41–50, 1996.

[MM94] Charlotte H. Mason and George R. Milne. An approach for identifying cannibalization within product line extensions and multi-brand strategies. *Journal of Business Research*, 31:163–170, 1994.

[Pou97] Jeffrey S. Poulin. *Measuring Software Reuse*. Addison–Wesley, 1997.

[Sch00] Klaus Schmid. Scoping software product lines — an analysis of an emerging technology. In Patrick Donohoe, editor, *Software Product Lines: Experience and Research Directions; Proceedings of the First Software Product Line Conference (SPLC1)*, pages 513–532. Kluwer Academic Publishers, 2000.

[Sch01] Klaus Schmid. An initial model of product line economics. In *Proceedings of the International Workshop on Product Family Engineering, 2001*, 2001. To appear.

[Wit96] Jim Withey. Investment analysis of software assets for product lines. Technical Report CMU/SEI-96-TR-010, Software Engineering Institute, Carnegie Mellon University, 1996.

Considering Product Family Assets when Defining Customer Requirements [1]

Günter Halmans, Klaus Pohl

University Essen, Software Systems Engineering,
Altendorfer Str. 97-101, 45117 Essen, Germany
Email: {halmans, pohl}@informatik.uni-essen.de

Abstract : *The success of a product family heavily depends on the degree of reuse achieved when developing product family based customer specific applications. If a significant amount of the customer requirements can be realized by using the communality and the variability defined for the product family, the reuse level is high; if not, the reuse level is low.*

In this paper we elaborate on the influence of customer requirements on the degree of reuse achieved within a product family. We argue that the level of reuse can be increased by considering the capabilities of the product family when defining the requirements for the customer specific application. We finally show how this can be supported by product family specific use cases.

1 Introduction

Software product families aim in decreasing the costs and time required to produce a customer specific product. In product family engineering one distinguishes between two development processes: domain engineering and application engineering (cf. Figure 1). In domain engineering the communality and the variability of the product family is defined. Shared assets are implemented such that the commonality can be exploited during application engineering while preserving variability [2]. During application engineering individual products are ideally being developed by selecting and configuring shared assets resulting form the domain engineering.

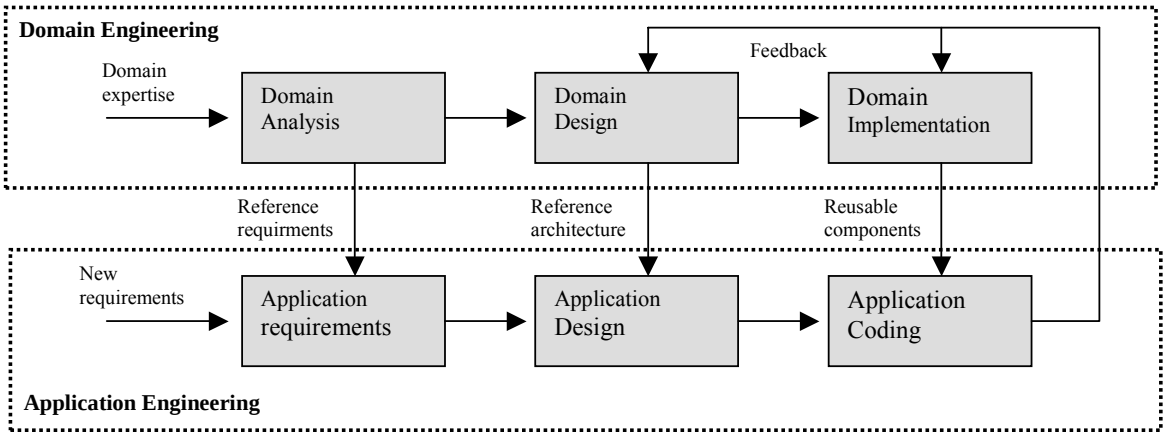

Figure 1: Domain and application engineering in product family development [1]

The success of a product family heavily depends on the degree of reuse achieved when developing product family based (customer specific) applications. If a significant amount of the customer requirements can be realized by using the communality and the variability defined for the product family, the reuse level is high; if not, the reuse level is low.

[1] This work has been funded by the ESAPS project (BMBF, Förderkennzeichen 01 IS 902 C/9 and Eureka Σ! 2023 Programme, ITEA project 99005).

As experienced in single product development, at the beginning of the requirements engineering process, customer requirements are often vague, i.e. the customer does normally not exactly know what she really wants. Requirements engineering is thus an iterative learning process in which the requirements engineer mediates the requirements definition among the stakeholders of the customers.

When making trade-off decisions during requirements engineering for a costumer specific application, the stakeholders should be aware of the consequences of their decisions with respect to capabilities of the product family. In other words, they should know which of the alternatives under consideration can be realized at low costs and in short time by exploiting the communalities and variability of the product family, and which of the alternatives conflict with the capabilities of the product family and thus lead to higher costs and longer development times.

Of course, an obvious approach to reach a very high level of reuse is to define a set of standardized product under the product family from which the customer can select his favourite. This works in situations where there is well-understood domain, and the understanding is shared by large parts of the potential market (customers). But even in those domains like financing (take SAP as an example), customer have specific requirements and want the standardized product to be adjusted to their needs. The need to develop products which do not dictate the requirements to the customer, but which satisfy the customers requirement is, according to our experience, increasing. As a consequence, also in product family development the customer has to be supported to identify and define his requirements – which from the product family provider point of view should largely correspond to the product family capabilities.

In this paper we identify what is required to empower the requirements engineer and the stakeholders to consider the capabilities of the product family when making trade-off decisions. (section 2). We argue that the level of reuse can be increased by considering the capabilities of the product family when defining the requirements for the customer specific application.

In section 3 we sketch two complementary types of support which help the requirements engineering in mediating between customer requirements and the product family capabilities. For supporting the trade-off decisions, we propose to classify each requirement with respect to its estimated realization effort. Therefore we propose seven categories. Each category represents a significant different way of realizing a customer requirements with the product family. Moreover, we propose use cases as means to provide product family specific information at the right abstraction level and discuss the extensions required to achieve this.

Finally in section 4 we summarize our contribution and provide an outlook on future work.

2 Requirements Engineer as Mediator between Customer Requirements and Product Family Capabilities

The first “phase” in the application engineering is eliciting, negotiating, documenting and validating the customer requirements for the desired customer specific application. Requirements engineering of product family applications differs significantly from requirements engineering in single product development. The main difference is that when performing requirements engineering within the application engineering process, one has to consider the capabilities of the product family based on which the customer specific application should be build. Reusing product family assets when realizing customer specific requirements reduces the costs and development time for the application and increases the return on investment for the product family provider; i.e. the more customer requirements fit with the product family capabilities, the higher the return on investment.

In the following we sketch the principle activities which empower the requirements engineering to provide product family information to the stakeholders during requirements engineering for a customer specific application (see also Figure 2).

First of all, the requirements engineer needs to know the capabilities of the product family (see (1) in Figure 2). Given that a industrial product families easy realize 1000 or even more requirements, and given that the realisation encompasses (several) hundred components. Moreover, existing variation points and their variants are not independent, i.e. there are dependencies between the variation points which have to be considered when exploiting the variation points for building customer specific

applications. Making the requirements engineer aware of the product family capabilities is thus by far not trivial.

Second, in the discussion with the customer the requirements engineer has to transfer the knowledge about the product family to the customer (see (2) in Figure 2). A customer has a different viewpoint on the product family than the requirements engineer. For example, a customer typically does not care about variation points, variants or binding times. In general, the knowledge exchanged in the task (2) is more abstract than the knowledge in task (1). The requirements engineer must thus continuously mediate between the customer and the product family capabilities. The requirements engineer must thus continuously "transform" the knowledge of the product family in terms the customer is able to understand. We will discuss possible support for this task in section 3.

Third, after the customer requirements are established, the requirements engineer has to transfer the result to the product developer. To support this task, the requirements engineer ideally provides some kind of requirements classification by which the potential realisation effort and the effects on the product family are indicated. Besides the realisation of the requirements, she should also provide some indication for requirements which cause a change to the product family, e.g. since they require new variation points or new components to be integrated in the product family.

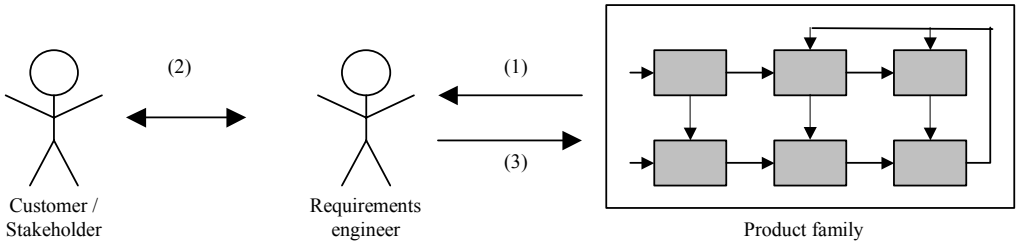

Figure 2: Requirements Engineer as Mediator

Due to space limitations, we focus on the second task (2), the mediation between the customer and the product family capabilities. However, the solutions described in section 3 provide an ideal basis to support the tasks (1) and (3) sketched above.

3 Communicating Product Family Capabilities to the Customer

We provide two types of support for the requirements engineer to communicate the influence of alternatives in trade-off decision to the customer:

- we categorize customer requirements with respect to their realization in a product family and sketch their use in supporting trade-off decisions (section 3.1);
- we describe how use cases which appropriate extensions could facilitate the mediation task of the requirements engineer (see section 3.2)

3.1 Realizing Customer Requirements under a Product Family: A Categorization

When realising (implementing) a customer requirement under a product family, one can distinguish between seven principle categories. Each of those categories stands for a significant different way of realizing a customer requirements with the product family. The effort required in terms of time and costs differs significantly for each of the seven categories defined in the following:

A: **Asset reuse:** Requirements which fall in this category can be realized by simply reusing a core asset of the product family; i.e. this category causes no implementation effort for satisfying a customer requirement;

B: **Binding a variant:** Requirements which fall in this category can be realized by choosing a variant provided by the product family for a given variation point and by defining the actual binding time of the requirement, e.g. during compilation, during runtime. To satisfy the requirement and adaptation of the configuration of the application might be required. However, the implementation effort required is almost zero;

C: **Creating a new variant:** Similar to B, the requirements in this category can be realized by binding a variation point. In contrast to B, the variant required to satisfy the customer requirement does not exist and must be realized. Depending on the complexity of the variant this could require some implementation effort; overall the implementation effort is low and does not effect the architecture of the product family. Moreover, the newly created variant can be used for future application development;

For realizing a requirement which falls into category A,B or C the variability of the product family is used and thus the implementation effort is very low. In contrast, the realization of requirements which fall in the categories D,E,F or G require a change of the core assets of the product family itself. Those, depending on the category and the change required, the implementation effort for realizing a requirement could be significant; in some cases it could be even impossible to satisfy the requirement with the product family (category G).

D: **Removing functionality:** Requirements which fall in this category require that some functionality or even components are removed for the product family and that this can not be achieved by a appropriated configuration of the variation points of the product family. In other words, the product family provides core functionality which is not desired by the application under development. If the functionality could be removed without changing the architecture of the product family (e.g. by replacing for the customer specific applications the product family components by customer specific components with reduced or restricted functionality) the requirements fall in this category. If the removal leads to a change of the overall product family architecture, the requirements fall in category F. The implementation effort required might differ significantly. However, in average a requirement in this category requires more effort than a requirement in category C, and more effort than a requirement in category E

E: **Adding functionality:** Requirements which fall in this category require that some functionality is added to the product family. Note that when adding the desired functionality could be achieved by using existing variation points or by implementing a new variant the requirements fall in category B or C respectively. Requirements which fall in this category those require the integration of new component(s) into the product family. In other words the implementation effort is the sum of the effort required for developing the component(s) and the effort required for integrating the components into the product family. . If the integration leads to a change of the overall product family architecture, the requirements fall in category F.

F: **Changing the architecture of the product family:** Realizing the requirements which fall in this category require a change of the product family architecture and those influence all existing product-family-based application. We do thereby not distinguish if a new functionality is added or if existing functionality is removed through the integration of new variation points and variants, or even a combination of both. The high development effort required to satisfy the requirements in this category mainly comes from the fact that the changes made influence already existing applications and lead to a new version of the product family architecture. The effort required to satisfy a requirement of this category is, in average, those significantly higher than for satisfying a requirement of the categories A-E:

G: **Impossible to realize**: Requirements which fall in this category essentially cause that the whole customer specific application can not be realized based on the product family. We call those requirements also “killer requirements”. In other words, the realization of a requirement of this category with the product family requires more effort than a realisation of the overall application without the product family. However, also in this case some assets of the product family might be reused when developing a product-family-independent application.

Figure 3 depicts the average level of reuse achieved when realizing a requirements of the seven categories defined above.

As already stated, the implementation effort required to satisfy a requirement of a given category might significantly differ depending on the “complexity” of the requirement itself. This holds especially for the categories D,E and F. For example, a requirement of category F could be lead to less

implementation efforts than a requirement of category D; especially if the requirement in category F is of less complexity as the requirement of category D.

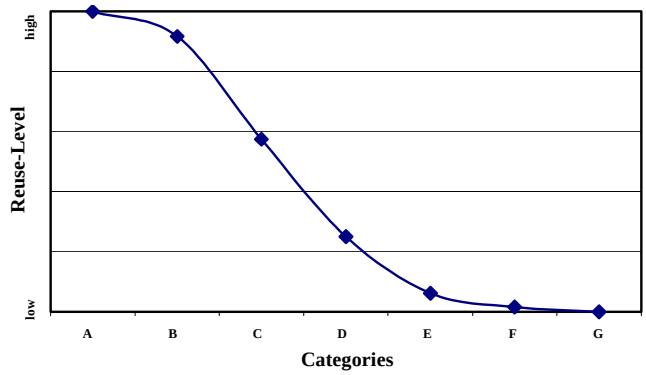

Figure 3: Categories and their Reuse-Level

However, when making trade-off decisions about alternatives for a given requirement, the alternatives under consideration are most likely of similar complexity and thus the difference of implementation effort within each category do not matter.

Classifying the requirements (alternatives under consideration) in the above categories is thus a good estimate for the realization efforts expected to satisfy the requirements. The categories provide excellent means to communicate the estimated realization efforts to the customer.

We thus use the categorizes for supporting trade-off decision during requirements engineering.

3.2 Extended Use Cases

The application of use cases has proven in practice and research to achieve better customer involvement and thus better requirements specifications. Use cases set requirements into a certain usage context and thereby reduce complexity and make requirements easier to understand.

Given this positive feedback from single product development – which is less complex than product family development – we suggest use cases for transforming information about functional capabilities, and their variability (variation points; variants) of the product family to the customer. To be able to apply use cases during application engineering in a product family context, however, some extensions are required. In other words, the use case templates proven successful in industrial praxis (cf. e.g., [3; 4; 5]) have to be extended to

- ***Capture variability:*** The variability within a use case has to be documented. All involved variation points associated with the actual use case have to be registered. For example in a use case template a section which points out the number of variation points can be added. Further more the variation points have to be represented in the use case diagrams. Also the activity diagram has to be extended, so that the consequences of binding a variation point in the dynamic of a process can be documented. Numbering all variation points enables the requirements engineer to check the possible alternatives within a given context which is important for the understanding by the customer.
- ***Capture binding times:*** In software product lines, variability is made explicit through variation points which are bound to a particular variant at a certain time (cf. [2] for details). For empowering the discussion of the requirements engineer with the customer it is essential to document these possible variants within a given context. So for every variation point which is numbered in the use case the possible variants and the binding conditions have to be listened and to be described. That means for every variant in the use case a main path scenario has to be described. Also for every variant main path an activity diagram can be defined and the related constraints have to be documented. With these information the requirements engineer is able to describe the particular alternatives within a use case.

- ***Document the consequences of the variation point binding:*** The several variants which can be realized for example by binding one variation point effects other use cases or other variants from other variation points within the use case. For describing a potential behavior of the product by concretization a variant it is necessary to know these effects. So in the use case template these associations have to be documented. This can be done by using extensions in the use case-diagram.
- ***Capture the categorization***: As mentioned in section 3.1 to categorize the requirements in different ways of realization support the requirement engineer and the customer in trade off decisions during defining the product. So these categorization of the requirements have to be documented. Normally several requirements are associated with one use case and therefore it may occur that the use case capture requirements from different categories. In an use case template, which numbers the associated requirements, the category of each of them should be documented by an additional attribute. In other cases the occurred categories have to be evaluated and a categorization of the use case may depend on which category the main requirements have.

Use cases are ideal facilitator for empowering the requirements engineer in the discussion with the customers stakeholder because they define a context with for example a special goal and given actors. This enables the requirements engineer to explain the variability in special scope. This is essential because of the complex product families. As mentioned in section 2 the categorization of the requirements makes it possible to give a first estimate of the related costs

4 Conclusion

Requirements engineering during application engineering has to consider the product family capabilities if the customer specific application should be realizable with a high degree of reuse. In this paper we have briefly discussed the challenges requirements engineering for customer specific applications faces in a product family context.

To support the requirements engineer in "mediating" the product family to the customer – for example when making trade-off decisions about possible alternative requirements - we have suggested

a) to classify the requirements with respect on their realisation efforts and times, i.e. the level of reuse achieved
b) to use apply extended use case to transfer knowledge about the product family, e.g. alternatives provided by variation points, to the customer.

We have thereby concentrated on one – out of three potential tasks – the requirements engineer has to perform when defining with the customer the requirements for a product-family-based applications. The solutions sketched in this paper, however, also support the other three tasks, e.g. if the product family is defined based on use cases, changes to a use case required by a customer can be captured as "deltas" and thus the realisation of the changes can be supported (see [6] for details).

We will apply the support sketched in this paper in an industrial setting. Although a validation appears to be non-trivial, we hope to be able to report on first results of this experiment soon.

References

[1] ITEA: 4th ESAPS Workshop on Derivation of Products and Evolution of System-Family Assets;2-4 April, 2001 Bad Kohlgrub

[2] Jan Bosch, Gert Florijn, Danny Greefhorst, Juha Kuusela, Henk Obbink, Klaus Pohl: Variability Issues in Software Product Lines, Fourth International Workshop on Product Family Engineering (PFE-4), Bilbao, Spain, 2001

[3] Alistair Cockburn: Writing Effective Use Cases; Addison Wesley, 2001

[4] Hans-Erik Eriksson, Magnus Penker, Buisiness Modeling with UML, Business Patterns at Work; OMG Press, 2000

[5] Putnam P. Texel, Charles B. Williams: Use Cases combined with BOOCH OMT UML; Prentice Hall, 1997

[6] Pohl, K.; Brandenburg, M.; Gülich, A.: „Scenario-based Change Integration in Product Family Development", 2nd ICSE Workshop on Software Product Lines: Economics, Architectures, and Implications, May 2001

Document Information

Title:	Proceedings of the PLEES'01 International Workshop on Product Line Engineering: The Early Steps: Planning, Modeling, and Managing
Date:	September 13, 2001
Report:	IESE-050.01/E
Status:	Final
Distribution:	Public