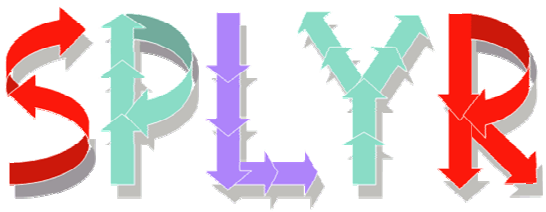
Fraunhofer Institut
Experimentelles
Software Engineering

Proceedings of the

Second International Software Product Lines Young Researchers Workshop (SPLYR)

Rennes, France – September 26th, 2005
In conjunction with the 9th International Software Product Line Conference (SPLC-Europe)



Editors
Birgit Geppert
Avaya Labs, USA
Isabel John
Fraunhofer IESE, Germany
Giuseppe Lami
ISTI - Italian National Council of Researches

IESE-Report No. 076.05/E
Version 1.0
September 10, 2005

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 competitive market position.

Fraunhofer IESE is directed by
Prof. Dr. Dieter Rombach (Executive Director)
Prof. Dr. Peter Liggesmeyer (Director)
Fraunhofer-Platz 1
67663 Kaiserslautern

Introduction: Family Planning beyond the Software: The second sibling

Different to many other software engineering techniques, product-line engineering arose from practical experience in industry. But as with every successful technique product line-engineering can also be considered only a real standard approach when it is not only applied in practice but also widely researched and especially taught in academia. Practice and academia are the two sides of the same coin. While industry sets the requirements, academia prepares the practitioners of tomorrow. The peculiarity of the SPLYR workshop is that it is specifically addressed to young researchers, in particular Ph.D. students, having original ideas and initiatives in the product line field.

The first SPLYR workshop was successfully held at SPLC 2004 in Boston, the second SPLYR workshop is now part of SPLC Europe 2005. Different from the standard procedure of most workshops, we have no blind peer reviews at SPLYR. Each student was assigned to the product line expert who reviewed the proposal and discussed pros and cons of the work with the student. We would like to thank Len Bass, Stuart Faulk, Stefania Gnesi and Robyn Lutz for reviewing the proposals and for the effort they spent. We selected four proposals for presentation at the workshop. The topics range from organizational issues to enterprise applications, variability management and evolution.

For many of the students this is the first time to present their work to an international audience. The workshop aims at providing the right platform for this and for discussing the presented work among each other and with experts in the field. We hope that with this workshop each of the students will get valuable feedback for the further development of the work.

The SPLYR Organizers

Birgit Geppert
Isabel John
Giuseppe Lami

Keywords : Software Product Lines, Software Product Line Young Researchers Workshop, Proceedings, SPLYR.

Organization

SPLYR is co-located with 9th International Software Product Line Conference (SPLC-EUROPE 2005) 26-29 September 2005 ,Rennes, France.

Workshop Chairs :

- Birgit Geppert
  Avaya Labs, Software Technology Research
  Basking Ridge, NJ, USA
  bgeppert@research.avayalabs.com
- Isabel John
  Fraunhofer IESE
  Sauerwiesen 6, D-67661 Kaiserslautern
  john@iese.fraunhofer.de
- Giuseppe Lami
  Istituto di Scienza e Tecnologie dell'Informazione "Alessandro Faedo"
  Area della Ricerca CNR di Pisa, Via G. Moruzzi 1
  Giuseppe.Lami@isti.cnr.it

Reviewers / Panelists:

- Len Bass
  Software Engineering Institute, USA
- Stuart Faulk
  University of Oregon, USA
- Stefania Gnesi
  ISTI-CNR, Italy
- Robyn Lutz
  Iowa State University, USA

Workshop website and email:

http://www1.isti.cnr.it/SPLYR/

SPLYR@isti.cnr.it

Table Of Contents

Introduction: Family Planning beyond the Software:
The second sibling 5

Organization 6

Table Of Contents 7

Workshop Program 9

1 *Gentzane Aldekoa, Goiuria Sagardui, Leire Etxeberria*
On the Evolution of the Production Plan 10

2 *Andreas Helferich*
Developping customer-oriented Enterprise Applications using
Software Product Lines and Quality Function Deployment 20

3 *Tao Zhang, Dong Xiang, Haipeng Wang*
vADL：A Variability-Supported Architecture Description Language
for Specifying Product Line Architectures 30

4 *Sven Raes, Prof. Dr. Ir. Frank Gielen, Prof. Dr. Ir. Piet Demeester*
Software Product Lines as the Basis of Managing Software and
Knowledge Assets across Research, Innovation, and Development
Organizations. 40

Workshop Program

09:00	**Welcome – Introduction to SPLYR**
09:15	**G. Aldekoa** - On the Evolution of the Production Plan (20 min. Presentation + 15 min. Discussion)
09:50	**A. Helferich** - Mass Customizing Enterprise Applications using Software Product Lines and Quality Function Deployment (20 min. Presentation + 15 min. Discussion)
10:30	*Coffee Break*
11:00	**T. Zhang** - vADL A Variability-Supported Architecture Description Language for Specifying Product Line Architectures (20 min. Presentation + 15 min. Discussion)
11:35	**S. Raes** - Software Product Lines as the Basis of Managing Software and Knowledge Assets across Research, Innovation, and Development Organizations (20 min. Presentation + 15 min. Discussion)
12:10	Final Discussion + Conclusions
12:30	*Lunch*
14:30	**Individual appointments with the panelists**

1 *Gentzane Aldekoa, Goiuria Sagardui, Leire Etxeberria*
On the Evolution of the Production Plan

On the Evolution of the Production Plan

Gentzane Aldekoa[1], Goiuria Sagardui, Leire Etxeberria

Computer Science Department
University of Mondragon
Loramendi 4, 20500, Mondragon, Spain
{galdekoa, gsagardui, letxeberria}@eps.mondragon.edu

Abstract. In Software Product Lines the production plan describes how products are built. The evaluation of the change impact becomes an important issue because the production plan evolves. Existing works on the evolution of software product lines have focused on other issues such as architectures, requirements, etc. On the contrary, the production plan has received little attention. To alleviate this situation, this paper provides a first review on the evolution of the production plan. First, the paper describes why changes occur due to distinct factors (i.e.: cultural, organisational, technological, etc) that force production plan evolution. Secondly, the work identifies the changes from the perspective of the production plan. Thirdly, several insights are presented about how changes occur. These ideas are inspired by previous works on product line architecture evolution. Based on these works, the aim of our work is to propose a methodology to manage production plan evolution.

1 Introduction

A **Software Product Line** (SPL) is *"a set of software-intensive systems, sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way"* [6]. SPLs explicitly distinguish between Core Asset development and Product development. The former sets up Product-line capability (creating the production plan) whereas the latter develops products (using the production plan). A core asset is "*an artifact or resource that is used in the production of more than one product in a software product line*" [6]. A core asset may be a requirement, an architecture, a software component, a process model, a test case, a plan, a document, or any other useful asset to build a system, i.e. anything used to produce a product on the product line. Furthermore, a production plan is needed in order to define how products are built. The **production plan** is "*the guide to how products in the software product line*

[1] 1st year Ph.D. student

will be constructed from the product line's core assets" [6]. This plan is used to build end-products.

The first studies focused on the software maintenance and **evolution** back in 1969. Lehman and Belady proposed in 1985 a set of laws or hypothesis known as Lehman's Laws [12]. These laws are based on the observation of the evolution of a great number of software systems. Lehman [13] and Parnas [16] agree that, without active countermeasures, the quality of a software system gradually degrades as the system evolves. As Lehman said [12] the **maintenance** is an inevitable process and when a system changes, its structure degenerates. Furthermore, evolution is more complex in a SPL setting [4].

SPL assets are developed to be reused for product production in the long-term. However, these assets evolve, and this evolution impacts on many products. Thus, not predicted evolution or not foreseen changes might cause an early retirement of the product line, and consequently a low return on investment or even not recovering the investment at all.

In previous studies, SPL evolution work has been focused on the requirements, SPL architecture [4, 17] and on product-specific architecture [2]. However, production plan evolves as other core assets. In general, the evolution of the production plan is a secondary effect of the evolution of other assets of the SPL (e.g.: the change in a COTS component may force the way in which the component is assembled). In addition, there are specific reasons that solely force the evolution of the production plan (e.g.: the change in the production strategy may only affect to the production plan).

SPL efforts are faced because of time to market, cost, and quality improvements[2]. These goals change due to SPL evolution. Likewise, the evolution of the production plan may impact in these goals. Nevertheless, the evolution of the production plan has not received enough attention, and this is the aim of this first approach. To this end, this paper presents the state of the development on the evolution of the production plan and highlights some issues related to its evolution.

The rest of the paper is structured as follows. Section 2 presents the current approach to the evolution of the production plan. Section 3 describes the ongoing work. Finally, future work is presented and some conclusions are given.

2 Evolution of the Production Plan

Concepts regarding change in organizations can be viewed from three dimensions [14]. The first dimension examines **why change occurs** ("*drivers of change*"). The second dimension concerns **what the changes are** ("*content of change*"). The third dimension regards **how change occurs** ("*process of change*").

[2] Organizations adopt a product line approach in order to achieve these goals but are not limited to [6].

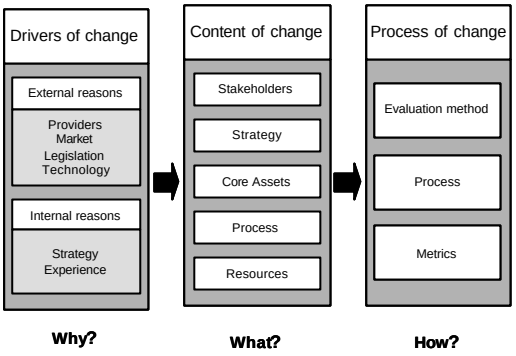


Figure 1 The three dimensions of change in production plan

These dimensions are used to analyze the evolution of the production plan. Firstly, drivers of change are the reasons that force evolution on a SPL. Secondly, content of change is any element of the production plan. These elements evolve due to previous defined drivers of change or secondary effects. Finally, the process of change analyzes how the drivers of change impact on the elements of the production plan and defines the steps to evolve the production plan. Figure 1 shows the elements for each dimension. These elements are elaborated later on and are a contribution of this paper.

Content of change is inter-related to drivers of change in two ways: directly and indirectly. The former forces **direct** changes in the production plan, whereas the latter forces changes in a core asset, which **indirectly** forces changes in the production plan.

Several parts of the production plan are affected when drivers of change evolve. Thus, it is important to analyze how drivers of change impact the production plan in a systematic way. To attain this, drivers of change, contents of change and process of change are to be considered when the evolution of the production plan is analyzed.

Next sections present these three dimensions in detail.

2.1 Drivers of Change

SPL evolution has been analyzed in different case studies [4]. Mostly, the reasons that force evolution are twofold: **internal** and **external** reasons to the organization. On the one hand, there are external reasons that cause product evolution or its retreat from the market. On the other hand, there are internal reasons which force the organization to evolve the product. External and internal reasons can be used to foresee the changes in the production plan (see Table 1). In addition, some other issues need to be studied when the content of a production plan (what) evolves due to drivers of change (why):

- *Impact degree*: states the impact level between a specific driver of change and a specific content of change.

- *Probability*: is the likelihood a specific driver of change forces a modification on a specific content.
- *Change origin*: states whether the change is directly forced by a driver, or is a secondary effect or indirectly.
- *Domain*: study how each specific domain impacts on above mentioned issues.

In the future, we plan to make a survey with distinct companies in order to validate our ideas with a gathering and analysis of information.

External reasons	
Providers	Software or hardware component providers, subcontracted companies, tools, etc.
Market	Users or system buyers, competitors, service organizations, etc.
Legislation	Certification agencies or the legislation itself.
Technology	Technologies offered by the providers to develop products, technologies used by the customers and technological standards.
Internal reasons	
Strategy	Economic, time and quality reasons.
Experience	Experience of the company in developing the products, having into account the skills of people and the knowledge about the product, structure, processes and methods.

Table 1 Drivers of change

2.2 Content of Change

Existing works say that a production plan can be from a **textual description** document [5] up to a **software program** capable of generating products automatically [3,7]. The organization states the production strategy and defines a production plan to achieve this.

The knowledge, users should have about the production plan, changes depending on the automation level of the production plan. Likewise, other parts may be affected.

Some SPL approaches propose variability in the production plan, considering a family of production plans [8]. Therefore, it is important to bear in mind all the ideas related to SPL evolution while evolving the production plan.

This section focuses on what the changes are within the production plan. To illustrate this, an outline of a textual description for a production plan proposed by Chastek and McGregor is used [5]. Instead of using the detailed outline of the production plan this research focuses on the major issues of the production plan, namely: stakeholders, strategy, core assets, process, and resources. Table 2 illustrates, the production plan major issues.

Major issues	Content
Stakeholders Defining a production plan, it is important to consider the whole people involved in the creation and operation of the production plan.	***Audience*** People who use the production plan and have responsibilities on it.
	Qualifications Knowledge that users should have about the production plan.
Strategy The strategy of the organization has to be established to define the production plan.	***Assumptions*** Ideas or assumptions about product lines development and evolution taken into account when creating the production plan.
	Qualities Quality attributes of the production strategy to satisfy the aim of the product line.
	Production context Production domain, brief description of products.
	Products possible from the available assets Relation between products and assets.
	Production strategy Strategy for producing products in the product line, and how concerns this strategy on the development of the product or on the same product.
Core Assets The production plan has to describe the core assets used to produce products, as well as their variability.	***Products possible from available assets*** Relation between products and assets.
	Basic inputs and dependencies Core assets list (code, non code assets, and tools) with their description.
	Variations Overview of the types of variation available for using the core assets.
Process The phases to develop a product have to be considered while creating the production plan.	***Detailed production process*** Phases or steps of the production process.
	Tailoring production plan to product-specific production plan Specific requirements and steps for the development of a product.
Resources This issue describes resources needed for the development of a product.	***Schedule*** Tasks to develop a specific product.
	Production Resources Description of human, material resources, and tools needed to produce the product.
	Bill of materials A list of core assets and their costs.
	Product-specific details Changes to be done to include a modification in a specific product.
	Metrics Metrics of a product.

Table 2 Production plan content

2.3 Process of Change

The process of change guides how change occurs. Until now, this has been an informal process. The main purpose of our work is to propose a controlled and systematic methodology. To attain this, we attempt to provide a methodology to drive the evolution of the production plan. This methodology consist of (1) an evaluation method to assess the production plan, (2) a process to guide the change of the production plan, as well as (3) some metrics to measure the whole process.

3 Ongoing Work

Svahnberg and Bosch present a product-line architecture evolution taxonomy [17]. This taxonomy provides six categories of requirements, eight categories of product-line architecture evolution, and five types of evolution that a component can be subjected to. The requirements are the *drivers of change*, whereas the others are the *content of change*. Inspired in this work, we are extending this taxonomy to the evolution of production plans based on categories defined for architectures by Svahnberg and Bosch. For instance, "*construct new product family*" requirement category [4] forces the organization to evolve the production plan:

1. Split of the production plan: in this case the new production plan uses the old one as template.
2. Derived production plan: the production plan derives some parts of the previous production plan in order to construct the new SPL. Both SPLs will share a common part of the production plan.

The impact of changes in the production plan is known after the evolution occurs. However, it could be of interest to foresee the impact before it happens. To attain this, an evaluation methodology is needed. Chastek and McGregor [5] propose some general criteria to evaluate the production plan: appropriateness for purpose, clarity, brevity, sufficient detail, internal modularity, internal and external consistency and traceability, usability, etc. However, it is also necessary to assure that the production plan provides the necessary modifiability to the SPL. To this end, the strategy of the production plan should be evaluated. The production strategy may have some quality attributes to assure that the production plan addresses the goals of the product line [5]. Modifiability is one of these quality attributes. The evaluation may guarantee that the production strategy is sufficiently flexible to allow evolution with the minimal effort.

Production plan evaluation is related to product line architecture evaluation; both evaluations should be done together. One of the most used techniques for evaluating SPL architecture modifiability is the scenario-based technique. A scenario is a specified sequence of steps involving the use or modification of the system [1]. This technique permits to define specific changes to be made to the system, and study how well the architecture responds to these changes. Scenarios allow evaluating abstract quality attributes such as modifiability within a context. There are several scenario-based

methods to evaluate software architectures: SAAM [6], ATAM [6], and etc. As well, there are specific methods for evaluating product-line architectures such as FAAM [10], AQA [9] [15], D-SAAM [11], etcetera. We are currently studying whether these methods are appropriate to evaluate the production plan with minimal modifications.

4 Future Work and Concluding Remarks

In a SPL setting, the evolution of core assets may be controlled and change impact may be analyzed. This paper presents an initial work to study production plan evolution. To this end, it defines why, what, and how changes occur. Our intention is to define a controlled and systematic methodology to manage the evolution of a production plan. The methodology will be evaluated with the study of industrial cases.

In this paper, different ideas about the evolution of the production plan have been presented. In addition, other interesting ideas should be studied in the future:

- How the production plan evolution affects SPL evolution.
- Changes in core assets force indirect changes in the production plan. This relationship should be carefully studied.
- Configuration management is important in any software system. This issue is well-known to manage core assets, but it should be considered to manage the production plan.
- There are different automation levels strategies for the product production. Commonly, this level increases as a natural evolution of the production plan. Thus, this issue may be extensively studied.
- This work is based on the analysis of the production plan and does not make emphasis on the product-specific production plan, which is left for future consideration.
- The design of the production plan has not been studied. However, we believe the more the resilience to change of the production plan the better the readiness to its evolution.

Acknowledgement

This work was partially supported by the Basque government, department of education, universities and research. Leire Etxeberria enjoys a doctoral grant of the research education program. Our gratitude to Stuart Faulk and Robyn Lutz for their valuable feedback to improve this prospective thesis work.

References

[1] Abowd, G., Bass, L., Clements, P., Kazman, R., Northrop, L., Zaremski, A.: Recommended Best Industrial Practice for Software Architecture Evaluation. Technical Report, CMU/SEI-96-TR-025 (1997)

[2] Ajila, S.A.: Change Management: Modeling Software Product Line Evolution. Proceedings of the 6th World Multiconference on Systemics, Cybernetics and Informatics, Volume VII, Information Systems Development II (2002)

[3] Batory, D., Neal Sarvela, J. Rauschmayer, A.: Scaling Step-Wise Refinement. IEEE Transactions on Software Engineering, 30(6):355-371 (2004)

[4] Bosch, J.: Design and Use of Software Architectures: Adopting and evolving a product-line approach. Addison-Wesley ACM Press (2000)

[5] Chastek, G., McGregor, J.D.: Guidelines for Developing a Product Line Production Plan. Technical Report, CMU/SEI-2002-TR-006 (2002)

[6] Clements, P., Kazman, R., Klein, M.: Evaluating Software Architectures: Methods and Case Studies., Addison Wesley (2001)

[7] Czarnecki, K., Eisenecker, U.: Generative Programming. Addison-Wesley (2000)

[8] Díaz, O., Trujillo, S., Anfurrutia, F.I.: Supporting production strategies as refinements of the production process. Software Product Line Conference (SPLC 2005), LNCS 3714, pp.210-221 (2005)

[9] Dobrica, L., Niemelä, E.: A strategy for analyzing product line architectures. VTT Publications (2000)

[10] Dolan, T.J.: Architecture Assessment of Information-Systems Families. Ph.D. Thesis, Department of Technology Management, Eindhoven University of Technology (2002)

[11] Graaf, B., Van Kijk, H., Van Deursen, A.: Evaluating an Embedded Software Reference Architecture –Industrial Experience Report. 9th European Conference on Software Maintenance and Reengineering (CSMR 2005), Proceedings. IEEE Computer Society (2005)

[12] Lehman, M.M., Belady, L.: Program Evolution. Process of Software Change. Academia Press, London (1985)

[13] Lehman, M.M.., Ramil, J.F., Wernick, P., Perry, D.E., Turski, W.M.: Metrics and laws of software evolution- the nineties view. Fourth International Software Metrics Symposium, (1997)

[14] Levy, A. and Merry, U.: Organizational transformation: Approaches, strategies, theories, Praeger. New York (1986)

[15] Matinlassi, M., Niemelä, E., Dobrica, L: Quality-driven architecture design and quality analysis method: A revolutionary initiation approach to a product line architecture. VTT Publications (2002)

[16] Parnas, D.L.: Software aging. 16th International Conference on Software Engineering, IEEE Computer Society Press (1994)

[17] Svahnberg, S., Bosch, J.: A Case Study on Product Line Architecture Evolution. 2001 Working IEEE / IFIP Conference on Software Architecture (2001)

2 *Andreas Helferich*
Developping customer-oriented Enterprise Applications using Software Product Lines and Quality Function Deployment

Developping customer-oriented Enterprise Applications using Software Product Lines and Quality Function Deployment

Andreas Helferich

Universität Stuttgart
Chair of Information Systems II (Business Software)
Breitscheidstr. 2c, 70174 Stuttgart, Germany,
Helferich@wi.uni-stuttgart.de

Software Product Lines enable companies to offer a number of variations of a given application by promoting planned reuse. Thus, the members of the product line can be tailored towards the needs of the customer while developing the systems is still economical. But treating a number of applications as a set of applications instead of as single applications does by itself not lead to increased customer-orientation. By incorporating the ideas behind Mass Customization, a Portfolio of Products based on a Software Product Line can fulfill the promise of individual products produced at relatively low cost. This requires strategic planning of the product portfolio. My PhD thesis focusses on using the well-established quality method Quality Function Deployment (QFD) to identify customer segments, and define members of the Software Product Line to satisfy the customer segments identified.

1 Introduction

The research described here is conducted under the supervision of Professor Georg Herzwurm, an internationally renowned expert on Software Quality Function Deployment. Combined with my own interest in Software Companies and their Product Management, the connection to Software Product Lines was natural. Having started 25 months ago, the method integrating Software Product Lines and QFD is already quite well-developed (a paper describing the method has been accepted for SPLC Europe 2005), but empirical evaluation in industrial projects is still lacking. Actually, the applicability of Software Product Lines for Enterprise Applications, especially advantages compared to other approaches in Enterprise Application Engineering, still needs to be demonstrated. For example, the monolithic "one size fits all" approach commonly associated with SAP R/3, the approach outlined by Microsoft for their planned Enterprise Resouce Planning System codenamed Microsoft Green (essentially arguing that the last 10% of the application will be programmed by the user) or the current trend towards architectures based on Web Services[1] need to be compared

[1] I am aware that Web Services and Software Product Lines don't contradict each other, but one might argue that using only Web Services was sufficient

to an Software Product Line-based approach. Further research needs to be done for some details in the method, most prominently

- Integration into existing Software Product Line Engineering Process Models,
- Statistical clustering method to be used: the proposed method (called QFD-PPP for Quality Function Deployment - Product Portfolio Planning) uses statistical analysis to identity customer segments, existing marketing literature on clustering methods to be used to this purpose still needs to be analysed,
- Business Case: as one reviewer for SPLC rightfully pointed out, the business case is currently a bit weak: comparing the costs of a product line member or component with the expected revenue and what Ulwick called *opportunity algorithm* [1], is a rather coarse course of action,
- Mass Customization (MC): taken to the extreme, offering variations of products based on standardized building blocks leads to MC, but the possibility for the end-customer to select his configuration himself is important for MC, and neither configurators nor generators for Enterprise Applications are currently advanced enough for wide-scale industrial use.

2 The Importance of Product Portfolio Planning

Product Portfolio Planning is a management activity closely associated with product development. Integrating information about technical innovations, market demand, cultural and legal developments, Product Portfolio Planning tries to develop a portfolio of products that optimally satisfies customer demands (thereby leading to increased sales) and at the same time restricts the number of products offered (thereby reducing costs and the risk of new products "cannibalising" old products' sales, i.e. customers buying the new product instead of an existing one). In an advanced stage, this includes planning for several product generations, taking into account technology S-curves and technology roadmaps [2].

For a (software) product line, product portfolio planning seeks to answer the following questions:

- Which products should be members of the product line?
- What technologies should members of the product line utilize?
- Which features/technologies should be common to all members of the product line?
- What should be the differences between members of the product line?
- In what direction should the product line and its members evolve?

From a business point of view, the answers are quite easy in theory: there should be as many different members of a product line as are necessary to satisfy the needs of the customers in the planned, profitable market segment. The common "core" consists of all features common to all members of the product line. The differences result directly from the different needs of different customers in this market segment. And the technology used is the one best satisfying customer needs (including the need

"reasonable price", i.e. only solutions that can be built at a cost allowing the producer of the product line to sell the products at a "reasonable price" are part of the solution space).

In practice, none of these answers is easy, since customer needs are not easily identified and prioritized. The latter is necessary since some customer needs are conflicting, e.g. ease of use and a multitude of functions. Kano's *Attractive Quality Model* [3] provides some insight why even the customers themselves have problems stating their true needs. According to the model, customer needs can be classified into the three categories: *Must-be* or *Basic Attributes, One-dimensional* or *Performance Attributes,* and *Attractive* or *Exciting Attributes*. And according to Kano, only Performance attributes are voiced by the customer since he takes Basic Attributes for granted and Exciting Attributes are neither required nor expected by the customer. But nevertheless identifying and fulfilling the latter (e.g. breakthrough innovations) leads to great satisfaction and the willingness to pay a premium price [4]. Finally, it is important to notice that customer expectations change over time and today's attractive attributes can be tomorrow's basic attributes [3]. Thus asking (potential) customers to fill out a questionnaire is not sufficient, rather it is important to get a deep understanding of customer needs and cross-check with technological opportunities [5]. Research on software requirements engineering has come to another conclusion: since software is immaterial in nature, customers have big difficulties expressing their expectations before using the final product [6].

Quality Function Deployment can be used to answer the questions that are part of Product Portfolio Planning and overcome the problems associated with identifying customer requirements for software, as will be shown in the following.

3 Product Portfolio Planning using QFD

3.1 Quality Function Deployment

"QFD provides a systematic but more informal way of communication between customers and developers" [7] compared to traditional ways of formalizing and specifying product requirements. A project team consisting of customer representatives, developers/engineers and a moderator who is an expert in QFD works together during the whole QFD process. This is done in order to assure that the final product's features are not determined by the technically possible but by the fitness for use, i.e. the features the customers demand. The software developers and/or engineers assure the technical and economical feasibility of the features and that technological breakthrough innovations are not ignored.

The best known instrument of QFD is the so-called House of Quality (HoQ). Generally speaking, the HoQ is the matrix which analyzes customer requirements in

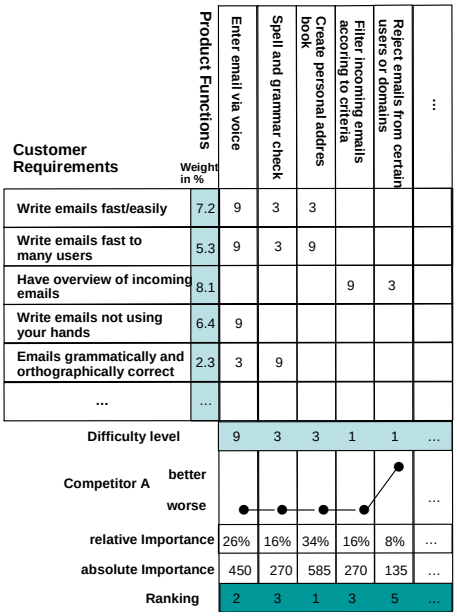


Fig. 1. Software-HoQ for an email client (adapted from [11]).

detail and translates them into the developers' language. The HoQ is the framework of most of the matrices used in QFD. For an in-depth description of QFD see [8].

QFD has been developed in the Japanese manufacturing industry [9], but can easily be adapted towards software development if two differences are considered: first, the software production process is basically a duplication process and implementation is largely determined by the system design, especially the system architecture. Therefore, the effort has to be directed mainly into the earlier stages. Secondly, "Software […] is valued not for what it is, but for what it does" [10]. Thus, the distinction between product function and quality element has to be made: a product function is a "functional characteristic feature of the product, usually not measurable (creates perceptible output)" [11], while a Quality Element is a "Non-functional characteristic feature of the product, possibly measurable during development and before delivery (does not create perceptible output)" [11]. The important first purpose of QFD in software engineering and the main focus of product planning is on setting prioritized development goals based on the most important customer requirements [11]. In planning software products the preference setting and focusing aspects of QFD by means of the HoQ are more important than the deployment by a matrix sequence. Applying QFD, however, takes more than filling out a HoQ matrix. A number of techniques (e. g. the Seven Management and Planning Tools and the Seven Quality Tools [11]) have to be combined in order to get all information that is necessary to form the matrices and to exhaust the potential of QFD as far as possible.

The entire QFD process is carried out by a team with representatives of all departments (development, quality management, marketing, sales, service etc.) and is to be extended in several team meetings by the selected typical customer representatives. Substituting a customer survey, one of the first meetings tries to ascertain customer

needs and to classify them in the Voice of the Customer Table. These requirements are structured using affinity- and tree diagrams and weighted (e. g. by pair-wise comparison or the Analytic Hierarchy Process [12]) by as many members of the customer groups as possible under control of the customer representatives. The weights of the different groups are then used to calculate the average weight by calculating the average of the weights assigned by the customer groups weighted with the importance of the groups.

The second major input is the Voice of the Engineer Table, compiled by the QFD team, among them particularly developers, that includes the potential product functions. The classic HoQ also uses measurable quality elements. These are derived from the requirements by the developers. The relationships between product functions and customer requirements in both prioritization matrices are identified together with the customer representatives. Analyzing the effects that one product function has on the other product functions leads to the roof of the HoQ [8]. Figure 1 displays an excerpt of a Software HoQ for an email-client including the tables of customer requirements and product functions.

An overview of the whole software development process using PriFo QFD can be found in [11], pg. 87. This approach has been used to develop the Calendar function in SAP R/3© [11]. A variation of PriFo QFD called Continuous QFD (C-QFD) using templates and iterative development cycles has been used for electronic and mobile business systems [13].

3.2 QFD-PPP

Our approach to Product Portfolio Planning makes extensive use of QFD while at the same time introducing two new matrices. First of all, the Voice of the Customer (VoC) is collected by asking existing and potential customers about the requirements they have for the product line. Once these answers are collected, they are analyzed and sorted before asking the customers to assign priorities to all requirements. Once these priorities are assigned, customer segments are derived based on these priorities using cluster analysis. Thus, unlike in PriFo QFD, there is no weighting of customer groups as this is only necessary to come up with common priorities. Another difference to PriFo QFD is the identification of customer groups not by attributes of the customer (e.g. job title or role description) but by statistical analysis.

The next step is to bring together developers, software architects and selected customers (based on the clusters identified) to build the Software House of Quality. Explicitly including the Voice of the Engineer in the form of product functions is important to identify exciting attributes according to the Kano model, i. e. software characteristics that customers themselves would not have come up with. Since a product function's level of fulfilling a customer requirement is independent from the weight assigned to the requirement, there is only one SW-HoQ for all the members of one product line. But since the weights of the customer requirements depend on the customer segments, the weight of the product functions does so either. The Software-HoQ in Figure 1 equals the Software-HoQ for one of the customer groups (including the weights), e.g. attorneys used to dictate letters who would therefore being able to dictate emails, too. The resulting matrix, including all customer requirements and

customer segments, including the importance assigned to the requirements is shown in Figure 2.

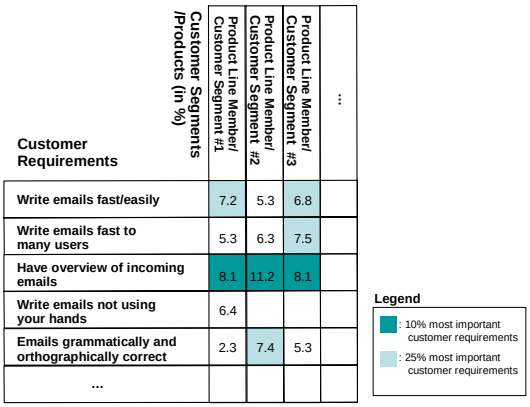


Fig. 2. Matrix Customer Requirements x Customer Segments

As indicated in Figure 2, the members of the product line are identified using the simple rule *one member of the product line per customer segment*. Core and variable features are identified by comparing the weight of the product functions for the different customer segments. This is visualized in the second new matrix: product functions x members of the product line displayed in Figure 3.

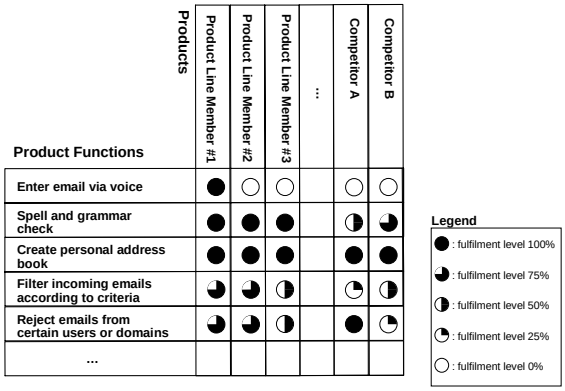


Fig. 3. Matrix Product Functions x Members of the Product Line

The software developers and software architects perform the next step evaluating different software architectures and technologies taking into account necessary quality attributes and product functions. This is also done by using matrices (Classic HoQ for the quality attributes, Software HoQ for product functions), where the roof is intensively used to analyze the impact that different architectural or technological elements have on each other. The results of this analysis are used to decide on the software architecture and the technologies to be used for prototypes.

These prototypes are then presented to the customers, thereby demonstrating exciting features the software developers and software architects came up with and the proposed solutions to the requirements voiced by the customers. Showing all customers all prototypes, some of the customers will decide to include some features they previously hadn't assigned value to, maybe drop some features they requested. This discussion is based on the product functions, not the original customer requirements and their weights. Only when large changes are asked for the customer requirements will be re-evaluated.

The second new matrix (product functions x members of the product line) helps prioritizing the variants. Inputs are the expected costs for the product functions and the expected revenue a product will achieve. The second depends on the size of the potential market, the products currently available on the market and the customer satisfaction with these products and the advantage the member of the product line have over these products. Ulwick's so-called *opportunity algorithm* [1] or the algorithm used in [11] can be used as indicators here. Both algorithms use the importance of a feature and the customers' satisfaction with the current solutions provided by own and competitors' products to identify features where improvements provide a competitive advantage. A more detailed economic assessment is presented in [14] and [15]. Figure 4 gives an overview of this part of the process (for reasons of clarity, classic HoQ, design-point analysis and the integration with systems design and implementation are omitted).

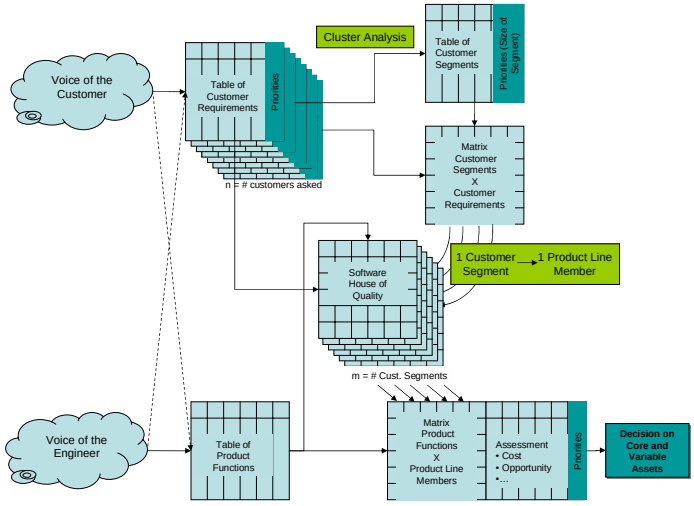


Fig. 4. Overview of QFD-PPP (simplified)

Finally, derivation of new products and the evolution of the Software Product Line and its members are facilitated, since the already existing matrices can be used as templates (a similar course of action for agile software development was proposed in [13]), leading to reductions in both time-to-market and costs, thus achieving important goals associated with Software Product Lines.

Also some of the more operational challenges in Software Product Line Engineering can be tackled using QFD: von der Maßen et al. identify challenges in the categories *Organization and Management, Requirements Engineering, Product- vs. platform-specific* and *Architecture* [16]. Some of these problems, most notably "high communication overhead", "Discussions on design and not on requirements level" and "No explicit prioritization of requirements" can easily be solved using QFD (see. [11] for problems in Requirement Engineering and solutions provided by Software QFD).

References

1. Ulwick, A.W.: "Turn Customer Input into Innovation", Harvard Business Review , 80(1) pp. 91-97 (2002).
2. Ulrich, K.T. and S.D. Eppinger. Product design and development - 2nd ed.. Irwin McGraw-Hill, Boston (2000).
3. Kano, N., Seraku, N., Takahashi, F. and S.Tsuji. "Attractive quality and must-be quality." In: The best on quality, edited by John D. Hromi. Volume 7 of the BookSeries of the International Academy for Quality. Milwaukee:ASQC Quality Press (1986).
4. Sauerwein, E.; Bailom, F.; Matzler, K. and H. H.Hinterhuber. "The Kano Model: How to delight your customers". In: Preprints Volume I of the IX. International Working Seminar on Production Economics, Innsbruck/Igls/Austria (1996), pp. 313 -327.
5. Aasland, K.; Blankenburg, D.; and J. Reitan. „Customer and market input for product program development". In: Proc. of the 6th Int. Symposium on QFD, Novi Sad, USA (2000).
6. Nuseibeh, B. and S.Easterbrook "Requirements Engineering: A Roadmap". In: Proc. of the Conference on The Future of Software Engineering, Limerick, Ireland (2000), pp. 35 – 46.
7. Herzwurm. G.; Schockert, S. and W. Pietsch: "QFD for Customer-Focused Requirements Engineering". In: Proceedings of the 11th IEEE International Requirements Engineering Conference, Monterey Bay, USA (2003), pp. 330-338.
8. Cohen, L.: "Quality Function Deployment", Addison-Wesley, Reading (1995).
9. Akao, J.: Quality Function Deployment: Integrating Customer Requirements into Product Design, Translated by Glenn H. Mazur and Japan Business Consultants, Ltd. Cambridge, Massachusetts (1990).
10. Zultner, R. E.: "Software quality function deployment – the North American experience". In: Software Quality Concern for people. Proceedings of the Fourth European Conference on Software Quality, Zürich (1994), pp. 143-158.
11. Herzwurm, G.; Schockert, S. and W. Mellis: Joint requirements engineering: QFD for rapid customer-focused software and Internet-development, Vieweg, Wiesbaden, Germany(2000).
12. Saaty, T. L.: Decision Making for Leaders: The Analytic Hierarchy Process for Decisions in a Complex World., 3rd edition, Pittsburgh, USA (1995).
13. Herzwurm, G.; Schockert, S.; Breidung, M. and U. Dowie: "Requirements Engineering for Mobile-Commerce Applications", in: In: Proceedings „M-Business 2002", Athens, Greece.
14. Schmid, K.: "Planning Software Reuse – A Disciplined Scoping Approach for Software Product Lines". Fraunhofer IRB, Stuttgart (2003).
15. Clements, P.C.; McGregor J.D. and S.G. Cohen: "The Structured Intuitive Model for Product Line Economics (SIMPLE)".Technical Report CMU/SEI-2005-TR-003 (2005).
16. von der Maßen, T. et al.: Key challenges in Industrial Product Line Engineering. In (Adelsberger, H.H. et al., ed.): Multikonferenz Wirtschaftsinformatik 2004 Band 1, Akademische Verlagsgesellschaft Aka GmbH, Berlin (2004), pp.260-272.

3 *Tao Zhang, Dong Xiang, Haipeng Wang*
vADL：A Variability-Supported Architecture Description Language for Specifying Product Line Architectures

vADL：A Variability-Supported Architecture Description Language for Specifying Product Line Architectures

Tao Zhang, Dong Xiang, Haipeng Wang

School of Computer Science, Northwestern Polytechnical University,
Xi'an 710072, Shaanxi, P. R. China

Abstract. Product line architecture describes reference architecture of a set of similar products, and is one of most important core asserts of product line. This article describes the Variability－supported Architecture Description Language, a novel ADL that has been designed to address specification of product line architecture. The variable component is used to modeling variability on architecture level, and the customized interface of component is used to modeling variability on component level. It supports layered compositions to refine architecture; especially variant binding and guard condition are provided to assemble variability. It has a formal, well-founded semantic based on the extended typed π-calculus, so it is able to specify both static and dynamic architectures. It also can be customized to drive architecture instance, which is used to describe architecture of product line member.

1 Introduction

Product line is a set of similar products which have lots of common features and some variability. Product line engineering has been divided into two stages: domain engineering and application engineering [1]. Firstly core assets are developed in domain engineering, then products are developed by reusing core assets in application engineering. Product line architecture describes reference architecture of a set of similar products, and is one of most important core asserts of product line. Product line architecture can be customized and reused to describe product architecture, so it must specify commonalities and variability of products architectures at the same time.

Traditional architecture description language, such as Darwin, Unicon, Rapide and Wright, provides inadequate description for the variability of product line architecture, so they can not specify product line architecture. Koala is designed by Philips corporation for specify product line architecture in consumer electronics domain [2]. Koala has some extra features aimed at handling diversity efficiently: interface compatibility, function binding, partial evaluation, diversity interfaces, diversity spreadsheets, switches, optional interfaces, and connected interfaces. xADL2.0 is a highly extensible XML-based ADL [3]. xADL 2.0 supports run-time and design time modeling, architecture configuration management and model-based system instantiation. xADL 2.0 provides variants and options features to specify variability of product line architecture. But Koala and xADL 2.0 only support a little type variability and provide limited mechanism for assembling and analyzing variability. They are also lack of semantic model, so they only support few analysis of product line architecture. Thus, we design a novel architecture description language: vADL.

The rest of this paper is organized as follows: Section 2 describes basic concept and syntax of vADL in detail, Section 3 defines a semantic model for describe behavior of component, Section 4

discuss relations of variable components, and Section 5 gives a simple example to illuminate vADL. The final section summarizes our conclusions and describes our future work.

2 Concepts and syntax

In vADL, architecture is described in terms of components, connectors, composition, architecture and instance.

2.1 Component

Components are units of design, development, and more importantly reuse. Granularity of component may be small or large.

A component communicates with its environment through function interfaces. Function interface contains a set of ports. Ports can be divided into two groups: in-port and out-port. Components receive values from environment by in-port, and send values to environment by out-port. Value can be data, port and behavior which pass through port.

Interior structure and behavior of component may be variable. Customized interface contain a set of variants which describe component lever variability. Different structure and behavior can be derived by different variant value.

Non-functional properties describe component QOS such as performance, running environment and required resource.

Component behavior is formally modeled with π-calculus theory, which can be as basic for analyzing behavior of architecture.

2.2 Connector

Connectors are special-purpose components. They are described as components in terms of function interface, customized interface, non-functional properties and interior behavior. However, their architectural role is to connect together components. They specify interactions among components.

Therefore, components provide the locus of computation, while connectors manage interaction among components. A component cannot be directly connected to another component. In order to have actual communication between two components, there must be a connector between them.

2.3 Composition

Components and connectors can be composed to construct composite elements, which may themselves be components or connectors. Composite elements can be decomposed and recomposed in different ways or with different components in order to construct different compositions.

Composite components and connectors comprise function interface, customized interface, non-functional properties (i.e. observable from the outside) and a composition of internal architectural

elements. Function interface of composite element must be mapped to function interfaces of internal elements.

2.4 Instance

Instance is run-time component or connector. Internal elements of Composite elements should be declared as instances. Variants in Customized interface should be set a fixed value when element is instanced. Then variable behavior and structure of element will be decided. Architecture of member of product line is an instance of product line architecture.

2.5 Architecture

Architectures are composite elements representing systems. Architecture can be seen as a special composite component. Architecture has some internal elements which be declared as element instances. Some instances are common instances which will occurred in architectures of all product line members, but others are variable instances which will only occurred in architectures of some product line members.

Guard condition is a Boolean expression, which contains variants in customized interface. Every variable instance accompany with a guard condition. Variable instance will be saved when value of guard condition is true, otherwise variable instance will be removed when value of guard condition is false.

Variant mapping define variant of instance as a function or relation which contain variants of architecture. Variant of instance can be calculated according to variants of architecture, then behavior and structure of instance will be fixed.

2.6 Syntax Definition

Extended Backus-Naur form (EBNF) is the basis for defining syntax of vADL, which is defined using the following notation for the abstract production rules:

- ✧ keywords are written with bold;
- ✧ non-terminals are written without bold;
- ✧ one or more elements is written: Element+;
- ✧ zero or more elements is written: Element*;
- ✧ one or zero elements is written: Element?;
- ✧ alternative choices are written separated by |.

ArchSpecification ::= ArchType ArchName {Interface Variability ? Property ? Implement}
ArchType ::= Architecture | Component | Connector
Interface ::= **Interface** { Port +}
Port::= **In|Out** PortName (ValueType)
Variability::= **Variability** {Variant+}
Property ::= **Property** {Variant+}
Variant ::= ValueType VariantName
ValueType ::= BaseType| ConstructType
BaseType ::= Any|Integer|Float|Double|Boolean|Char|String|Behavior

ConstructType ::=Set(ValueType)|Sequence(ValueType)|Tuple(ValueType1… ValueTypen)

Implement ::= Behavior| Compound

Behavior ::= **Behavior** { π Expression}

Compound ::= **Compound** { InstanceDeclare Assembly Attach}

InstanceDeclare ::=(ArchName InstanceName GuardCondition? VariablilityBinding?)+

GuardCondition ::= (Boolean Expression)

VariablilityBinding ::= VariantBind { (VariantName = Value| Expression)+ }

Assembly::= (ComponentInstanceName. PortName **LinkTo** ConnectorInstanceName. PortName)+

Attach ::= (InterfaceName **AttachTo** InstanceName. InterfaceName)+

3 Behavior Semantics

The π -calculus was proposed by Milner, Parrow and Walker in 1992, It is a name-based calculus of mobile processes [4]. Names can express ports and values of components, and processes can express behaviors of components. Table 1 gives definition of behavior semantics of vADL.

Table 1. behavior semantics of vADL

Behavior name	Behavior definition	π -calculus form	describe
Local Type	type typeName as ValueType	(typeName).behavior	Declare local type
Local Variant	value valueName	(valueName).behavior	Declare local name
Input Prefix	**receive** value **by** port	port(value).behavior	Receive value from port
Output Prefix	**send** value **by** port	$\overline{port}(value).behavior$	Send value to port
Interior	**interior**	$\tau.behavior$	Silent behavior
Condition	**if** boolean **then** {behavior1} **else** {behavior2}	[boolean]behavior1|[!boolean]behavior2	If condition is meet then do behavior1, else do behavior2.
Select	**select** {behavior1; behavior2; ….behaviorN}	behavior1+behavior2+… + behaviorN	Non-determinism select behavior, do any process
Combine	**combine**{behavior1; behavior2; ….behaviorN}	behavior1|behavior2|…| behaviorN	The processes runs in parallel, in other words they are doing things at the same time.
Repeat	**repeat** {behavior}	!behavior	infinitely repeat behavior
Null	**done**	0	Null behavior
Application	Application(Value1, Value2…ValueN)	Application(Value1, Value2…ValueN)	An abstract behavior。

4 Relation of variable component

Structure variability of product line architecture is implemented by variable components, variable connectors and guard conditions. By analyzing guard conditions, some relations of variable components are identified. These relations are benefit to analyzing and validating variability of architecture.

Definition1 Let G be the guard condition of variable instance A. We say that:

i. A is common instance, if $G \equiv true$;

ii. A is false instance, if $G \equiv false$;

iii. Otherwise A is variable instance.

Definition2 if anyone interior instance of architecture A is a false instance, and then A is redundancy.

Definition3 Let G_A be the guard condition of variable instance A, G_B be the guard condition of variable instance B, if $G_A = true \Rightarrow G_B = true$ and $G_A = false \Rightarrow G_B = false$, then B depend on A , mark as $A \to B$.

Theorem1 if variable instance A and B have relation: $A \to B$, and variable instance B and C have relation: $B \to C$, then variable instance B and C have relation: $A \to C$.

Proof Let G_A be the guard condition of variable instance A, Let G_B be the guard condition of variable instance B, Let G_C be the guard condition of variable instance C. According to $A \to B$ and $B \to C$, we can get $G_A = true \Rightarrow G_B = true$, $G_A = false \Rightarrow G_B = false$, $G_B = true \Rightarrow G_C = true$ and $G_B = false \Rightarrow G_C = false$. Then $G_A = true \Rightarrow G_C = true$ and $G_A = false \Rightarrow G_C = false$ **Hence** $A \to C$

Definition4 if variable instance A and B have relation: $A \to B$ and $B \to A$, then A and B are coexist. Mark as $A \leftrightarrow B$.

Theorem 2 if variable instance A and B have relation: $A \leftrightarrow B$, and variable instance B and C have relation: $B \leftrightarrow C$. Then variable instance B and C have relation: $A \leftrightarrow C$

Proof According to $A \leftrightarrow B$ and $B \leftrightarrow C$, we can get $A \to B$, $B \to A$, $B \to C$, $C \to B$. Then It is clear that $A \to C$ and $C \to A$ by **Theorem1. Hence** $A \leftrightarrow C$.

Definition5 Let G_A be the guard condition of variable instance A, G_B be the guard condition of variable instance B, if $G_A \Rightarrow !G_B$ and $G_B \Rightarrow !G_A$, then A and B are mutex , mark as $A \otimes B$.

Theorem 3 if variable instance A and B have relation: $A \leftrightarrow B$, and variable instance B and C have relation: $B \otimes C$. Then variable instance A and C have relation: $A \otimes C$

Proof Let G_A be the guard condition of variable instance A, Let G_B be the guard condition of variable instance B, Let G_C be the guard condition of variable instance C. According to $A \leftrightarrow B$, we can get $G_A \Rightarrow G_B$ and $G_B \Rightarrow G_A$. According to $B \otimes C$, we can get $G_B \Rightarrow !G_C$ and $G_C \Rightarrow !G_B$. Then $G_A \Rightarrow !G_C$ and $G_C \Rightarrow !G_A$. **Hence** $A \otimes C$.

5 Case Study

Figure 1 gives a simple example of Treater product line architecture. Component Filter can be customized to filter non-character or non-number. Component Encrypter can encrypt character or number. Instance filter is common instance, and Instance simpleLinker and encrypter are variable instance.

Treater has two variants of customized interface : charFlag and encrypFlag. When charFlag be set as “true” and encrypFlag be set as “false”, Treater is customized as instance treater1 in figure 2.

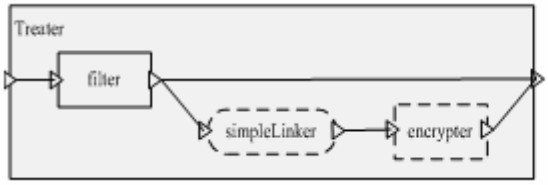


Fig. 1. Treater Architecture

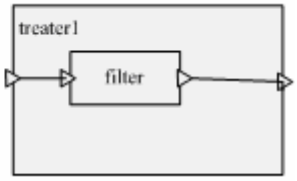


Fig. 2. Architecture Instance of Treater

```
Component Encrypter {
  Interface {
    Out portOut (Char);
    In   portIn (char)}
  Behavior {
     Value charValue
     receive charValue by
        portIn
     charValue = charValue
        +24
     send charValue by
        portOut}}

Connector SimpleLinker {
  Interface {
    Out portOut (Char);
    In   portIn (char)}
  Behavior {
    Value charValue
    receive charValue by portIn
    send charValue by
      portOut}}
```

```
Component Filter {
  Interface {
    Out portOut (Char);
    In   portIn (char)}
  Variablility{
     Boolean numberFlag }
  Behavior {
     Value charValue
     receive charValue by portIn
    if(numberFlag){
       if(0<=charValue<=9) then
          send charValue by
            portOut}
    else{
        if(a<=charValue<=z)
          then
          send charValue by
            portOut}    }}}
```

```
Architecture Treater {
  Interface {
     Out portOut (Char);
     In   portIn (char)}
  Variablility{Boolean charFlag
              Boolean encrypFlag }
  Compound{
    Filter filter VariantBind {
      numberFlag = !charFlag }
    Encrypter encrypter (encrypFlag)
    SimpleLinker simpleLinker (encrypFlag)
    filter.portOut LinkTo simpleLinker. portIn
    simpleLinker. portOut LinkTo encrypter. portIn
    portIn AttachTo filter. portIn
    portOut AttachTo filter. portOut
    portOut AttachTo encrypter. portOut }}
```

6 Conclusion

In order to specify product line architecture, a Variability－supported Architecture Description Language is proposed. vADL extends the framework of traditional ADL, and provides some variability mechanisms, such as: Customized Interface, Variable Instance, Guard Condition, Variant Mapping, etc. Then vADL is able to describe and assembly variability of product line architecture. A π-calculus behavior model is also provided, so analysis of behavior can be supported. In the near future, an Integration Development Environment of vADL will be designed and implemented.

References：

1. Clements Paul, Northrop Linda. Software Product Lines: Practices and Patterns. Addison-Wesley, 2001.
2. Ommering R V, Linden F V, Kramer J, Magee J. The Koala Component Model for Customer Electronics Software. IEEE Computer, 2000.33(3):78-85.
3. Dashofy E M, Hoek A , Taylor R N. An Infrastructure for the Rapid Development of XML based Architecture Description Languages. 24th International Conference on Software Engineering (ICSE 2002), 2002. Orlando, Florida: ACM.
4. Milner R. Parrow J. Walker D. A calculus of mobile process. Information and Computation. 1992.100:1-77

4 *Sven Raes, Prof. Dr. Ir. Frank Gielen, Prof. Dr. Ir. Piet Demeester*
Software Product Lines as the Basis of Managing Software and Knowledge Assets across Research, Innovation, and Development Organizations.

Software Product Lines as the Basis of Managing Software and Knowledge Assets across Research, Innovation, and Development Organizations.

Sven Raes, Prof. Dr. Ir. Frank Gielen, Prof. Dr. Ir. Piet Demeester

Ghent University - IBBT - IMEC
Department of Information Technology
Gaston Crommenlaan 8
9050 Gent, Belgium
sven.raes@ugent.be

Sven Raes is a 1st year PhD student

Abstract. During the last decade many universities and public research organizations (PROs) have been asked to add technology transfer to their mission statement. Research groups active in software engineering or research that produces a lot of software based assets (like tools, simulators, prototypes,...) will benefit from the strategic reuse mechanisms that are promoted by software product lines (SPLs). For those research organizations SPLs provide the means to capture the research results in a technology platform that has the potential to be applied in many vertical market segments. The introduction of SPLs as an R&D management strategy in software research management will allow to build a knowledge asset base and software technology platforms that can be transferred more easily and more efficiently to industry partners. To achieve this, the processes and the available methods that support SPLs need be to adapted to research environments and become part of the standard way that software research projects are managed.

1 Introduction

Adding technology transfer to the mission statement of PROs fundamentally influences the way research is managed as the capture and transfer of technology and competitive intelligence becomes more important. Software technology transfer isn't a success. Recent research shows that cooperation with PROs slows down the development process of new software products [9]. PROs active in software engineering produce a lot of software based assets like prototypes, tools,… They need a development strategy that stresses the goal of easy-reusable and easy-transferable, high-quality research results and software assets. SPLs are a software platform development paradigm allowing companies to realize order-of-magnitude improvements of several business drivers due to high-reuse and quality strategies [3, 5, 11]. The SPL

paradigm hasn't yet been introduced in PROs because their mission isn't to produce products. This paper explains how SPLs enable PROs to collect research result, building easy-reusable, licensable business technology and starting-points for industries to develop new innovative products. The positive effects of the SPL research approach on science to business technology transfer, together with the appropriate management techniques and new research culture are illustrated with a case-study that takes place at the INTEC Broadband Communication Networks Group (IBCN) at the Ghent University.

2 Relationship of Technology Transfer to the Architecture Business Cycle

2.1 Mission Statement of a Public Research Organization

Since the last decade both universities and companies are influenced by the open innovation model [7]. This model states that an organization is able to capture value from innovations if it can use the developed technology for leveraging current and future (research) activities, for licensing to other organizations or for the foundation of new spin-offs as valuable start IP. Borders of organizations start to disappear and new product development becomes 'cross-organizational' as many research, innovation and development organizations work together during the path from innovation to innovative product.

Within the open innovation model, PROs can perform an important role in developing valuable and innovative knowledge, building the basis of innovative products. For PROs active in software engineering this means the conversion of innovative ideas into concrete research results plus the proof-of-concept of the new idea in a working prototype. During this process a lot of other software based assets like tools and test applications arise.

2.2 Software Product Lines as Support for Technology Transfer and Software Related Research

The open innovation model implies a fundamental new role for PROs active in software engineering as they become the provider of new easily transferable knowledge and related assets. Four levels of maturity in the context of science to business activities can be observed. The first level is the awareness level. At his level technology transfer and innovation happen by accident: success depends on the heroic efforts of individuals and the organization takes ad-hoc initiatives to turn research into applications. During the repeatable innovation phase technology transfer and exploitation are introduced as formal activities in research projects. The structural innovation maturity level implies well documented prescribed practices and the motivation of all research organization members to follow the standardized innovation management process.

The structure of the organization reflects the strategic role of innovation management and all other management activities such as human resource planning are adapted to align the interest of individuals with this new mission. The most mature level is the optimized level when the organization is continuously monitoring and improving the innovation management process.

SPLs are a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of assets in a prescribed way [3] [5] [11]. A SPL strategy provides the processes to include technology transfer and exploitation as formal activities in software based research projects and creates an environment that can embed a complete metric and tracking system [5]. This implies opportunities for continuous monitoring and improving the innovation management process at the optimized level. Appling the SPL paradigm within PROs helps achieving this technology transfer maturity level and realizing the following open innovation requirements for developed software assets:

Reuse and Approved Quality. SPLs provide the basis for systematic software reuse. As new tools and research prototypes can be built from the same modules and interfaces, a huge amount of developed results is reused. Not only code is reclaimed, all surrounding knowledge is captured in the PRO SPL repository, providing the fundamentals for software and knowledge reprocess. This is an important advantage concerning the internal reuse of developed research results. Also the software industry profits from this approach as technology transfer takes place directly from PRO SPL repository to industry SPL repositories. Quality takes a very important place during product development. The introduction of SPLs within PROs implies a thorough and transparent software and document quality process. Assets are only posted in the repository after a profound quality control study which makes cooperation with PROs more appealing for companies.

Knowledge Management. A lot of the created knowledge within research organizations is tacit knowledge. This is knowledge that exists in the heads of people and that is not made explicit. As a consequence a lot of the tacit knowledge remains underutilized once a research project is finished. SPLs and their development environment result in a capture of knowledge that is easily reusable in derived research projects, research tools and applications. As a matter of fact, the aim of a public research SPL is to become a formal background intellectual property of the software based PRO.

License Model. The software architecture and reusable assets from the SPL can be licensed to other industrial partners. The management of intellectual property rights is a key success factor for technology transfer activities and is enabled by the encapsulation of knowledge in the PRO SPL.

High-tech Start-up Model. The software architecture and reusable assets from the SPL can be used as the intellectual property basis for new high-tech ventures. Creating spin-offs is an important part of the mission statement of PROs.

2.3 A Public Research Organization Software Product Line

Industry SPLs have a quite obvious evolutionary engineering life cycle [8] that consists of two major processes (Fig. 1). The SPL engineering process encloses all SPL related engineering activities, resulting in the SPL repository. The software application engineering process uses as much as possible software assets from the SPL repository and results in a new, derived product. The start point of the SPL construction process can be a forward evolutionary engineering approach, starting from nothing to build a SPL, or a backward evolutionary approach, starting from available software to build a SPL [8]. The mission of PROs is not to produce products as such. Therefore the model above can't be just 'copied and pasted' to this new environment. Software based PROs have to perform research, resulting in new knowledge and technology that can be easily reused, licensed or applied as start IP for new spin-offs (cf. open innovation model).

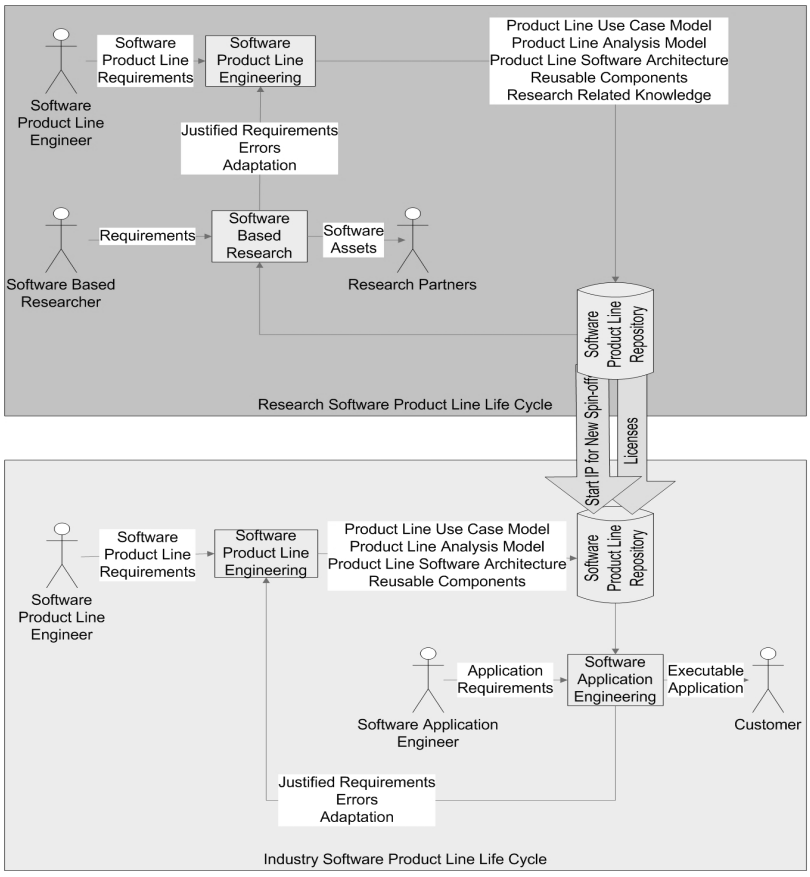


Fig. 1. The *research software product line life cycle* and *industry software product line life cycle*. The *PRO software product line repository* is the basis for new *software based research*, *licenses* and the *start IP for new spin-offs*

This PhD research introduces an evolutionary SPL lifecycle at software based research organizations that consists of two major processes (Fig. 1). The first process is the software based research process. Within this process new knowledge, together with new tools, prototypes,… are developed. The software assets, together with the related documents like use cases, test scenarios,… have to be reengineered, adapted and reused in function of the core competences of the software based research group. These activities make part of the PRO SPL engineering process which results in a repository that encloses both developed knowledge and software asset base.

It is clear that the PRO SPL repository is not just a software repository. It is also a knowledge repository where all related documents like use-cases, requirements, test scenarios,… are stored. When reuse of assets in the research repository takes place (e.g. licensing a combination of software components) not only software assets are available but also all surrounding knowledge. Due to the PRO SPL engineering process a reusable software and knowledge database is constructed that has an approved quality level (on which a continuous improvement process can be applied) and a formal background IP repository. This repository is the technology, business and competitive intelligence basis for new innovative research, new licenses and the start IP for new spin-offs.

3 The Research Software Product Line Approach

3.1 Quality Requirements

Research SPLs are designed to speed up the development of new tools, applications, etc. and to simplify the technology transfer process. The following quality attributes play an important role during the development process.

Modifiability. The architecture of the research SPL needs to be robust with respect to new features, technologies and requirements. New features should be added at minimum effort. This is necessary because software engineering research is an ongoing process that aims to anticipate on new evolutions, five to ten years in advance of the industry.

Testability. Each subsystem of the architecture must be easily integrated and tested so that possible errors are quickly found, isolated and corrected. As the SPL is employed in a research environment, bugs should be tracked and repaired at minimum effort. Debugging of new code needs to be quick and uncomplicated.

Usability. The SPL is intended to improve reuse in research, innovation and industrial environments. The architecture and reusable assets must be straightforward to understand, to use and to modify.

Knowledge Capturing. The architecture and reusable assets must be a way to capture knowledge of different, related projects, hereby serving as a formal background intellectual property of the research group.

Supporting Core Competences. The SPL should support the research organization short and longtime goals. The attainment of goals should be easily testable by different metrics in function of the core competences of the research group.

3.2 Implementation

A first implementation step is the design of a technology research roadmap and an innovation research roadmap. The first map is responsible for directing the fundamental research activities of the software based PRO and doesn't influence the SPL lifecycle. Research results and developed software assets at this stage are not mature enough to be considered for technology transfer and included in the SPL lifecycle. The innovation research roadmap takes into account the available, developed technologies and established research results to guide future valorization of these technologies through technology transfer. The corresponding software assets should meet the SPL requirements (cf. supra) in support of an optimized technology transfer level.

The second step is to align the engineering process of the software assets that arise during software based research. This is necessary to get a uniform, transparent and reusable repository. Improving the software based research process has a strong impact on the PRO SPL engineering process as less effort is needed to expand the PRO SPL repository. The outcome of software based research should be reusable software components and knowledge. This is a change process with impact on the whole research group and needs some time before being totally deployed and accepted. Not every kind of research will be influenced by this change process, only those research activities that are included in the innovation roadmap.

The third step is the improvement of the PRO SPL engineering process. A comprehensive study needs to investigate how software based assets, developed at PROs affect the growth-path of organizations. This should help to get more insight in the requirements of the research SPL and standardize the innovation management process.

The final step is to design an overall quality improvement process concerning the evolutionary, incremental SPL lifecycle at software based research organizations geared to the evolving core competences of the research group. Several metrics, a tracking system and other quality improvement related activities based on CMMI [5, 10] need to be developed in order to make the appropriate improvements to the SPL strategy.

The outcome is a PRO that has an optimized technology transfer level. SPL provide the processes to introduce and monitor standardized actions in support of science to business activities and exploration. The research SPL lifecycle proves the repeatable character of these processes and leads to structured innovation. The innovation roadmap, next to the technology roadmap, clearly stresses the strategic importance of innovation in the PRO management and aligns all management activities (like re-

source planning) to the technology transfer mission. The SPL metrics and tracking system provide opportunities of constant monitoring and improvement of the innovation management process.

3.3 Architectural Design and Benefits

Examples of industrial software product lines [3, 5, 11] show a strict layering, meaning that interactions among layers are restricted. A module residing one layer can only access modules in its own or the next lower layer. The layers are ordered, with hardware-dependent, core layers at one end of the relation and application-specific layers at the other. The grouping of modules is roughly based on the type of information they encapsulate. Modules specific to a particular customer product form a layer also. New products arise at the highest layers of the model.

The challenge with research product lines is the fact that the outcome of the product line isn't concrete products but 'building blocks' for new innovative software assets that are undefined at designing time. This implies serious management and organizational challenges as the point where new assets can be derived is situated in every layer of the product line. Future research will give more insight in the best SPL architecture design in the context of science to business environments.

Benefits of applying a SPL strategy in science to business environments are the improved management of technology, business and competitive intelligence because of the provided structured innovation research processes (cf. SPL lifecycle). Intellectual property rights management becomes more transparent as the SPL repository contains easy patentable knowledge embedded in high-quality and reusable software assets (cf. SPL engineering process). Through the introduction of an innovation roadmap, next to a technology roadmap, the competence building process al PRO becomes better manageable, guiding the organization to a new business culture and to improved science to business marketing.

4 Case Study and Future Work

IBCN is involved in research on broadband communication networks, distributed software and advanced networking applications. The IBCN group is also part of IBBT (the Interdisciplinary Institute for BroadBand Technology). This is a virtual public innovation organization that groups all the best-in-class research centers of the Flemish region in the ICT-domain.

Within IBCN a first research SPL, the Telecom Research Software (TRS) [6], is developed. TRS demonstrates that the above theory creates value for PROs achieving their open innovation mission statement. The repository of TRS contains software code, use cases, UML diagrams, javadocs, test scenarios,… Knowledge and software assets are captured by means of a webpage (containing all related information) and several java.jar files, containing the source code. A first spin-off, Comsof, has started up from the TRS IP and a lot of the repository software assets and knowledge is constantly reused in new research.

Several other innovation research projects are at starting point. These projects are focusing on different, challenging research domains like E-Health, Mobility and Logistics, E-Government and E-Media. The contribution of the PhD research is to set a SPL environment for these innovation projects. This offers a solid basis to get a lot of data and fine-tune all related SPL activities. A layered SPL architecture where new software assets can be derived from every layer is a new concept at research environments which has to be evaluated and possibly approved during the next steps of this PhD research.

A last part is the evaluation of the open innovation process itself. Information about the number of software assets and innovative products derived from the research SPL needs to be gathered. This research will be done in cooperation with the Vlerick Leuven Gent Management School and will focus on the concrete business expectations concerning software assets transfer from PROs to the software industry and the effect of this technology transfer on the growth path of these companies. In this PhD research phase cooperation with IBBT will be of great importance as this innovation organization aims at the realization of demand driven research programs, focused on the development of generic knowledge. During these programs research SPLs will play a significant role as they provide a repository for all developed knowledge and software assets.

References

1. Bachmann, Felix; Bass, Len: Managing Variability in Software Architectures. Symposium on Software Reusability. Toronto, Canada, 18-20 May (2001)
2. Bachmann, Felix; Bass, Len; Chastek, Gary; Donohoe, Patrick; & Peruzzi, Fabio: The Architecture Based Design Method (CMU/SEI-2000-TR-001). (2000)
3. Bass, Len; Clements, Paul; & Kazman, Rick: Software Architecture in Practice. Addison-Wesley (1997)
4. Bass, Len; Klein, Mark; Bachmann, Felix: Quality Attribute Design Primitives and the Attribute Driven Design Method. 4th International Workshop on Product Family Engineering. Bilbao, Spain, 3-5 October (2001)
5. Carnegie Mellon Software Engineering Institute <http://www.sei.cmu.edu/sei-home.html>
6. Casier, Koen; Verbrugge, Sofie; Colle, Didier; Pickavet, Mario and Demeester, Piet: Using Aspect-Oriented Programming for Event-Handling in a Telecom Research Software Library, Eighth International Conference on Software Reuse. Madrid, Spain, 5-9 July (2004)
7. Chesbrough H.: Open Innovation: The New Imperative for Creating and Profiting from Technology. Harvard Business School Press (2003)
8. Gomaa Hassan: Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures. Addison-Wesley (2004)
9. Heirman Ans: From Invention to Innovation: A Study of Research-Based Start-Ups. Submitted to the Faculty of Economics and Business Administration in fulfillment of the requirements for the degree of Doctor in Applied Economics, Ghent University (2004)
10. Margaret K. Kulpa; Kent A. Johnson: Interpreting the CMMI: A Process Improvement Approach. Auerbach Publications (2003)
11. Software Product Lines Website <http://www.softwareproductlines.com>

Document Information

Title:	Proceedings of the Second International Software Product Lines Young Researchers Workshop (SPLYR)
Date:	September 2005
Report:	IESE-Report No. 076.05/E
Status:	Final
Classification:	Public