

*Pre-publication*

Proceedings  
of the  
First International Workshop  
on  
Software Product Management

September 12, 2006  
Minneapolis/St. Paul, Minnesota, USA,

In conjunction with the  
14th IEEE International Requirements Engineering Conference,  
September 11-15, 2006

**Proceedings Editors**

Johan Versendaal, Utrecht University, the Netherlands  
Christof Ebert, Alcatel, France  
Sjaak Brinkkemper, Utrecht University, the Netherlands

*Pre-publication. Final proceedings to appear in the IEEE CS Digital  
Library*

# First International Workshop on Software Product Management

## Program

- 9.00 Welcome and introduction of the workshop participants  
*Sjaak Brinkkemper, Utrecht University, Netherlands*
- 9.10 Software Product Management: Practices and research challenges  
*Christof Ebert, Alcatel, France*
- 9.30 On the Creation of a Reference Framework for Software Product Management:  
Validation and Tool Support  
*Inge van de Weerd, Sjaak Brinkkemper, Richard Nieuwenhuis, Johan Versendaal, Lex  
Bijlsma, Utrecht University, Netherlands*
- 10.10 Can Agility be Introduced into Requirements Engineering for COTS Component  
Based Development?  
*Kendra M. L. Cooper, University of Texas at Dallas, USA*
- 10.30 Coffee break
- 11.00 Challenges of Knowledge and Collaboration in Roadmapping  
*Sami Jantunen, Kari Smolander, Lappeenranta University of Technology, Finland*
- 11.45 Lightweight Replanning of Software Product Releases  
*Thamer AlBourae, Guenther Ruhe, Mahmood Moussavi, University of Calgary, Canada*
- 12.30 Lunch break
- 14.00 A Cost-Based Approach to Software Product Line Management  
*Holger Schackmann, Horst Lichter, RWTH Aachen, Germany*
- 14.25 Towards Context-Aware Product-Family Architectures  
*Mohammed Salifu, Bashar Nuseibeh, Lucia Rapanotti, Open University, Milton Keynes,  
UK*
- 14.50 Ten Misconceptions about Product Software Release Management explained using  
Update Cost/Value Functions  
*Slinger Jansen, Sjaak Brinkkemper, Utrecht University, Netherlands*
- 15.30 Coffee break
- 16.00 Towards Comprehensive Release Planning for Software Product Lines  
*Muhammad Irfan Ullah, Guenther Ruhe, University of Calgary, Canada*
- 16.20 Panel discussion:  
Getting best practices practiced: How to address product managers?  
*Panellists will be announced*
- 17.30 Workshop closing

# Proceedings of the First International Workshop on Software Product Management

## Table of Contents

Introduction to the first international workshop on Software Product Management <i>Johan Versendaal, Christof Ebert, Sjaak Brinkkemper</i> .....	1
Software Product Management: Practices and research challenges <i>Christof Ebert</i> .....	3
On the Creation of a Reference Framework for Software Product Management: Validation and Tool Support <i>Inge van de Weerd, Sjaak Brinkkemper, Richard Nieuwenhuis, Johan Versendaal, Lex Bijlsma</i> .	7
Can Agility be Introduced into Requirements Engineering for COTS Component Based Development? <i>Kendra M. L. Cooper</i> .....	17
Challenges of Knowledge and Collaboration in Roadmapping <i>Sami Jantunen, Kari Smolander</i> .....	20
Lightweight Replanning of Software Product Releases <i>Thamer AlBourae, Guenther Ruhe, Mahmood Moussavi</i> .....	28
A Cost-Based Approach to Software Product Line Management <i>Holger Schackmann, Horst Lichter</i> .....	36
Towards Context-Aware Product-Family Architectures <i>Mohammed Salifu, Bashar Nuseibeh, Lucia Rapanotti</i> .....	42
Ten Misconceptions about Product Software Release Management explained using Update Cost/Value Functions <i>Slinger Jansen, Sjaak Brinkkemper</i> .....	48
Towards Comprehensive Release Planning for Software Product Lines <i>Muhammad Irfan Ullah, Guenther Ruhe</i> .....	55

# Proceedings of the First International Workshop on Software Product Management

## Workshop Introduction

In today's competitive software markets it is of utmost interest to have winning products. The success of any software product depends on skill-full and competent product management. Software product management includes product requirements, release definition, product release lifecycles, creating an effective multifunctional product introduction team and - above all - assuring a winning business case. Indeed software product management is complex: there are many stakeholders, many responsibilities and no formalized education or body of (scientific) knowledge.

This workshop aims at increasing the body of knowledge for this specific area of requirements engineering by providing a forum to exchange ideas and publish results. It will build and shape the community of leading practitioners and research experts.

Given the relevance of product management in IT and software companies, and the rather unexplored scientific contribution in this field, the workshop will deliver a state-of-the-art overview of the available scientific and practical knowledge on software product management, as well as an overview of areas within software product management for further research.

The following topics are considered to be in the scope of the workshop:

- Product management and requirements management
- Release definition and roadmapping
- Product management processes
- Product families and product line management
- Portfolio management and product life-cycle management
- Subcontracting, partnering and incorporation of open-source components
- Product strategy definition and marketing
- Measuring and improving the performance of the product manager
- Product management skill and competence building
- Structured customer contacts
- Product management at SME's
- Tools for product management

The call for papers was distributed worldwide to the industrial communities in product management and product marketing and to several relevant research communities. The workshop committee received ten submissions, which were each reviewed by three members of the workshop program committee. Based on the evaluations of the reviewers the workshop organizers selected for presentation at the workshop eight papers, equally divided into four full research papers and four research in progress papers. The selected papers form the contents of these proceedings.

The workshop organizing committee wishes to thank Martin Glinz, Program Chair of RE'06, Mikio Aoyama, Workshop Chair, and Robyn Lutz, General Chair, for their support in arranging this first workshop on Software Product Management. Furthermore, we thank Mats Heimdahl for the local arrangements for the workshop.

This is the first formal research exchange meeting in the area Software Product Management ever held anywhere in the world. Given the relevancy of this field for society and industry we hope that it will serve as the start of a series of workshops to explore the field and to establish a body of knowledge. We are looking forward to establishing this with our colleagues.

We hope you will enjoy the workshop proceedings.

### **Workshop organizing committee and proceedings editors:**

Sjaak Brinkkemper, Utrecht University, the Netherlands

Christof Ebert, Alcatel, France

Johan Versendaal, Utrecht University, the Netherlands

### **Workshop program committee**

Sjaak Brinkkemper, Utrecht University, the Netherlands  
Kendra Cooper, University of Texas at Dallas, USA  
Christof Ebert, Alcatel, France  
Xavier Franch, Technical University of Catalunya, Spain  
Leah Goldin, Golden Solutions, Israel  
Remko Helms, Utrecht University, the Netherlands  
Tsvi Kuflik, The University of Haifa, Israel  
Cornelius Ncube, City University, UK  
Manuel Pumarada, SSA Global, USA  
Carme Quer, Universitat Politècnica de Catalunya, Spain  
Björn Regnell, Lund University, Sweden  
Günther Ruhe, University of Calgary, Canada  
Pnina Soffer, The University of Haifa, Israel  
Stéphane S. Somé, University of Ottawa, Canada  
Johan Versendaal, Utrecht University, the Netherlands  
Tony Wasserman, Carnegie Mellon West, USA  
Didar Zowghi, University of Technology, Sydney, Australia



**Software Product Management: Practices and Research Challenges**

IWSPM '06  
12. Sep. 2006

<http://www.cs.uu.nl/groups/OI/IWSPM/>

ALCATEL

All rights reserved © 2006, Alcatel

---

---

---

---

---

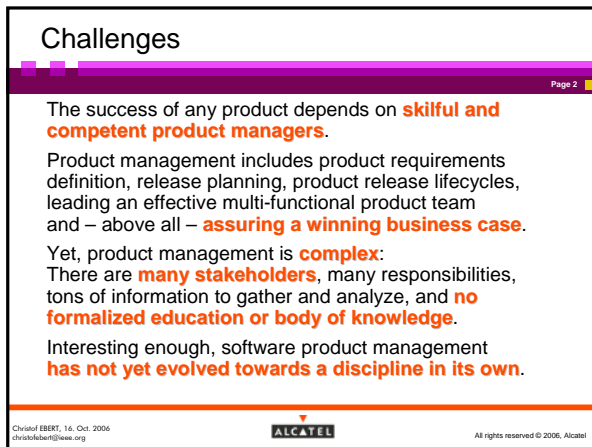
---

---

---

---

---



### Challenges

The success of any product depends on **skilful and competent product managers**.

Product management includes product requirements definition, release planning, product release lifecycles, leading an effective multi-functional product team and – above all – **assuring a winning business case**.

Yet, product management is **complex**: There are **many stakeholders**, many responsibilities, tons of information to gather and analyze, and **no formalized education or body of knowledge**.

Interesting enough, software product management **has not yet evolved towards a discipline in its own**.

ALCATEL

All rights reserved © 2006, Alcatel

---

---

---

---

---

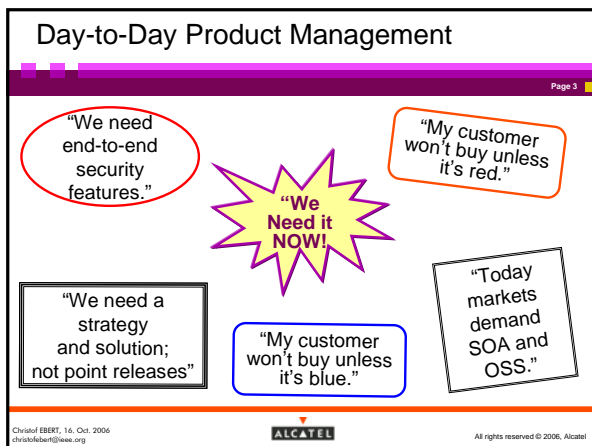
---

---

---

---

---



### Day-to-Day Product Management

“We need end-to-end security features.”

“We Need it NOW!”

“My customer won't buy unless it's red.”

“We need a strategy and solution; not point releases”

“My customer won't buy unless it's blue.”

“Today markets demand SOA and OSS.”

ALCATEL

All rights reserved © 2006, Alcatel

---

---

---

---

---

---

---

---

---

---

## Terminology

Page 4

### Product

- A product is a deliverable which creates value.
- It is a combination of materials and services that is produced, is tangible and can be either an end item in itself to be used or a component item used for other products.
- The term "Product" covers products, customer solutions, systems, services (e.g., consulting, install, operate, maintain), and the variants or versions of those.
- Products can be intended for general availability, limited deployment, pilots or prototypes.

### Product Management

- Product management is the discipline and business process which governs a product from its inception to the market/customer delivery in order to generate biggest possible value to the business.

Christof EBERT, 16. Oct. 2006  
christof.bernt@eesee.org



All rights reserved © 2006, Alcatel

---

---

---

---

---

---

---

---

---

---

## The Role of the Product Manager

Page 5

Product manager is **responsible for strategy and results.**

### He is the "Mini-CEO"

- He is accountable for definition, development, marketing and service of a product
- He understands customer needs and markets and positions the product accordingly
- He defines the business case, gets money to invest and delivers a ROI.
- He launches projects which deliver the committed results.
- He "lives" in middle management and has strong strategic and operational skills.
- He reaches his objectives without direct line responsibilities.

Christof EBERT, 16. Oct. 2006  
christof.bernt@eesee.org



All rights reserved © 2006, Alcatel

---

---

---

---

---

---

---

---

---

---

## Benchmarking

Page 7

- Chaos Report, 2003:  
34% of IT projects successful.  
15% of projects completely failed.  
2004/5 trend is worsening
- Chaos Report, 2003:  
Only 52% of the originally committed requirements appear in the final product.
- Ebert, 2004:  
Requirements change rate of 1-3% per month.  
Depends on market, positioning, novelty, etc.
- Cooper, 2004:  
Top 20% of enterprises deliver 79% of new products in time.  
Average delivers only 51% of projects in time.

Sources: Stanish Group 2004: study with 10000 Projekten per year; 58% in USA; 45% Fortune 1000; ca. 50% NPI 2006  
State of Embedded Market Survey; Cooper, R.G. et al: Benchmarking Best NPD Practices: Research - Technology  
Management, Part I, Jan. 2004, pg. 31, Part II: May 2004, pg. 43; Part III: Nov. 2004, pg. 43. Accessible via: www.apqc.org.  
Ebert, C. et al: Best Practices in Software Measurement. Springer, New York, Heidelberg, 2004

Christof EBERT, 16. Oct. 2006  
christof.bernt@eesee.org



All rights reserved © 2006, Alcatel

---

---

---

---

---

---

---

---

---


---

## Reasons for Failure

Page 8

### Bermuda Triangle of

- Sales  
"Sell today and cash in the bonus – agree it tomorrow."
- Marketing  
"We know what our customers want! We told 'em so."
- Product management  
"I am the only one understanding all details – and that's how it should be."
- Engineering  
"If the others would just be quiet. Our technology will sell itself."



Christof EBERT, 16. Oct. 2006  
christof.bernt@ees.org

ALCATEL

All rights reserved © 2006, Alcatel

---

---

---

---

---

---

---

---


---

---

## What Are Success Factors?

Page 9

1. Clear strategy, vision and objectives
2. "Mini-CEO" – empowered and accountable
3. "Voice of the customer": good understanding of market and customer needs
4. Contribution is visible from sales (top-line) to profits (bottom-line)
5. Assumptions are periodically checked
6. Risks are taken and managed
7. Dependable project management
8. Life-cycle management and milestones
9. Discipline, responsibility, decisiveness
10. Lean processes and standardized templates



Christof EBERT, 16. Oct. 2006  
christof.bernt@ees.org

ALCATEL

All rights reserved © 2006, Alcatel

---

---

---

---

---

---

---

---

---

---

## Trends in Product Management

Page 10

<p><b>External trends (society, technology)</b></p> <ul style="list-style-type: none"> <li>• Everything is fashion</li> <li>• Individualism</li> <li>• Focus on value</li> <li>• Ever-changing expectations</li> <li>• Service, service, service</li> <li>• Global competition</li> <li>• Economic and ecologic behaviors</li> <li>• Security and stability</li> </ul>	<p><b>Trends in product management</b></p> <ul style="list-style-type: none"> <li>• Focus on value (rather than price alone)</li> <li>• Understanding the voice of the customer</li> <li>• Dynamic segmentation down to the single-buyer segment</li> <li>• Building and systematically utilizing eco-systems</li> <li>• Innovation. Market and solution focus rather than technology alone</li> <li>• Flexibility. Detecting and managing uncertainties</li> <li>• Knowledge management</li> </ul>
--	---

Christof EBERT, 16. Oct. 2006  
christof.bernt@ees.org

ALCATEL

All rights reserved © 2006, Alcatel

---

---

---

---

---

---

---

---

---

---



## Research Questions to Address

Page 11

- How to cross-fertilize the product- and market-perspectives?
- How to deal (or cope) with uncertainty, changing assumptions and unknown requirements?
- How to optimize the duration from inception to break-even? What is "good enough" market and requirements analysis?
- How to evolve a single product perspective to a more global portfolio management perspective?
- How do parameters such as market, project type, product age, variation of changes, architecture, technology, platform alignment, culture influence overall product success?
- How to anticipate and foster disruptive innovations?
- Can knowledge management be more effectively used?

Christof EBERT, 16. Oct. 2006  
christof.ebrit@eesa.org



All rights reserved © 2006, Alcatel

---

---

---

---

---

---

---

---

---

---

## A Sayin from the Folks on the Continent

Page 12

**It mattereth not  
what thou doest,  
so long as thou dost  
cure the pain  
thy customer feeleth!**

Christof EBERT, 16. Oct. 2006  
christof.ebrit@eesa.org



All rights reserved © 2006, Alcatel

---

---

---

---

---

---

---

---

---

---

# On the Creation of a Reference Framework for Software Product Management: Validation and Tool Support

Inge van de Weerd, Sjaak Brinkkemper, Richard Nieuwenhuis, Johan Versendaal, Lex Bijlsma

Department of Information and Computing Sciences  
Utrecht University, The Netherlands  
{i.vandeweerd, s.brinkkemper, r.nieuwen, j.versendaal, a.bijlsma}@cs.uu.nl

## Abstract

*Software product management does not get as much attention in scientific research as it should have, compared to the high value product software companies ascribe to it. In this paper, we give a status overview of the current software product management domain by performing a literature study and field studies with product managers. Based on these, we are able to present a reference framework for software product management, in which the key process areas, stakeholders and their relations are modeled. To validate the reference framework, we perform a case study in which we analyze the stakeholder communication concerning the conception, development and launching of a new product at a major software vendor. Finally, we propose the Software Product Management Workbench for operational support for product managers in product software companies.*

## 1. Product management

In the past decades, the software market has made a shift from primarily developing customized software to developing software as a standard product. With this shift, a new function within product software companies emerged: the product manager function. In other industrial sectors, especially in manufacturing, product management has been established since the industrial revolution in the 19<sup>th</sup> century [29]. Recently, product software companies like Microsoft [18] and Alcatel [20] [21] [33] paid attention to product management as well. In addition, scientific literature has covered software product management [29].

Product management is of critical strategic value in many companies. However, it is also rather complex, since a product manager has many responsibilities covering requirements management, release definitions, and new product launches. What makes these responsibilities even more complex, is the fact that the product manager must take the many internal and external stakeholders into account [15] [46]. Although product management has been established for several decades,

software product management has some new challenges. Software products differ from other products in the fact that the manufacturing and distributing of extra copies do not require extra costs for the company [17]. Also, software products can be changed or updated relatively easy by using patches or release updates. The downside of these advantages lies in the fact that due to the nature of software products, the requirements organization is highly complex. Furthermore, the release frequency is high, since the product can be altered easily. Finally, a software product manager has many responsibilities, but does not have the authority over the development team. Because of these problems, we claim that it is necessary to integrate research efforts in this key domain.

In a few (software) product management areas know-how for research and educational purposes is available, but it is very fragmented. The domain is in need for an integrated body of knowledge, as exists in software development [10] and project management [37]. In this paper, we aim to develop a (preliminary) body of knowledge for software product management, by providing a reference framework for all its activities and deliverables. This reference framework has been based on an extensive overview of state-of-the-art literature, industrial case studies, and by exploring opportunities for operational tool support.

The organization of the paper is as follows. In the next section we elaborate on the rationale for the reference framework, and the research method we have applied to develop it. Then, in section 3, we discuss the basic structure of the reference framework. The four process areas are elaborated on in section 4. In section 5, we describe a case study at a major Enterprise Resource Planning software vendor. Subsequently, in section 6, we describe the Software Product Management Workbench, for operational tool support for the product manager. Finally, we describe our conclusions and future research.

## 2. Rationale and research method

In many fields, reference frameworks have proven to be valuable for research and practice. Examples are the ISO/OSI layers for the layering of network services [26]

and the ANSI/SPARC 3-schema architecture for database management systems [45]. The desire to get an understanding of the complete software product management domain can be satisfied by developing a reference framework. Both research contributions as well as developments in the software industry can be positioned in this reference framework. In this way, the consequences can be interpreted in a uniformed context. Also, the software product management reference framework can provide as a starting point for (a) a definition of key terms in software product management and the identification of open research questions; (b) the education of product managers and competence building; (c) the development of improved, integrated tool support.

The available industrial and scientific knowledge on software product management is limited and fragmented. Therefore, we use a proper mix of empirical and theoretical research steps for conceptualizing the reference framework, which are:

1. Field interviews and discussions with experienced product managers;
2. Literature review on both non-software product management as well as on software product management;
3. Creation of a draft reference framework;
4. Validation by an extensive case study at a large product software company;
5. Validation with input from an industrial workgroup on product management;
6. Finalization of the reference framework.

The resulting draft framework was adjusted several times after suggestions from practitioners and researchers. We do not claim that we now have produced the definitive version of the reference framework. Small enhancements might still be needed, but we are convinced that the basic structure has been established. The framework served furthermore as input for the design of the architecture of the product management workbench.

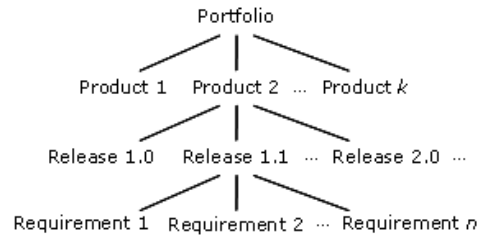
### 3. Basic framework structure

The nature of software products has a major impact on how the product management function is carried out. Therefore, we base the reference framework on its core, the software product itself, structured in a hierarchical way. Since part of the complexity is caused by the communication with the various stakeholders, we position them to reveal their interactions concerning product management.

#### 3.1. Artifact hierarchy

Professional software product management is in essence a matter of well-organized processing of issues related to

requirements, products and releases [19] [15]. A hierarchical ordering of these artifacts (see Figure 1) imposes a structure on the process areas.



**Figure 1. Artifact hierarchy of product management**

Starting on top, the scope of work of software product management concerns the complete set of products of the company, the so-called *product portfolio*. Small or young companies may have a portfolio of just one product, whereas larger companies have several, due to acquisitions and/or product derivation.

All products have a release sequence of past, present and future *releases*. The release numbering is usually determined by internal conventions, where major changes in the technical architecture are a reason to call it an X.0 release. Marketing reasons may lead to commercial numbering using the year of release or the same release code as an important customer.

Finally, each release definition consists of a set of selected *requirements*. Each requirement implies the addition of a technical or functional feature to the product. Non-functional requirements are also considered, such as performance constraints or availability requirements.

The type of work differs when dealing with artifacts from the distinct hierarchy levels. The hierarchy gives rise to a subdivision of software product management into four process areas: *portfolio management* to deal with the products in the product portfolio; *product roadmapping* to deal with the different releases each product has, also called roadmapping; *release planning* to deal with the set of requirements of each release; and *requirements management* to deal with the content and administrative data of each individual requirement.

Observe however, that for the sake of diagram clarity, we have swapped the positions of requirements management and release planning in the reference framework (Figure 2). Release planning processes communicate about complete releases to internal stakeholders, whereas requirements management interacts with all stakeholders.

#### 3.2. Stakeholder interaction

Software product managers are dealing with many requirements, originating from internal and external stakeholders. We distinguish the following internal stakeholders [15] [19]:

- The *Company board* is responsible for the definition and communication of strategy, vision and mission to the rest of the company. Also, it has the managerial supervision of the different departments, including product management. Occasionally, requirements are communicated through its strategy, but it can occur that a requirement is sent directly to the product manager.
- *Research & innovation* has two core responsibilities: (1) doing research to new opportunities for product innovations and (2) finding ways to incorporate improvements or new features into the existing products. The first one results in requirements in the form of technology drivers that are communicated to the product manager.
- The consultants of the *Services* department are responsible for the implementation of the software product at the customer organization. They need to be aware of new release features and they gather new requirements from the customers.
- *Development* has as main responsibility the execution of the release plan. The release definition also includes functional explanation of the product requirements that serve as input for the functional and technical design. It may occur that during the development process new requirements can arise, due to more complex requirements than was anticipated.
- *Support* stands for the helpdesk to answer questions (1<sup>st</sup> line support) and for small defect repair unit (2<sup>nd</sup> line support). Large defect repair is usually performed by Development (3<sup>rd</sup> line support).
- *Sales & marketing* is the first contact with a potential customer. Through these contacts new requirements can be gathered.

The following external stakeholders are recognized [32]:

- The *Market* is an abstract stakeholder, standing for potential customers, competitors and analysts, such as Gartner and Aberdeen. Numerous trends may be recognizable in the market, either in an explicit way by one of the market players, or in an implicit way by product management.
- Most companies have different kinds of *Partners*: (1) implementation partners, who implement the product at a customer; (2) development partners, with whom product components are developed; and (3) distribution partners, selling the product.
- *Customers* often have new feature requests in the process of closing the deal or during the usage of the product. These requests can be communicated to

Services, Sales & marketing, Support, but also directly to the product manager.

Observe that the stakeholder names are generic, so that naming or grouping may differ in product software companies. It is obvious that external stakeholders are harder to be influenced in their operational execution and decision making, whereas internal stakeholders should act according to the corporate strategy.

## 4. Reference framework

Little scientific literature explicitly addresses the software product management domain. Only some sub domains, like requirements engineering (e.g. [36], [38] and [39]) and release planning (e.g. [11], [28] and [42]) are covered. Vähäniitty [48] found that product portfolio management is largely overlooked in literature, and if it is addressed, it does not mention small and medium sized product software companies. Although software development is largely addressed it adheres to project-related development [24]. In this section we provide an overview of the existing state-of-the-art research on (software) product management.

Figure 2 shows the reference framework for software product management. The framework was developed after literature research and field interviews with experienced product managers that were employed at six product software companies from the Netherlands. The size of the companies ranges from 75 to 2,700 employees.

Besides the four process areas, we show the sub functions of the product management domain the relations with the internal and external stakeholders. The elements are connected with information flows, indicated with arrows. Note that these flows should not be read as a linear route, but as a continuing, iterative process. In the remaining of this section, each of the four process areas is provided with an explanation supported by research contributions.

### 4.1. Portfolio management

Portfolio management covers decision making about the set of existing products; introducing new products by looking at market trends and the product development strategy; making decision about the product lifecycle; and establishing partnerships and contracts. Product line management is positioned in this area as well. In [14], a software product line is defined as 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. Several case studies have shown that introducing product lines organizations improves performance [8] [9] [44]. They

are most popular in telecommunication organizations [33], but the last years, also the software industry pays more and more attention to this topic [1] [4] [14]. Some research has been done to tool support for product lines. An example is Laqua [30], who proposes a product line content & knowledge base on top of arbitrary configuration management system. Product lifecycle management is a comprehensive approach for product-related information and knowledge management within an enterprise, including planning and controlling of processes that are required for managing data, documents and enterprise resources throughout the entire product lifecycle [1]. This is a key process in decision making about the product portfolio. Also partnering & contracting are important issues in product management [7].

Portfolio management is placed on top of the reference framework. It contains the following main processes: partnering & contracting, market trend identification, product lifecycle management and product line identification. The Company board, Market and Partner companies provide input for this process area.

## 4.2. Product roadmapping

Roadmapping is a popular metaphor for planning and portraying the use of scientific and technological

resources, elements and their structural relationships over a period of time [47]. It is complex due to dependencies on other related products (even from partners), technology changes, and the distributed development [13]. The origins of roadmapping lie in the manufacturing industry. Here, it is used for business oriented long-term planning and technology forecasting [32]. In the product software industry roadmaps are used for planning purposes [47]. In [27] the term roadmapping is used in two perspectives: forecasting and planning. Forecasting concerns technology or market trends; and planning concerns products, product lines, resources or the entire company. We use the definition of [39]: a roadmap is a document that provides a layout of the product releases to come over a time frame of three to five years. It is written in terms of expectations, plans and themes and core assets [34] of the product.

As is illustrated in Figure 2, product roadmapping receives input regarding product lines from portfolio management. This input is used to identify themes and core assets. Themes are used to give a clear direction to the roadmap and later on to structure the requirements. Core assets are components that are shared by multiple products, for example an authorization function that is used by multiple software products. All information is gathered and described in the product roadmap.

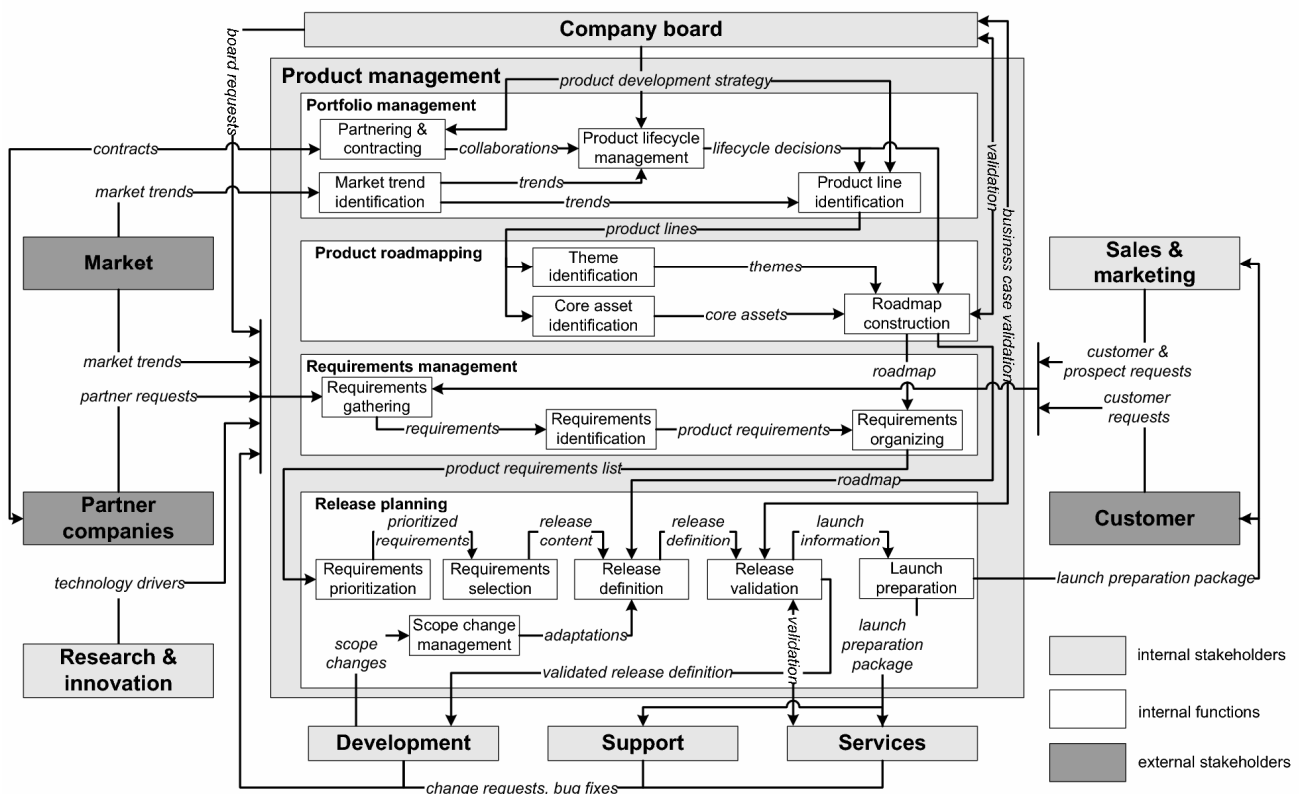


Figure 2. Reference framework for software product management

### 4.3. Requirements management

Requirements management entails the activities of gathering, identifying and revising incoming requirements and organizing them by keeping in mind dependencies, existing core assets, product lines and themes. Sources are customers, sales and marketing, development, support, R&D and the company's management.

Requirements management is a key area in product software companies [12], but [38] already recognized that requirements engineering for product software is different than for customized software. In [36], the following core requirements engineering activities are recognized: eliciting requirements, modeling and analyzing requirements, communicating requirements, agreeing requirements, and evolving requirements. Especially analyzing requirements costs a lot of time in product software companies, due to the (often) high requirements rate, and the different sources of requirements. An example is the use of linguistic engineering to link customer wishes to requirements [35]. Another problem is the integration of a software product with other systems. Customers cannot expect that all their requirements are met, which may lead to a software product that does not integrate with their existing systems. In [31] several improvements are suggested to this practice. In [20], the requirements process in 246 industry projects is investigated and the results show that four techniques improve schedule performance, if used in parallel: installing of an effective core team for each product release; focusing on the product-lifecycle on upstream gate reviews; evaluating requirements from various perspectives; and assuring a dependable portfolio and release planning implementation.

The position of requirements management in the reference framework is between product roadmapping and release planning. The process starts with gathering all requirements from within the company and from external stakeholders. The requirements gathered and organized into product requirements. Product requirements are identified by removing the duplicates, connecting requirements that describe a similar functionality, and by rewriting the requirements in understandable product requirements. Then, the requirements are organized per product and core asset. Also, the mutual dependencies between the different product requirements are described. In [35], a distinction is made between *market requirements*, which refer to wishes related to future products, defined in the customer's perspective and context; and *business requirements*, a product requirement to be covered by the company's products, described in the company's perspective and context. We use a similar distinction. However, we make a distinction

between *requirements* and *product requirements*. Requirements refer to all incoming wishes and change requests. These are not only market requirements, but also service requirements, board requests, technological drivers from research & innovation, etc.

### 4.4 Release planning

Software release management is the process through which software is made available to, and obtained by, its users [25]. Core functions in this process are requirements prioritizing; release planning; constructing and validating a release requirements document; and scope management

Much research has been carried out on the domain of release planning, where the set of requirements for the next release is determined. Examples are release planning using integer linear programming [1], the analytical hierarchy process [41], stakeholders' opinions on requirements importance [40] and linear programming techniques using requirement interdependencies [11]. More techniques can be found in [2] and [5].

In the reference framework, release planning starts with the product requirements prioritization. Not only the product management is responsible for this, but also the other stakeholders can influence this process. After the prioritization, product requirements are selected that will be implemented in the next release. This can be done in multiple ways: one can choose the product requirements with the highest priority or use integer linear programming to estimate the best set of requirements. During this process, also the resources have to be applied in the calculations. When the product requirements are selected, a release definition is written that is validated by different stakeholders. A business case is sent to the company board. When this has been approved by the board, a launch preparation package is constructed and sent to the stakeholders.

## 5. Case study

In finding confirmation for the validity of the identified context, activities and relations depicted in the reference framework, we analyzed the conception, development and launching of a new product at a major Enterprise Resource Planning (ERP) software vendor during the period September 2000 to June 2002. The responsible product manager at this company provided us with all incoming e-mail traffic regarding this new product as a source for our analysis. In the mentioned period the product manager received about 1,200 emails related to this product, which serve as the source for this case study.

## 5.1 ERP vendor case

After an organizational repositioning, the management board of the ERP vendor decided to focus on providing add-on products, so-called solutions, next to ERP products. So, from September 2000 onwards, an integrated procurement product was planned, including direct materials purchasing, indirect materials purchasing, e-procurement, e-invoicing and e-kanban (a Just-In-Time purchasing strategy solution), to be integrated via one Supplier Trading eXchange (STX). Note that at the start, some of the functionality was already available in existing products (e.g. direct materials purchasing in the ERP-product), while other functionality needed to be created. Existing and new functionality needed to be disclosed through STX.

As for portfolio management, a number of e-mails represented the assignment of solutions, including the STX solution. Although the board indicated (based on market signals) the necessity of solutions, the product manager verified the need for a specific procurement solution through industry analysts, important customers and competitor analysis. Specifically the successful implementation at Komatsu of a predecessor application of the STX, i.e. the E-Collaboration tool, encouraged the product manager to further prepare development of the STX. In one of the e-mails the product manager was invited by someone from the ERP vendor's consultancy department to attend a knowledge transfer on E-Collaboration based on the successful implementation at Komatsu: "I spoke with Komatsu today just to see how things are going and to ask permission to access their site tomorrow for a knowledge transfer session that I am doing for some of the consulting folks and Sales Managers; you are most welcome to call-in". Note that this particular implementation has been described in a case study in a separate paper [49].

A potential partner company was approached to further enhance functionality regarding the so-called 'round-trip' requisitioning (i.e. linking into suppliers' item catalogues at the suppliers' websites in order to purchase goods from suppliers' sites directly). Integration between the partner's product and STX would make this possible, as one of the e-mails states: "Supplier's product information is dynamically available through agent technology in the partner product's Java code".

Regarding product roadmapping, it became clear that not all topics and themes for an (according to the product manager) ideal procurement solution through the STX could be covered in one release. An example was the late discussion of e-kanban and its incorporation in the future: in one of the e-mails the product manager asked a colleague to "provide me with some compelling arguments why it is good to develop E-Kanban in STX

from the business perspective". In general, in many e-mails dealt with themes projection over future anticipated releases of the STX. This included communication with the management board of the company.

Many of the 1,200 e-mails dealt with requirements management and release definition. A number of detailed requirements became clear from the previous implementation at Komatsu. In addition, communication with the support and consultancy departments provided other requirements for the STX. At the end of 2000, an early version of a release definition was communicated with a number of internal departments, including marketing & sales, development, and the release management department. Later on, the architect of the development department interpreted the requirements in a functional design document: "Here is the first draft of functional design document" (e-mail of 9 March 2001). Subsequent e-mails from the development department mainly dealt with requirement clarification ("I need clarification about the off-line purchase in the STX") and scope changes ("shouldn't we support RosettaNet message exchange?").

In cooperation with other departments and associated country organizations the product manager prepared the launch of the STX: e.g. a white paper was written on the product with involvement of marketing and sales ("Sure thing! I'll make sure this is in the plan and we can work together to get it done".).

## 5.2. Case analysis

In the case study on STX we note that all main product management areas (portfolio management, product roadmapping, requirements management and release management) were addressed. Some areas and some topics within each of the areas were more subject in e-mails than others. For example, product lifecycle management in portfolio management was not so much addressed, as it concerned the first releases of STX, therefore roadmap construction was more extensively addressed. Also, requirements prioritization and selection was not addressed extensively, the scope of the STX, and the list of all requirements was rather small. However, proposed scope increases were weighed carefully in order to either include or exclude them: the product manager had to balance between allowing scope creep for development and satisfying sales & marketing.

All identified stakeholders in figure 2 were extensively involved in the communication with the product manager, even for research & innovation: the development department prototyped the round-trip functionality with the STX's partner product.

The largest category of all the 1200 e-mails came from development. This can be explained by the fact that

development took place in another country than the country of origin of the product manager. Much communication was through conference calls and e-mails.

## 6. The Software Product Management Workbench

Product management is key to product software companies and should be addressed and supported well. Although there are several tools supporting part of the product management functionality, they do not provide a coherent and complete set of features dedicated to software product management. Because there is a need for an integrated tool to support the product manager, we propose the software Product Management Workbench. At the same time, we use this workbench to validate our reference framework. We use the identified process areas to outline the architecture of the system.

### 6.1. Existing support tools

Several portfolio management support tools exist, e.g. ProSight's Application Portfolio Management, supporting top-down portfolio management solutions for a company, and UMT's Portfolio Manager Software Suite, a web-based application for portfolio management.

Few support tools for product roadmapping exists. ReleasePlanner [40] covers part of it. ReleasePlanner is a web-based system solution to enable intelligent planning, priority and road-mapping decisions.

Tools that focus especially on requirements management are Borland's CaliberRM for managing requirements throughout the software delivery process and IBM's RequisitePro, a requirements and use case management tool. ReqSimile [35] is a tool that supports the linkage process in large-scale requirements management, by using a linguistic engineering approach.

Some tools exist in the release planning area. The Accept 360° platform from Accept Software is a product planning and delivery solution that addresses the spectrum of business requirements in all levels of the organization. ReleasePlanner [40], earlier mentioned in this section, is a uses integer linear programming and prioritization of features for purposes of release planning. This tool focuses on (but is not limited to) software companies. In the Release Planner Prototype [11] a selection algorithm is implemented that presents a number of valid and good release suggestions.

### 6.2. An integrated solution

To provide operational support for software product management, we propose a tool: the Software Product Management Workbench. As explained further, it supports portfolio management, product roadmapping, release planning and requirements management, in an integrated way.

The workbench is divided into four main modules, all intended to aid the product manager with his daily routines. The four modules are: *requirements module*, *release planning module*, *roadmap module*, and *product portfolio module*, their names corresponding to the functionality they provide.

The workbench is designed for different user types. The *product manager* is the main user, but there are three other users that are able to login into the system, all with their own privileges. These three users are: *administrator*, *core asset developer*, and *employee*. Product software companies usually have multiple software products all furnished with new releases every once in a while. The main task of the administrator is to start new products or new product releases. When a new core asset has been identified, the core asset developer can login into the system and add this new core asset to the system. In this way the product manager can use the core asset in defining a release, and the development team has always access to information on the latest core assets. An employee can logon to the system for reading the latest news of the development progress or report some news about his work on an upcoming release.

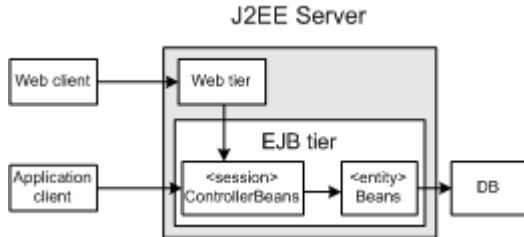
### 6.3. Architecture

The Software Product Management Workbench is a so-called *enterprise application*. Building enterprise applications is a hard and taunting task [23], because they deal with a lot of persistent data, concurrent data access, multiple users with different roles, and are built in a distributed way. In the workbench the difficulties are found in the great amounts of requirements that have to be persistent, different actors that can login into the system, and more. J2EE is a platform that enables the easy creation of enterprise applications, since J2EE handles all the difficult tasks described above for you. This means that enterprise programmers only have to deal with programming the business logic. For technical information of J2EE see [6]. In [43], Szyperski provides a thorough evaluation of the J2EE platform.

Figure 3 gives a high level overview of the architecture. The tool uses two types of clients: a *web client* and an *application client*. Application clients run on the client machine and offer the ability to perform heavy calculations on the client machine, without affecting the server. Enterprise Java Beans (EJB) form the core of the J2EE platform that makes the life of an



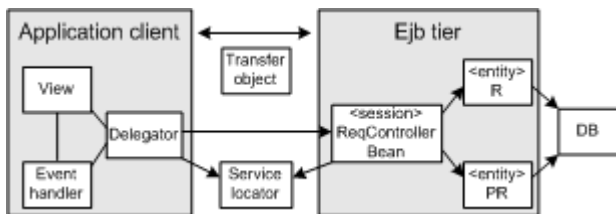
enterprise programmer easier. Two types of EJBs are used in the architecture, namely *entity* and *session* beans. One entity bean represents one row in a database table (or a row in the result of a join operation). Two types of *session* beans exist, which are *stateful* and *stateless* session beans. A stateful session bean can maintain conversational state for one client. A stateless session bean offers its services to multiple clients.



**Figure 3. High level architecture**

The response time, the amount of time it takes for the system to process a request from the outside, is of great importance [23]. The product manager uses functionalities of the tool that require a lot of processing time, so he is the only one able to login into the application client to execute these calculations. The web tier handles all the requests generated by the web client and directs these requests to the controller beans that are deployed into the EJB tier, which provide coarse grained access to the entity beans. The application client accesses the controller beans directly. Note that the web tier and the EJB tier do not have to reside on the same machine.

The extensibility of the tool is also an important issue. As mentioned before there are now four main modules, but the tool should be able to be extended with minimum effort to provide other kinds of functionality. Figure 4 shows a small part of the full architecture, but captures some of the patterns used.



**Figure 4. Requirements administrator module**

The figure shows part of the requirements module, where incoming requirements are connected to product requirements. Remote calls and calls from the web tier to the EJB tier are relatively very slow, so this number should be minimized. The system uses transfer objects that capture as much data that the client possibly wants to get his hands on, instead of getting one piece of data at

the time at the cost of one remote call every time. The different components of the system have to be located with so called “look-ups”. It is efficient to extract this code from all the components and put all the look-up code in a service locator object. In this way, references to components can be cached for other components that may need a reference to that component, minimizing the look-ups. There is no tight coupling between the components, which makes the tool easy to change. If for example the presentation logic has to be adapted, only the view has to be changed leaving the other components unharmed.

### 6.4. Prototype

Figure 5 shows a screenshot of the prototype of the Software Product Management Workbench. It shows the requirements window, in which the product manager can link requirements with product requirements that refer to the same functionality [35]. At the top of the screen, a list of product requirements is depicted. A product requirement can be selected in order to find matching requirements from the requirements list at the bottom of the screen. After the system has found all the possible candidates, the requirements are displayed together with the source, similarity ratio and the option to link this requirement to a product requirement. When the preferred requirements are selected, the linkage can be saved.

### 7. Conclusions and further research

In this article we discussed the difference between product management and software product management, and the need for operational support for the latter. By performing field interviews and discussions with product managers and by doing a literature review on (software) product management, we developed a reference framework for software product management. Furthermore, we provided an overview of state-of-the-art literature on software product management. By carrying out a case study, we found confirmation of the validity of the identified context, processes and relations in the reference framework for software product management. Finally, we proposed the Software Product Management Workbench, which integrates several software product management areas. This workbench is currently being developed. When it is finished, several industrial case studies will be performed to test the functionality.

We are convinced that the software product management reference framework is a first step to position this important industrial domain in the field of scientific research on software product management. In the future, we hope to contribute to further refinements of the reference framework and to its application in various domains.

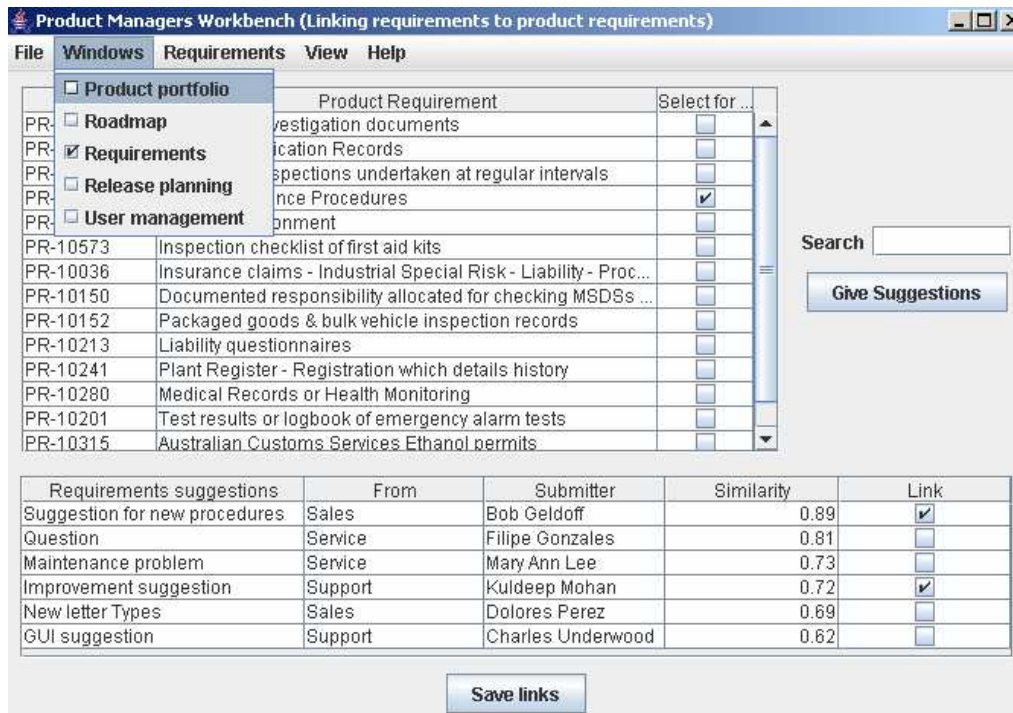


Figure 5. Screenshot of the Software Product Management Workbench

## References

- [1] M. Abramovici and O.C. Soeg, *Status and Development Trends of Product Lifecycle Management Systems*, Ruhr-University Bochum, Chair of IT in Mechanical Engineering (ITM), Germany, 2002.
- [2] M. van den Akker, S. Brinkkemper, G. van Diepen, and J. Versendaal, "Flexible Release Planning Using Integer Linear Programming", *Proceedings of the 11th International Workshop on Requirements Engineering for Software Quality (REFSQ'05)*, 13-14 June 2005, Porto, Portugal, Essener Informatik Beitrage, Band 10.
- [3] D. Alur, J. Crupi, and D. Malks, *Core J2EE Patterns*, Prentice Hall PTR, 2001.
- [4] M. Ardis, N. Daley, D.M. Hoffman, H. Siy, and D. Weiss, "Software Product Lines: a Case Study", *Software - Practice and Experience*, 2000, vol. 30, no. 7, John Wiley & Sons, Ltd, New York, pp. 825-847.
- [5] P. Berander and A. Andrews, "Requirements Prioritization", *Engineering and Managing Software Requirements*, A. Aurum and C. Wohlin (eds.), Springer Verlag, Berlin, Germany, 2005, pp. 69-94.
- [6] S. Bodoff, D. Green, K. Haase, E. Jendrock, M. Pawlan, and B. Stearns, *The J2EE Tutorial*, Addison Wesley Professional, Boston, MA, 2002.
- [7] A. Bonaccorsi and A. Lipparini, "Strategic Partnerships in New Product Development: An Italian Case Study", *Journal of Production Innovation Management* vol. 11, no. 2, 1994, pp. 134-145.
- [8] J. Bosch, "Product-Line Architectures in Industry: A Case Study," *ICSE Proceedings*, Los Angeles, CA, 1999, pp. 544-554.
- [9] L. Brownsword, P. Clements, *A Case Study in Successful Product Line Development*, Technical Report CMU/SEI-96-TR-016, Carnegie Mellon, 1996.
- [10] P. Bourque and R. Dupuis, (ed.), *Guide to the Software Engineering Body of Knowledge*, 2004 edition, IEEE Computer Society, Los Alamitos, California, USA, 2004.
- [11] P. Carlshamare, "Release Planning in Market-Driven Software Product Development Provoking and Understanding", *Requirements Engineering*, vol. 7, no. 3, Springer, London, 2002, pp. 139-151.
- [12] P. Carlshamre, B. Regnell, "Requirements Lifecycle Management and Release Planning in Market-Driven Requirements Engineering Processes", *11th International Workshop on Database and Expert Systems Applications (DEXA'00)*, 2000, p. 961.
- [13] E. Carmel, *Global Software Teams*, Prentice Hall: Upper Saddle River, NK, 1999.
- [14] P. Clements and L. Northrop, *Software Product Lines: Patterns and Practice*. Reading, MA: Addison Wesley, Boston, MA, 2001.
- [15] D. Condon, *Software Product Management*, Aspatore Books, Boston, MA, 2002.
- [16] R.G. Cooper, S.J. Edgett, and E.J. Kleinschmidt, "Portfolio Management for New Product Development: Results of an Industry Practices Study", *R&D Management*, vol. 31, no. 4, 2001, pp. 361-380.
- [17] M.A. Cusomano, *The Business of Software*, Free Press: New York, 2004

- [18] M.A. Cusumano and R.W. Selby, *Microsoft Secrets*, Simon and Schuster, New York, 1995.
- [19] A.S. Dver, *Software Product Management Essentials*, Anclote Press, 2003.
- [20] C. Ebert, "Understanding the Product Lifecycle: Four Key Requirements Engineering Techniques", *IEEE Software*, vol. 23, no. 3, 2006, pp. 19-25.
- [21] C. Ebert and J. De Man, "e-R&D – Effectively Managing Process Diversity", *Annals of Software Engineering*, vol. 14, no. 1, 2002, pp. 73 – 91.
- [22] C. Ebert and M. Smouts, "Tricks and Traps of Initiating a Product Line Concept in Existing Products" *Proceedings of the 25th International Conference on Software Engineering (ICSE '03)*, IEEE Comp. Soc., Portland, Oregon, USA, pp. 520-525.
- [23] M. Fowler, *Patterns of Enterprise Application Architecture*. Addison-Wesley, Boston, MA, USA, 2003.
- [24] R.L. Glass, I. Vessey, and V. Ramesh, "Research in Software Engineering: an Analysis of the Literature", *Information and Software Technology*, no. 44 2002, pp. 491-506.
- [25] A. van der Hoek, "Software release management", *Proceedings of the Sixth European Software Engineering Conference together with the Fifth ACM SIGSOFT Symposium on the Foundations of Software Engineering*. Springer: Heidelberg, Germany, 1997, pp. 159-175.
- [26] Information Technology - Open Systems Interconnection - *Basic Reference Model: The Basic Model*. International Standard, ISO/IEC 7498-1. 2nd ed. Geneva: ISO, 1994.
- [27] T. Kappel, "Perspectives on Roadmaps: How Organisations Talk about the Future", *IEEE Engineering Management Review*, vol. 29, no. 3, 2001, pp. 36-48.
- [28] J. Karlsson and K. Ryan, "A Cost-Value Approach for Prioritizing Requirements", *IEEE Software*, vol. 14, no. 5, 1997, pp. 67-74.
- [29] T. Kilpi, "Product Management Challenge to Software Change Process: Preliminary Results from Three SMEs Experiments", *Software Process - Improvement and Practice*, vol. 3, no. 3, 1997, pp. 165-175.
- [30] R. Laqua, "Concepts for a Product Line Knowledge Base & Variability", *Proceedings of NetObjectDays 2002*, October, 2002, pp. 30-39.
- [31] S. Lauesen, "COTS Tenders and Integration Requirements", *12th IEEE International Requirements Engineering Conference (RE'04)*, 2004, pp. 166-175.
- [32] L. Lehtola, M. Kauppinen, M., and S. Kujala, "Linking the Business View to Requirements Engineering: Long-Term Product Planning by Roadmapping", *Proceedings of the 13th IEEE international Conference on Requirements Engineering (Re'05)*. IEEE Computer Society, 2005, pp. 439-446.
- [33] J. de Man and C. Ebert, "A Common Product Life Cycle in Global Software Development", *Eleventh Annual International Workshop on Software Technology and Engineering Practice*, 2003, pp. 16-21
- [34] M. Moon and K. Yeom, "An Approach to Develop Requirement as a Core Asset in Product Line", *Lecture Notes in Computer Science*, no. 3107, 2004, pp. 23 – 34.
- [35] J. Natt och Dag, V. Gervasi, S. Brinkkemper, and B. Regnell, "A Linguistic-Engineering Approach to Large-Scale Requirements Management" *IEEE Software*, vol. 22, no. 1, 2005, pp. 32-39.
- [36] B. Nuseibeh and S. Easterbrook, "Requirements Engineering: A Roadmap", *The Future of Software Engineering*, A. Finkelstein (ed.), ACM Press: New York, 2000, pp. 37-46.
- [37] *PMBOK, A Guide to the Project Management Body of Knowledge*, first ed., Project Management Institute, Pennsylvania USA, 2000.
- [38] C. Potts, "Invented Requirements and Imagined Customers: Requirements Engineering for Off-the-Shelf Software," *Second IEEE International Symposium on Requirements Engineering (RE'95)*, 1995, p. 128.
- [39] B. Regnell and S. Brinkkemper, "Market-Driven Requirements Engineering for Software Products", *Engineering and Managing Software Requirements*, A. Aurum and C. Wohlin (eds.), Berlin, Germany, Springer Verlag, 2005, pp 287-308.
- [40] G. Ruhe and M.O. Saliu, "The Art and Science of Software Release Planning", *IEEE Software*, vol. 22, no. 6, 2005, pp. 47-53.
- [41] T.L. Saaty, *The Analytic Hierarchy Process*, McGraw-Hill, New York, NY, 1980.
- [42] M.O. Saliu and G. Ruhe, "Supporting Software Release Planning Decisions for Evolving Systems," *29th Annual IEEE/NASA Software Engineering Workshop*, 2005, pp. 14-26.
- [43] C. Szyperski, *Component Software: Beyond Object-Oriented Programming*, 2nd ed., ACM Press and Addison-Wesley, 2002
- [44] S. Thiel, S. Ferber, T. Fischer, A. Hein, and M. Schlick,, "A Case study in Applying a Product Line Approach for Car Periphery Supervision Systems", *Proceedings of In-Vehicle Software, SAE 2001 World Congress SP-1587*, Detroit, Michigan, USA, 2001, pp. 43-55.
- [45] D. Tschritzis and A. Klug, "The ANSI/X3/SPARC DBMS Framework Report of the Study Group on Database Management Systems", *Information Systems*, vol. 1, 1978, pp. 173-191.
- [46] S. Unger, "Ten Marketing Challenges that Can Make or Break your Business... and How to Address Them", *Productmarketing.com*, vol. 1, no. 1, 2003.
- [47] J. Vähäniitty, C. Lassenius, and K. Rautiainen, "An Approach to Product Roadmapping in Small Software Product Businesses", *Quality Connection - 7th European Conference on Software Quality (ECSQ2002), Conference Notes*, Center for Excellence Finland, Helsinki, Finland, 2002, pp. 12-13
- [48] J. Vahaniitty and K. Rautiainen, "Towards an Approach for Managing the Development Portfolio in Small Product-Oriented Software Companies", *Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS'05)*, 2005, p. 314.
- [49] J.M. Versendaal and S. Brinkkemper, "Benefits and Success Factors of Buyer-Owned Electronic Trading Exchanges: Procurement at Komatsu America Corporation", *Journal of Information Technology Cases and Applications*, vol. 5, no. 4, 2003, pp. 39-52.

# Can Agility be Introduced into Requirements Engineering for COTS Component Based Development?

Kendra M. L. Cooper  
The University of Texas at Dallas  
kcooper@utdallas.edu

## Abstract

*The Rational Unified Process is a popular software development process framework that can be tailored to meet the needs of different kinds of projects (large, small, component development, component based application development, etc.). This position paper presents an overview of a comprehensive RUP based process for developing COTS based systems and poses a set of questions that need to be considered to introduce agility into the process from a requirements engineering perspective.*

## 1. Introduction

The need to rapidly produce quality software and respond to changes in a flexible and quick manner has driven the definition of new techniques including Agile Methods (AM) and Component Based Software Engineering (CBSE) approaches, which use a variety of software component products (e.g., off-the-shelf (OTS), commercial OTS (COTS), etc.). These approaches share many common goals: increasing the productivity of the development teams, reducing products' time-to-market, reducing the development costs, and improving customer satisfaction.

AMs are a set of software processes that share the same core values defined in Agile Manifesto [1]: individuals and interactions over processes and tools; working software over comprehensive documentation; customer collaboration over contract negotiation; responding to changes over following a plan. Numerous AMs have been proposed including Scrum [11], Extreme Programming [2], and DSDM [6]. AMs are not prescriptive processes: they don't define detailed procedures for how to create a given type of model; instead they provide advice, or guidance.

The requirements in AMs are recognized as key elements; this can be seen from the highest priority principle in agile development: satisfy the customer. The requirements engineering (RE) activities and

artifacts are defined in some form by the AMs. For example, XP describes the RE activities in terms of User Stories. A User Story is a brief summary of a discussion with the customer about the requirements that is hand-written on an index card; they capture high-level functional and non-functional requirements and are prioritized. Simplified, point form versions of use cases have been proposed as an agile alternative to a fully defined use case model [2]. The point form use cases do not necessarily contain an overview, pre-conditions, post-conditions, and so on. The flows of events can be presented very simply as bullets.

CBSE methods are a set of software processes that support the identification, evaluation, acquisition, integration, testing, and/or maintenance of systems that are developed using components (i.e., reusable software products), rather than from scratch. The focus in this work is on COTS CBSE (CBSE<sub>C</sub>) techniques, in which the products are blackboxes (source code is not available). The use of OTS products needs to be considered as a separate, more general problem, as the products are whiteboxes (source code is available). Proposals that emphasize the requirements engineering activities in CBSE<sub>C</sub> include Scarlet [9], COTS Aware Requirements Engineering and Software Architecting [5], and the Evolutionary Process for Integrating COTS Based Systems (EPIC) [3].

Introducing agility into CBSE<sub>C</sub> techniques has received little attention. An analysis of selected CBSE<sub>C</sub> techniques with respect to agile principles is reported in [10]. Points of consistency and conflict are identified. For example, the authors' analysis on the agile principle to "welcome changing requirements" is viewed as being consistent with COTS based software development. Other principles, however, have a mix of consistent and conflicting points. For example, the principle that states the "business people and developers need to work together" brings up practical issues of having the vendors working on site, protecting intellectual property, etc. In AMs, the business people include stakeholders who understand

the requirements for the application under development.

The position presented in this work is that it is important to further investigate how to introduce agility into a CBSE<sub>c</sub>. Here, the first step is taken by identifying some of the research questions and issues involved. A specific CBSE<sub>c</sub> approach, EPIC, has been selected for this phase of the work to provide a concrete process definition to work with; additional approaches need to be considered. EPIC has been captured as a tailored version of the established Rational Unified Process (RUP).

The remainder of this paper is structured as follows. The RUP and a tailored version of the RUP for CBSE<sub>c</sub>, the IBM RUP COTS plug-in, are overviewed in Section 2. Open research questions on defining an agile component oriented RE process are considered in Section 3. Conclusions are in Section 4.

## 2. Tailoring the Rational Unified Process

### 2.1 Rational Unified Process

The Rational Unified Process (RUP) is a popular, adaptable process framework that describes how to effectively develop software using proven techniques [8]. RUP is organized along two dimensions. The horizontal dimension represents the dynamic structure; it is described in terms of cycles, phases, iterations, and milestones. The vertical dimension represents the static structure. Here, disciplines (or workflows) such as business modeling, requirements, etc. are described in terms of the roles, activities, artifacts, workflows, and additional process elements (templates, guidelines, concepts, roadmaps, and help on tools). RUP defines activities (how to do the work), roles (who is doing the work), workflows (when the work is done), and the artifacts (what is delivered).

The RE discipline begins in the inception phase, in which the architecturally significant use cases are identified and outlined by the analyst. The work continues with substantial effort in the elaboration phase, in which the use case model is completed to 80%, then tapers off in the construction and transition phases. The requirements are managed throughout.

The RUP is intended to be tailored, to better meet the needs of a specific project or organization. RUP is included into the IBM Rational Method Composer (RMC) product, which allows customizing the process and publishing the resulting process through the internet [7]. A tailored version is called a plug-in. For example, a plug-in for agile projects with the Agile Unified Process has been proposed [1]; numerous

plug-ins have been published by IBM, including the COTS component based development plug-in.

### 2.2 EPIC and the IBM RUP COTS Plug-in

EPIC is a comprehensive, tailored version of the RUP that meets the challenges of building, fielding, and supporting COTS component-based business solutions [3]. It simultaneously defines and makes trade-offs among four *spheres of influence*: the stakeholders' needs and their business processes (i.e., more traditional RE, analysis), the components currently available in the marketplace, the architecture and design of the system, and programmatics and risks. The trade-offs are considered in each iteration. As iterations progress and decisions are negotiated, the solution space becomes smaller, knowledge of the solution accumulates, and stakeholder buy-in increases. As a result, the components available in the marketplace are re-examined in each iteration to determine if they are suitable (refer to Figure 1).

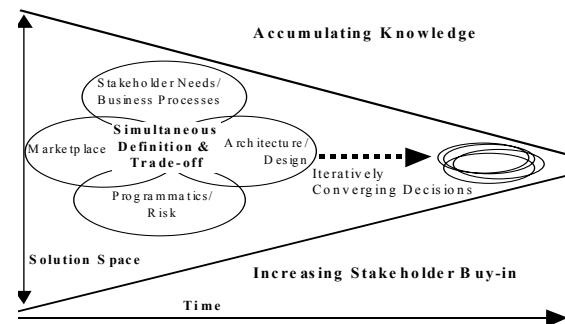


Figure 1. Overview of the EPIC Framework [3].

The milestones, workflows, activities, artifacts, terms, and descriptions from the RUP are re-used and/or modified as needed; new process elements are added. For example, new artifacts to characterize the component marketplace include the market segment information, component dossier (for each component that is considered), and the component screening criteria and rationale. For each new artifact, EPIC provides a guideline that describes the purpose of the information gathered, key questions to answer, the information needed, and techniques to use.

EPIC has been used as the basis for the IBM RUP for COTS Package Delivery plug-in. For example, templates for the COTS inception, elaboration, construction, and transition iterations are defined. Under each of these, the activities are refined into sub-activities and ultimately tasks. Each task has properties that can be instantiated, including the guidelines. The guidelines are left empty in the plug-in to provide

flexibility. The guidelines are targeted in this work as a way to introduce agility into the process.

### 3. Considering Agility and COTS in RE

The Agile Manifesto and the 12 principles for AMs [2] provide the basis for introducing agility into the IBM RUP COTS plug-in. Many of the principles are highly related to the RE discipline (e.g., satisfy the customer, adapt to changing requirements, deliver working software frequently, business people and developers work together). Of these principles, one in particular stands out as a significant technical challenge in introducing agility into RE for component based systems: *deliver working software frequently*.

AMs propose to deliver working software in short iterations (from a couple of weeks to a couple of months). From a RE perspective, these frequent deliveries are excellent opportunities to validate the requirements realized in the software with the customer. When the delivery is demonstrated, the customer has an opportunity to identify what they do and don't like; problems can be quickly addressed. However, many questions arise when considering the short iterations from a component perspective:

1. Does a short iteration give enough time to wisely identify, evaluate, select, and integrate COTS components to meet the customer's requirements?
  - a. Can a component dossier be simplified and still be useful or replaced with tacit knowledge?
2. How does the relationship with the vendor need to change to make COTS selection agile?
  - a. Can the evaluation workshop (i.e., meeting to evaluate the components) with the vendor be streamlined?
  - b. When, or for how long, is it appropriate to have the vendors working on-site?
  - c. Can the process to address change requests to a COTS component product be more agile?
3. What and how much information should be collected for the four spheres of influence, particularly in the inception and elaboration phases, when the RE effort is significant?
4. Are only some kinds of components suitable in an agile approach? For example, should only the following kinds of components be considered:
  - a. The components meet an international standard (IEEE, ISO, ANSI, etc.); their capabilities are well documented and understood
  - b. The components are relatively small and/or simple to understand and evaluate with respect to the customer's requirements. If the component is small and/or simple, then is it worth buying or should the requirements be realized in house?

- c. The developers have used the component before to meet similar requirements, reducing the effort needed to understand and use it.

5. Are all of the requirements potential candidates for realizing with COTS? AMs focus on realizing the requirements for the current iteration. Is there a way to characterize requirements that are better suited for realization with COTS, for example, those that are less likely to change in future iterations?

### 4. Conclusions and Future Work

There are numerous questions to investigate just considering the single agile principle to "deliver working software frequently" in the definition of an agile RE process that addresses the management of the component based system under development and the reusable software products. The other agile principle highly related to RE also need to be investigated.

### 5. References

- [1] Agile Manifesto, available at <http://www.agilemanifesto.org/>
- [2] Ambler, S., *Agile Modeling: Effective Practices for Extreme Programming and the Unified Process*, Wiley, March, 2002.
- [3] Albert, C. and Brownsword, L., *Evolutionary Process for Integrating COTS Based Systems (EPIC) Building, Fielding, and Supporting Commercial-off-the-Shelf (COTS) Based Solutions*, technical report CMU/SEI-2002-TR-005, November 2002.
- [4] Beck, K., *Extreme Programming Explained: Embrace Change*, Addison-Wesley, 1999.
- [5] Cooper, K., Ramapur, C., and Chung, L., *A COTS-Aware Requirements Engineering and Architecting Approach: Defining System Level Agents, Goals, Requirements and Architecture (Version 4)*, UTDCS-24-05, The Univ. of Texas at Dallas, December 2005.
- [6] DSDM Consortium, [www.dsdm.org](http://www.dsdm.org)
- [7] IBM Rational Method Composer, [www-128.ibm.com/developerworks/rational/library/nov05/kro11/index.html](http://www-128.ibm.com/developerworks/rational/library/nov05/kro11/index.html)
- [8] Kruchten, P., *Rational Unified Process, The: An Introduction*, 3<sup>rd</sup> Edition, Addison-Wesley, 2004
- [9] Maiden, N., Kim, H., Ncube, C. "Rethinking Process Guidance for Selecting Software Components", in Proc. of 1st ICCBSS, LNCS 2255, 2002.
- [10] Navarrete, F., Botella, P., and Franch, X., "How Agile COTS Selection Methods are (and can be)?", in Proc. of the 31st EUROMICRO Conf. on Software Engineering and Advanced Applications, Porto, Portugal, Aug. 30th – Sept. 3rd, 2005, pp. 160-167.
- [11] Schwaber, K. and Beedle, M., *Agile Software Development with Scrum*, Prentice-Hall, 2001.

# Challenges of Knowledge and Collaboration in Roadmapping

Sami Jantunen

Lappeenranta University of Technology  
sami.jantunen@lut.fi

Kari Smolander

Lappeenranta University of Technology  
kari.smolander@lut.fi

## Abstract

*Today's software organizations need to cope with ever intensifying technical and commercial turbulence. In such environment, the level of market orientation could be the deciding factor determining the future success of a company. The activities bridging market orientation to software product development terminology have commonly been linked to terms such as roadmapping and release planning. The purpose of this paper is to explore the roadmapping practices currently followed in software product development organizations. Due to the lack of established theories of roadmapping practices, exploratory and inductive research methods were used. The study resulted with a conceptualized view of roadmapping activities. We identified three types of participants that need to be present in a roadmapping context. Each of the identified participant types relates to particular set of challenges regarding knowledge and collaboration in a roadmapping context. When comparing our results to the existing theories, we found out that the knowledge-based theory of the firm could contribute to further theory development in market-oriented software development and related areas.*

## 1. Introduction

Today's software organizations need to cope with ever intensifying turbulence of technical and commercial environments. These challenges are often further complicated with the global competition. In such operating environment one would assume that understanding the demand originating from the market would be a source of competitive advantage. In other words, the level of *market orientation* could be the deciding factor determining the future success of a company. Yet, according to Boehm [1], much of current software engineering practice and research is done in a value-neutral setting, in which every requirement, use case, object, and defect is treated as equally important. The current practice and research mostly sees software

engineers as responsible for turning software requirements into verified code – without giving a greater thought about business or market priorities and values.

Why there appears to be a gap between the recognized need of market oriented product development activities and the actual practices currently followed in software organizations? What are the obstacles of introducing market orientation to software organizations? Such largely unanswered questions motivated us to gain a deeper understanding of market oriented product development in practice. In our learning process, we wanted to pay our particular attention to the activities where the future direction of a product is under discussion, namely to the processes of roadmapping and release planning. We believed that the results of such study would be important because, as expressed in [2], “a distinct and workable conceptualization of market oriented product development is the first step towards understanding the implementation process”.

The topic of market orientation in general has been widely researched from various perspectives resulting with several definitions for it. In their work on finding synthesis from the most cited market orientation definitions, Jaworski and Kohli [3] proposed their own definition as: “the organization wide *generation* of market intelligence pertaining to customers, competitors, and forces affecting them, internal *dissemination* of the intelligence, and reactive as well as proactive *responsiveness* to the intelligence”.

The relationship between market orientation and software product development has still been uncovered to a large extent. The activities and artifacts bridging market orientation to software product development terminology have commonly been linked to terms such as *roadmapping*, *release planning* and *a roadmap*. For the sake of clarity we will provide the definitions used in this paper for these terms. The definition for *a roadmap* has been adopted from [4] as: “A roadmap describes a future environment, objectives to be achieved within that environment, and plans for how those objectives will be achieved over time”. A roadmap usually includes a description of how the pieces of technology fit together and how they evolve in the future. The definition for

*roadmapping* has also been adopted from [4]: “The roadmapping process helps a team gather diverse perspectives on all aspects of the environment and the plan. It also helps the team build consensus and gets buy-in of its members to carry out the plan”. For term *release planning* we use the following definition from [5]: “The release planning is the activity of determining a feasible combination of dates, features, and resourcing for the *next* release of a software product”. Based on these definitions we consider release planning as a subset of roadmapping activities. Therefore, the following findings from existing literature regarding release planning can be considered relevant also to the activity of roadmapping.

Several studies (e.g. [6, 7]) have identified the necessity of listening the success critical stakeholders and understanding the value of their needs when planning the future releases of a product. However, despite the importance of these objectives, the current practice of release planning has been reported to be problematic with such identified issues as lack of systematic release planning practices, lack of resource considerations and insufficient stakeholder involvement [8]. An important factor that may potentially result as poor release planning practices could be the complex nature of the release planning phenomenon. Supporting evidence for such proposition can be found from [6] with a conclusion that “The initial attempt at supporting release planning proved to be based on overly simplistic and structuralistic view”.

Requirements Engineering (RE) has an essential role in accomplishing the goals of release planning task. RE is a multi-disciplinary activity that is concerned with the identification of the goals of stakeholders and their elaboration into precise statements of desired services [9]. Several studies (e.g. [10-12]) have pointed out that market-driven software organizations face unique challenges in their RE activities compared to the customer-specific software organizations. One fundamental difference is due to the large customer base of a product making it difficult to take into account a large number of stakeholders that have varying and often conflicting needs. The global presence of a product exacerbates the RE related challenges even further. It has been claimed [13] that the prioritization and negotiation of customer requirements for a particular release is the most significant challenge of a global organization. Similar findings have also been presented by Lehtola *et.al.* [10] with a statement that “prioritization methods may have limited ability to support decision-making in a complex area like requirements prioritization in market-driven product development”. Therefore, they argue further that prioritization results should be taken more as being indicative than as an ultimate truth.

Relevant to the challenges of roadmapping and release planning, a new discipline of Value-Based Software Engineering (VBSE) has been emerging in recent years

aiming at “integrating value considerations into all of the existing and emerging software engineering principles and practices, and of developing an overall framework in which they compatibly reinforce each other” [1]. Some of the elements in the research agenda of VBSE addressing roadmapping and release planning challenges includes value based requirements engineering for identifying success-critical stakeholders and their objectives, and value-based design and development for ensuring that the objectives and values are inherited by the design and development [1]. More recent advances in the discipline of VBSE can be found from [14], in which the challenge of human involvement in Software Engineering has been acknowledged to result in less formal, timeless and universal theory in terms of situations, stakeholders, and products. Similar findings regarding the challenge of human involvement has also been identified in the context of release planning. Ruhe and Saliu [7] distinguish between the *art of release planning* addressing the need for human intuition, communication, and capabilities to negotiate between conflicting objectives and constraints and the *science of release planning* formalizing the problem and applying computational algorithms to generate best solutions.

Given these findings from existing literature, we have decided to make a deep dive into roadmapping activities in practice and focus our attention to human knowledge and collaboration in the roadmapping context. We believe that this will provide important insights for the further development of roadmapping practices. In particular, we wish to answer the following questions in our study:

- What types of stakeholders should participate in a roadmapping context? Why?
- What types of information is gathered in practice from the product’s stakeholders?
- What types of information has proved to be of most significant value?
- What challenges companies have faced on facilitating roadmapping activities? What could be underlying reasons for the identified problems?
- What challenges companies have faced on disseminating roadmapping knowledge into the organization?

The remaining part of this document is organized as follows. Section 2 describes the research process and methods used in this study. Section 3 presents our findings and observations regarding roadmapping and release planning in practice. In section 4, the research results are reflected to the existing literature and finally section 5 presents the conclusions of the study.

## 2. Research Process

This study explores roadmapping and release planning practices currently followed in software product



development organizations. The study area has been fairly unexplored up to this point and lack previous theories. Therefore, a qualitative and theory-forming strategy is a necessity. The grounded theory method [15, 16] has proved its suitability for theory-forming research in the disciplines of software engineering and information systems development [17-19]. Because of this fact, we chose grounded theory as the research method in this study.

The study used theme-based interviews as its main data collection method. In total, we used 27 interviews from 7 organizations as the data material for this study. The organizations represented different areas of software industry, such as engineering applications, content management, database management, mobile applications and telecom applications. Common to all the organizations were that a major part of their business was based on software products. The interviewees represented various functions of their organizations, including product management, marketing management, product development, and general management. All interviews were tape-recorded and transcribed to text. The total amount of recordings adds up to 38 hours and the number of transcript pages is in total 554 pages. In addition to the interviews, data was also collected from company and product presentations that were held to us and we also received a good number of additional material, including process, product, and company descriptions and marketing material.

The data analysis is still at the time of writing an ongoing process. The analysis started with open coding [16], where essential sections of the data were conceptualized and identified as categories. The categories represent regularities or irregularities or any phenomenon that was considered important in relation to the research question. Currently we have continued to axial coding [16], where the relationships between the categories are in focus. This paper presents some of the results of this phase that are related to roadmapping and release planning. The study is currently continuing to selective coding, where a coherent picture or theory of the core category, market-driven software development, will be formed.

### **3. Challenges of Knowledge and Collaboration in Roadmapping**

Why is it hard to be systematic in roadmapping practices? Why stakeholders are not listened to the sufficient extent when making decisions related to future versions of a product? In order to be able to answer such questions, we searched the data for mentions regarding the phenomenon of roadmapping.

Since existing studies had recognized the human knowledge as an important element of roadmapping, we

decided to focus our study to the challenges of human knowledge and collaboration in a roadmapping context. In our view, this was necessary in order to be able to alleviate the obstacles of successful roadmapping practices. In our analysis, we identified the following types of participants that need to be present in a roadmapping context:

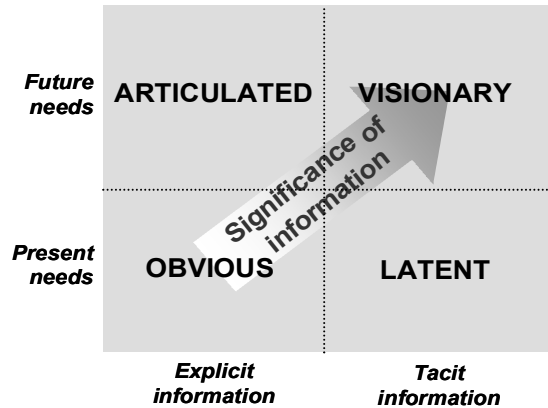
- *Contributor*, who bring valuable information to a roadmapping context,
- *Controller*, who ensures that roadmapping is being done in a systematic manner,
- *Distributor*, who absorbs information at a roadmapping context and disseminates it to those who will need to act upon it.

The remaining part of this section will elaborate in detail the identified challenges of knowledge and collaboration regarding each of the identified participant types.

#### **3.1. Helping to see the unseen: Contributor**

In order to be able to comprehend the gathered information from the market and transform it to a usable form of knowledge, it is essential that there are *contributors* in a roadmapping context. That is, there must be roadmapping participants that possess such mental models that enable successful outcome of a roadmapping activity. The challenge for the organizations is to understand which mental models should be applied in order to build economically sound understanding regarding product's future. Who should be the ones selected to act as contributors in a roadmapping context? What types of knowledge are most valuable for the activity of roadmapping?

We investigated the practices currently followed in the industry regarding how the future oriented understanding of a product is formed. We searched the data so as to find out in what ways the companies are currently gathering information about the market, how the gathered information is used and what challenges companies have faced when gathering the information. We considered this important in order to build a deeper understanding of the types of market information and their potential contribution to the roadmapping activity. A more detailed description of market-driven practices in one of the companies can be found from [20]. In our analysis, we identified four categories of market knowledge that were differentiated according to their time orientation (present-future) and level of explicitness (explicit-tacit). The identified categories of market knowledge (Figure 1) included:



**Figure 1. The categories of market information.**

- *Obvious* knowledge, that helps to explicitly describe stakeholder’s current needs regarding the product,
- *Articulated* knowledge, that helps to explicitly describe stakeholder’s future needs regarding the product,
- *Latent* knowledge that helps to constitute understanding about stakeholder’s current needs that are beyond their own recognition.
- *Visionary* knowledge that helps to anticipate stakeholder’s latent needs they are going to have in future.

But what kind of contribution the different types of market knowledge may offer in a roadmapping context? We believed that the answer to the posed question can be found by investigating the current challenges regarding the roadmapping practices. In our analysis, a particular pattern across the interviews was clearly visible. The companies were not in general facing challenges of collecting the explicit needs from the stakeholders. Instead, the problem appeared to be gaining understanding about the potential values behind the stated needs of the market. Without such understanding it was difficult for the companies to optimize the use of their resources while attempting to maximize the value of future work. This challenge can be expressed with the following quotation:

“We receive large amounts of market information, but the typical problem we are facing with it is that the business implication behind the customer need is often missing. In such case, we have difficulties on prioritization. We might not be able to see that focusing on other request would actually benefit us much more. We have a horn of plenty on receiving market information, but understanding the priority of information often gets lost in the abundance of technical details.”

This finding indicates that tacit knowledge is valued over explicit knowledge. The finding can be further strengthened with the following quotation:

“There is no equation that can determine the priorities of market needs correctly. It takes certain touch, hunch and experience to understand the priorities. This knowledge has just been built into the organization. [...] The more we have made business, the more we have gained this tacit knowledge.”

Furthermore, a repeated theme across the interviews was the necessity of being able to satisfy the unspoken needs of a customer. While this further supports the necessity of having tacit knowledge available in a roadmapping context, it also gives insight about why future-oriented knowledge may be valued over knowledge about present needs:

“We have approximately one year release cycles. If we want to take into account a feature request from a customer, we can consider it to be included to the release that is not currently under implementation. In the worst case, this means that the request will be implemented only after 24 months from the time of request and in such case the customer have definitely searched other options by then. In order to avoid such situations, we need to be able to anticipate customer’s latent needs in advance.”

We interpret these findings so that, for the success of roadmapping, the future-oriented tacit knowledge regarding the market will be of most significant value. As a result, the roadmapping *contributors* should possess primarily knowledge of *visionary* type.

### 3.2. Keeping it all intact: Controller

In order to produce a successful outcome from roadmapping activities, the activity needs to be managed. In other words, there need to be *controller* participants in a roadmapping context. Their duty is to introduce and facilitate systematic processes that guide the roadmapping participants to produce desired outcome. How the companies we interviewed have accomplished this in practice? What challenges have they faced? The interviews revealed that facilitating the roadmapping activities is a complex task. A commonly found issue across all interviewed companies was that the companies were facing challenges in establishing systematic practices into a roadmapping context. In particular, it appeared to be challenging to determine which of the articulated market needs should be included in the forthcoming releases of a product:

“Everyone has their own view regarding what the customer has said and how loud they have shouted. After a voice vote [in a roadmapping session] some kind of consensus will emerge describing what features we will be able to do in following 9 months and what features will not be implemented.”

One of the underlying causes that complicate the understanding of priorities is that the requirements have often dependencies. One dimension that introduced such dependencies was the global presence of a product:

”We have a challenge in the future that the ever increasing product offering should be taken into global marketplace. How to set the priorities in such situation? They are convergent to some extent [in different geographical areas] but not completely the same.”

Another dimension introducing dependencies identified in the interviews was the availability of a product in diverse customer segments:

”It is quite a challenge for our resources and processes that we need to serve pragmatic existing customers while searching new businesses and being a forerunner and a visionary.”

An articulated market need requested in a certain geographical area may benefit other regions and other customer segments to some extent. How is it possible then to determine priorities of a market need in a presence of such dependencies? According to our analysis, a great contributor to the problems of determining the priorities of market needs is the inability to understand the value of them. If the needs of market would be complemented with the potential values to the success-critical stakeholders, the prioritization and the decision processes in a roadmapping context could be made more systematic and transparent making the goals of a controller easier to achieve.

### 3.3. Disseminating the roadmapping knowledge: Distributor

A roadmap is nothing without implementation. The challenge is then to know how to disseminate the results of roadmapping to those who need to act on it. As several studies (e.g. [21, 22]) report the weaknesses on transferring knowledge solely in a written form, it is therefore necessary that there exist *distributors* in a roadmapping context. Their duty is to disseminate the knowledge produced in a roadmapping context to the ones who depend heavily on it. But who should be selected as distributors to a roadmapping context? In order to answer such question, a deeper understanding is needed on who are the consumers of roadmapping knowledge and what is the nature of their need.

In our study we investigated how information was exchanged between organizational functions in the context of product development and what challenges the companies have faced regarding such collaboration. Our study revealed that there are different levels related to the consumption of roadmapping knowledge (Figure 2).

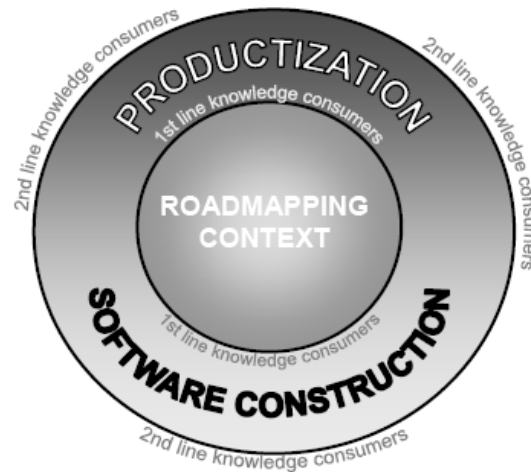


Figure 2. The lines of roadmapping knowledge consumers.

The first line knowledge consumers depend heavily on roadmapping knowledge as a roadmap is one of the main inputs to their activities. The determination of the first line knowledge consumers is a context dependent issue that varies across the companies. However, based on the interviews, two most likely candidates to be regarded as the first line knowledge consumers are those units that are responsible on construction and productization of the software product. The failure of disseminating roadmapping knowledge to such units may leave the employees of an organization ignorant:

”Our marketing department has not been able to write anything related to the new product until the product has been implemented.”

or misinformed:

”In many cases, knowing the plans for the future versions would have an impact on the design decisions. If we would know that a certain requirement is actually laying a foundation to something forthcoming, we would implement the requirement differently.”

The second line knowledge consumers are less dependent on the richness of roadmapping knowledge. Their information need can therefore be satisfied in most cases with documented form of roadmapping knowledge. In addition, they may be consumers of the knowledge produced by the first line knowledge consumers. As with the first line knowledge consumers, the determination of second line knowledge consumers is a context dependent issue. Some of the typical second line knowledge consumers identified in our study were customers, partners, sales and technical support.

#### 4. Discussion: Reflection to the Knowledge-Based Theory of the Firm

Through the study we have built our understanding regarding the complex phenomenon of roadmapping in software development. The results presented in this paper have been derived inductively from the data. When reflecting the results of this study to the literature, there appears to be a great resemblance with one particular theory. In this section, we will introduce the knowledge-based theory of the firm as it has been described by Nonaka and Toyama in [23] and point out why it appears to be relevant in the context of roadmapping.

The knowledge based theory of the firm can be described with a model of knowledge creation (SECI model) presented in Figure 3. According to Nonaka and Toyama [23], knowledge creation starts with *Socialization*, which is the process of converting new tacit knowledge through shared experiences in day-to-day social interaction. Since tacit knowledge is difficult to formalize and often time- and space-specific, tacit knowledge can be acquired only through shared direct experience, such as spending time together or living in the same environment. In our view, this phase of knowledge creation addresses the challenges of market information elicitation giving support to our finding of valuing the *visionary* type knowledge at a roadmapping context. Elicitation of visionary knowledge requires a deep understanding of the customer’s domain and surrounding environment. Such kind of knowledge can be acquired from customers, suppliers and even competitors by empathizing with them through shared experience [23].

The tacit knowledge is articulated into explicit knowledge through the process of *Externalization*. Here, dialogue is an effective method to articulate one’s tacit knowledge and share the articulated knowledge with others. Through the dialogue with individuals, one tries to see the entire picture of the reality by interacting with those who see the reality from other angles, that is, sharing their context [23]. In our view this phase represents the early parts of roadmapping activity supporting our finding that it is necessary to have *contributors* in the shared physical context of roadmapping .

Explicit knowledge is collected from inside or outside the organization and then combined, edited, or processed to form more complex and systematic explicit knowledge through the *Combination* process. The new explicit knowledge is then disseminated among the members of the organization [23]. In our view, this phase is similar to the late parts of roadmapping activities where plan for the future versions of a product is created and disseminated to the organization. This supports our finding regarding the necessity of having *distributors* in a roadmapping context.

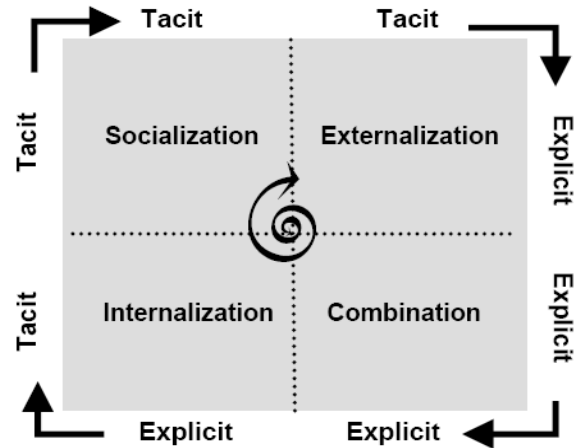


Figure 3. SECI model of knowledge creation adopted from [23].

Explicit knowledge created and shared throughout an organization is then converted into tacit knowledge by individuals through the *Internalization* process. This stage can be understood as praxis, where knowledge is applied and used in practical situations and becomes the base for new routines [23]. In our view this phase relates to the task of implementing new versions of a product and gathering experiences of it. It is thus a precursor phase for the next round of *Socialization*. Organizational knowledge creation can be considered as a never-ending process that upgrades itself continuously [23].

Similar to the roadmapping context, Nonaka and Toyama [23] defines a term *ba* as a shared context in motion, in which knowledge is shared, created, and utilized. They see organizations as organic configurations of various *ba*, where people interact with each other and the environment based on the knowledge they have and the meaning they create. Similarly to our finding of the three roles necessary in a roadmapping context (contributor, controller and distributor), Nonaka and Toyama have identified the need of having three distinct roles in a context of *ba*. That is, there need to be *innovators*, who senses the new reality first; *coaches*, who attains inter-subjectivity by interacting with the innovator and brings in his/her own viewpoint; and *activists*, who take a higher viewpoint and attain trans-subjectivity, make the new reality understandable and tangible for other people and who protects the team from outside influence so that the other roles can keep their own viewpoints.

#### 5. Conclusions

Given that the phenomenon of roadmapping and release planning in software development practice have been rather unexplored up to this point, we conducted a

qualitative study attempting to gain a deeper understanding of the challenges of knowledge and collaboration in a roadmapping context.

When interpreting the gathered data by using grounded theory as the research method, we identified three distinct roles that appeared to be necessary in a roadmapping context. Each of the roles relates to particular challenges regarding the knowledge and collaboration.

In case of *contributor*, the challenge is to determine what type of information is of most significant value at a roadmapping context. We sought for such answer by investigating the practices and the challenges faced regarding current information elicitation mechanisms and concluded that future-oriented tacit knowledge of the market will be most valuable for the organization.

The challenge of *controllers* is to introduce systematic practices to the roadmapping context and provide transparency of the made decisions. One underlying cause for experienced problems appeared to be the inability to understand the values behind the expressed needs.

One of the challenges regarding the dissemination of the roadmapping knowledge into the organization is the ability to understand who are most dependent of the roadmapping knowledge. The answer of such question will help on determining who should be taking the role of *distributor* in a roadmapping context.

When reflecting our findings to existing theories, we identified great resemblance with the knowledge-based theory of the firm [23]. Therefore, this theory could contribute to further theory development in market-oriented software development and related areas, such as Value-Based Software Engineering [14].

## 6. References

- [1] B. Boehm, "Value-Based Software Engineering", *SIGSOFT Softw. Eng. Notes*, vol. 28, pp. 3, 2003.
- [2] R. A. W. Kok and B. Hillebrand, "What makes product development market oriented? Towards a conceptual framework", *International Journal of Innovation Management*, vol. 7, pp. 137-162, 2003.
- [3] B. J. Jaworski and A. K. Kohli, "Market Orientation: Review, Refinement, and Roadmap", *Journal of Market Focused Management*, vol. 1, pp. 119-135, 1996.
- [4] R. E. Albright, "Roadmapping Convergence," "Commercializing and Managing the New Converging Technologies" -Workshop September 22 2003.
- [5] D. A. Penny, "An Estimation-Based Management Framework for Enhance Maintenance in Commercial Software Products", Proceedings of International Conference on Software Maintenance (ICSM'02), 2002.
- [6] P. Carlshamre, "Release Planning in Market-Driven Software Product Development: Provoking an Understanding", *Requirements Engineering*, vol. 7, pp. 139-151, 2002.
- [7] G. Ruhe and M. O. Saliu, "The Art and Science of Software Release Planning", *IEEE Software*, vol. 2, pp. 47-53, 2005.
- [8] G. Ruhe and M. O. Saliu, "The Science and Practice of Software Release Planning", *IEEE Software (Submitted)*.
- [9] "Home page of The Requirements Engineering Specialist Group of the British Computer Society," vol. 2006.
- [10] L. Lehtola and M. Kauppinen, "Suitability of Requirements Prioritization Methods for Market-driven Software Product Development", *Software Process Improvement and Practice*, vol. 11, pp. 7-19, 2006.
- [11] L. Karlsson, Å. G. Dahlstedt, J. Natt och Dag, B. Regnell, and A. Persson, "Challenges in Market-Driven Requirements Engineering - an Industrial Interview Study", Proceedings of Eight International Workshop on Requirements Engineering: Foundation for Software Quality, Essen Germany, 2002.
- [12] B. Regnell, M. Höst, J. Natt och Dag, P. Beremark, and H. Thomas, "An Industrial Case Study on Distributed Prioritisation in Market-Driven Requirements Engineering for Packaged Software", *Requirements Engineering*, pp. 51-62, 2001.
- [13] D. E. Damian and D. Zowghi, "RE challenges in multi-site software development organisations", *Requirements Engineering*, vol. 8, pp. 149-160, 2003.
- [14] A. Jain and B. Boehm, "Developing a Theory of Value-Based Software Engineering", Proceedings of 7th International Workshop on Economics-Driven Software Engineering Research (EDSER), St. Lois, Missouri, 2005.
- [15] B. Glaser and S. A.L., *The Discovery of Grounded Theory: Strategies for Qualitative Research*. Chicago: Aldine, 1967.
- [16] A. L. Strauss and J. Corbin, *Grounded Theory Procedures and Applications*. Newbury Park, CA: Sage Publications, 1990.
- [17] G. Paré and J. J. Elam, "Using Case Study Research to Build Theories of IT Implementation", Proceedings of The IFIP TC8 Working Conference on Information Systems and Qualitative Research, Philadelphia, Pennsylvania, United States, 1997.
- [18] R. L. Baskerville, B. Ramesh, L. Levine, J. Pries-Heje, and S. Slaughter, "Is Internet-Speed Software Development Different?" *IEEE Software*, vol. 20, pp. 70-77, 2003.
- [19] K. Smolander, M. Rossi, and S. Puraio, "Going beyond the blueprint: Unraveling the complex reality of software architectures", Proceedings of 13th European Conference on Information Systems (ECIS 2005), Regensburg, 2005.

[20] S. Jantunen and K. Smolander, "Towards Global Market-Driven Software Development Processes: An Industrial Case Study", Proceedings of 28th International Conference on Software Engineering & Co-Located Workshops. The first International Workshop on Global Software Development for the Practitioner (GSD 2006), Shanghai, China, 2006.

[21] J. E. Orr, *Talking about Machines: An Ethnography of a Modern Job*. Ithaca, Ny: ILR Press, 1996.

[22] I. Nonaka, "A Dynamic Theory of Organizational Knowledge Creation", *Organization Science*, vol. 5, pp. 14-37, 1994.

[23] I. Nonaka and T. Ryoko, "The knowledge-creating theory revisited: knowledge creation as a synthesizing process", *Knowledge Management Research & Practice*, vol. 1, pp. 2-10, 2003.

# Lightweight Replanning of Software Product Releases

Thamer AlBourae  
Software Engineering  
Decision Support Labs,  
University of Calgary,  
albourae@cpsc.ucalgary.ca

Guenther Ruhe  
Software Engineering  
Decision Support Labs,  
University of Calgary,  
ruhe@ucalgary.ca

Mahmood Moussavi  
Schulich School of  
Engineering,  
University of Calgary,  
moussavm@enel.ucalgary.ca

## Abstract

*Well defined product features are the essence of good product management. High quality features lead to successful software products, both functionally and financially. One of the crucial processes in software product management is release planning where features are assigned to releases. Volatile features, resources and stakeholder preferences have been recognized as factors that decrease release quality. In this paper, we propose a lightweight replanning process model where old features are compared with newly added ones using the Analytical Hierarchy Process (AHP). Then, a greedy replan algorithm is applied to select the most promising features to accommodate changing market driven product demands.*

## 1. Introduction and Motivation

As incremental development gains greater acceptance in the software product development society, the farther we move from the monolithic waterfall model, and the greater the importance of the release planning process becomes. Incremental software product development allows customers to receive deliverables early. This means the business value of the product is recognized before the product is fully delivered. Furthermore, it opens the door for early feedback from clients. This implies the incorporation of the feedback into future product releases to gain higher customer satisfaction.

Features are the basis of software release products. They provide a foundation to plan and re-plan software releases, estimate effort and resources constraints and reflect the customer expectations of the product. Release planning is a process where we make decisions on which features are assigned to which releases. The planning process assigns incremental

components of the software system to be delivered on a specific calendar date. These sub-systems are planned in sub-releases to be delivered incrementally. As change requests arrive, plans need to be adjusted (replanned) for newly added features.

The process of planning or re-planning releases considers several aspects, including effort estimation, resource constraints and stakeholder preferences to gain higher customer satisfaction [1].

As the software product development process begins, a continuous stream of new change requests arrive. This is common in market driven software products [2]. These changes imply the modifications of some features or the addition of new ones. Since not all features can be implemented due to effort and resources boundaries, we have to decide which feature should be delivered and which have to wait. Moreover, we cannot neglect these changes requests since a high level of stakeholder satisfaction is important.

In this paper, we present the foundation for a process model to handle change requests and re-plan software releases. The model pools all available features ready to be allocated to releases. Then, we apply the analytical hierarchy process (AHP) [3] to define a degree of importance for each feature.

The rest of the paper is organized as follows. In Section 2, we present literature models related to the problem. In Section 3, we describe the problem being addressed. In Section 4, we formalize the re-planning problem from a decision support perspective. In Section 5, we perform a case study to demonstrate the contribution of the new approach. In Section 6, we summarize the findings and give an outlook for future work.

## 2. Related Work

### 2.1 Release Planning

Release Planning has gained interest in market driven software product development [4], [5]. This selection process has to take into consideration multiple stakeholder preferences and available resources. In addition, features interdependencies [6] must be handled. Feature interdependencies can force a low-priority feature to be implemented before a higher-priority one. This may decrease the client satisfaction and could lead to losing market share. Thus, the decision making process of selecting features and the tradeoffs involved are complex.

Various approaches address the problem of Release Planning (RP) both in industry and academia. Anton emphasized that complex software projects are likely to fail without a plan [7]. Penny [8] proposed a high level approach to RP where the estimation of the total effort required for developing a project features should fall in a certain confidence level. The planning game (GM) in extreme programming tackled the same problem in [9]. Others looked at the problem from an optimization point of view like [10] and [11]. Ruhe *et al* proposed a model (EVOLVE\*) where a synergy between the computational intelligence and human decision maker is combined [12].

The replanning further adds to the complexity of the problem. Considering new changes, maintaining the stakeholders' satisfaction and adjusting releases to fit within available resources all have to be accounted for.

The re-planning of a product release is used to improve the process of release planning and increase the quality of the releases [13]. Furthermore, it makes us learn from the previous mistakes we committed in case a retrospective analysis is conducted [14].

Davis's paper [15] addresses a special case of the release planning problem where changes take place and we have to select a new set of requirements for the next release satisfying a given time and resource capacity.

### 2.2 Change-Based Process Models

Many Change-Based process models have been proposed in the literature. These models include the PRISM process model [16] and the FEAST process model [17]. PRISM is concerned with change propagation from environmental perspective while the FEAST model is more focused on the software process as it changes over time. Other studies proposed a Change Maturity Model based on their work on the

Design Improvements in Requirements Evolution (DESIRE) project [18].

### 2.3 Change Categorization

Many studies addressed the importance of categorizing changes to define processes for each category. Harker and Eason [19] proposed a classification to distinguish between stable and volatile features (emergent, consequential, mutable, adaptive and migration). Other studies agreed that categorization is a wise step towards better change management [20]. Some literature focused on the problem from a management perspective and categorized changes based on a cause-effect diagram. Then, different scenarios were identified, and for each scenario the authors define: a goal to be achieved, strategy to be adopted, metric to be used and failure mode to be avoided [21].

Some industrial organizations studied the problem of change to define classification schemes. For example, Stark's *et al* studied the problem [22] on 44 releases. Then, depending on the data collected, the study proposed a formula to predict the time needed to implement these changes. Also, the study extracted some guidelines to be considered in the process of release planning. On the same path, Higgins *et al* [23] studied the features change problem to overcome the potential of bottlenecks that can occur due to lack of resources. Also, the study defined a process that handles changes requests as they arrive.

### 2.4 Other Related Research

Other research provided some guidelines for handling changes [24]. For instance, identifying change early in the product development cycle, providing a process to incorporate changes in the development cycle and reducing the amount of change by investigating its origins, are all considered change handling fundamentals.

Other studies address the concept of incremental evolutionary delivery [25] or tackle the problem of categorizing changes [26].

## 3. Problem Description

Figure 1 illustrates the process of newly arriving features competing against those already planned for, where some features are already implemented. In the beginning of the product development cycle, a set of features  $F(i) = (f_1, f_2, \dots, f_n)$  are assigned to releases to be delivered in a certain sequence. As the development



process starts on  $t_0$  (moment of time), change requests arrive on  $t_1$  with new added features  $\Delta F(i) = (f_{n+1}, f_{n+2}, \dots, f_{n+m})$  and we have to re-schedule the next release to be delivered on  $t_2$ . The dashed line represents  $t_1$ , when these changes requests arrive,  $t_2$  when the delivery of the release being developed is shifted to release time  $t_2$ , and the dotted line separates the old and new features.

Figure 1 shows the different categories of features at a given time  $t_1$ . We have identified four main categories of features in this scenario, these include:

- A. Implemented features: features which are already implemented and they are not included in the process of re-planning.
- B. On going features: features which are in the process of being implemented.
- C. Planned for features: features which are planned to be implemented in the future, but have not been started yet.
- D. Added features: features proposed from stakeholders to be added to the next release.

Thus, we can formulate the problem of replanning of the release being developed by defining a function to find a new set of features  $F'$  out of the previous set of feature  $F$  plus the new added ones  $\Delta F$ , where:  $F' \subset F + \Delta F$ .

The new set  $F'$  contains the best possible set of features that provide the highest stakeholders satisfaction while having a limited capacity of effort and time available for the next release.

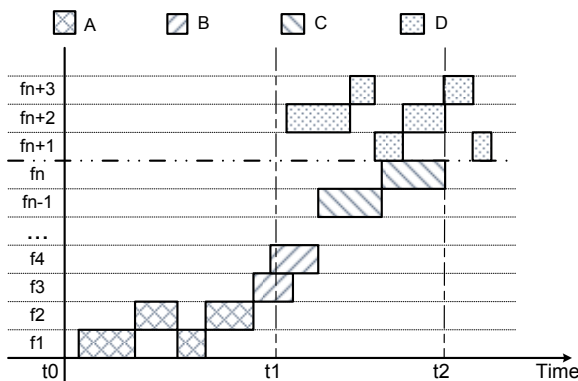


Figure 1: The Re-plan problem

## 4. Lightweight Replan

### 4.1 Overview

The main goal of the proposed Lightweight Replan model is to develop a new product plan that achieves higher stakeholder satisfaction given a limited capacity of time and resources. The Lightweight character reflects the fact that replanning consumes a

considerable amount of the Product Manager's time and effort [13]. Moreover, the computation complexity is reduced by handling fewer change requests instantly. This means incorporating changes earlier in the product life cycle where changes cost less. Applying the heavily weight release planning process proposed in [1] is time consuming and of a high complexity. This would lead to delay incorporating the changes proposed to later phases where implementing these changes will cost more. In this section, we present the Lightweight Replan process model and describe the individual steps of it in details.

Figure 2 shows a generic process model describing main release replanning activities including their inputs and outputs. These activities work together to produce a framework for replanning product releases. It is noteworthy that the activities inside the shaded dashed box are part of the product release planning process [1], while other activities complete the proposed Lightweight Replan model.

In this model, three main roles are recognized – *Product manager*, who is responsible of the whole development process – *Stakeholders*, which include any team member who are concerned with the product development – and *the supporting environment*, which facilitates the achievement of the processes goals. The supporting environment can be processes or tools that support the process. Major activities are represented in rectangles while outputs of any process are represented in ovals.

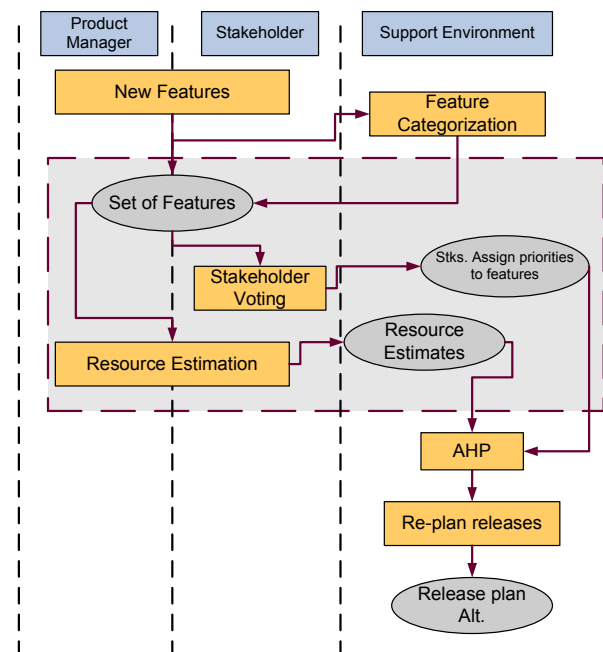


Figure 2: Generic process model for the Lightweight Re-planning

Every activity or output is engaged with the roles appearing at the top, within that activity or outcome's "swim lane". For example, the product manager and other stakeholders contribute to the process of "Resource Estimation" while the supporting environment facilitates the achievement of "Resources Estimates" through a tool. In the next section, we describe each activity or output in detail.

## 4.2 Main Steps

**4.2.1 New Features.** As the product development starts, the development team begins receiving new change requests for the features sets. The change requests received are added to the old sets of features and directed to be categorized by the feature categorization process.

**4.2.2 Feature Categorization.** Adopted from Higgins's et al work [23], change requests should be categorized to distinguish between duplicated features, on-going features or newly added ones. This process also helps assure that we have a complete and consistent set of features.

**4.2.3 Stakeholder Voting.** Stakeholders  $Stk(p) = \{Stk(1), Stk(2), \dots, Stk(q)\}$  are a group of people who are concerned with the software product plan in a direct or indirect way. They may include team members from different organizational levels such as management, development or customer services. In the previous release planning process, an objective function is used to maximize stakeholder's preferences considering their relative weight of importance while observing the available resource constraints. To learn more about the process and how it is conducted, we refer the reader to [12]. Furthermore, stakeholders are required to assign to each new feature a level of attractiveness.

**4.2.4 Resource Estimation.** Resources capacities are one of the boundaries that should be considered when replanning product releases. The main aim is to determine the likely usage of effort  $Effort(f_i)$ , and time  $Time(f_i)$  for each feature  $f_i$  for the next release. Resource estimation is a complex problem on itself and far beyond the scope of our research. Thus, we encourage the reader to refer to [27] for detailed approaches to effort estimation.

In our process model, we need to consider the rate of consumption for implemented and ongoing features when the changes were received at  $t_1$ . Also, we estimate effort and time needed for new features. Other

feature categories estimations and releases capacities are assumed to remain unchanged. Figure 3 describes the effort estimation process for features and the release being developed.

The main goal is to maintain the effort and time available as we replan so that the new replanned release does not exceed the capacity available.

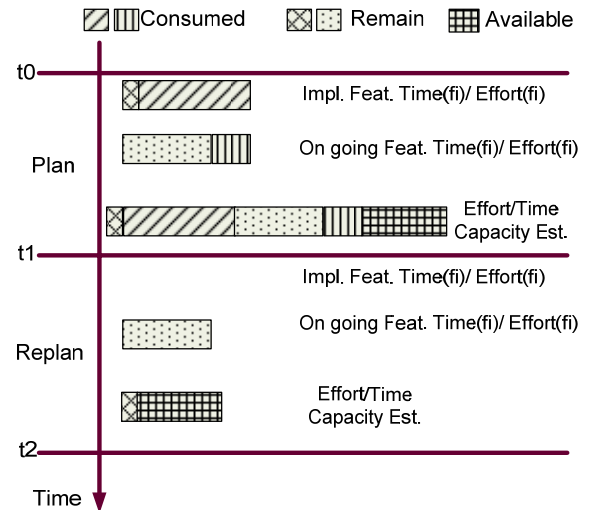


Figure 3: Effort and time estimation over Time  $(t_0, t_1)$

**2.5 The Analytical Hierarchy Process (AHP).** The Analytical Hierarchy Process (AHP) is a multi criteria decision making method [3]. This process elicits experts' preferences in a formalized manner using a pair-wise comparison technique. Each expert evaluates a pair of features with respect to a defined criteria. Figure 4 shows the pair-wise comparison prioritization process using AHP.

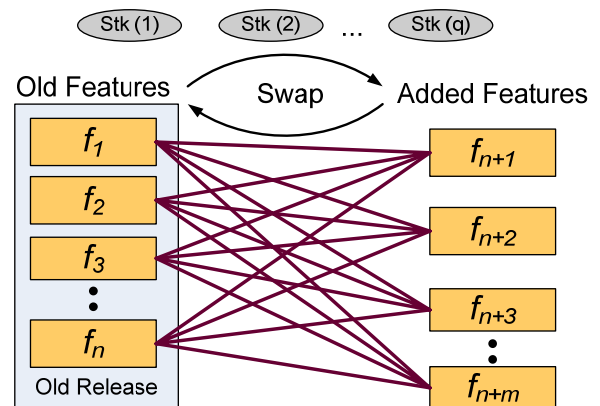


Figure 4: The Analytical Hierarchy Process (AHP) pair wise comparison

Each stakeholder compares between the old set of features  $F$  and the newly added features  $\Delta F$  with

respect to the degree of attraction. The Weighted Average Satisfaction (WAS) was already computed for F during the previous planning process [1]. Assuming that the WAS values remain unchanged, each stakeholder compares a pair of features (an old feature from F with a newly added one from  $\Delta F$ ) to express the level of attraction. In this situation, some old low attractive features are substituted or swapped with newly added ones. Then, the most promising features are gathered in a new set to enter into the Greedy Replan algorithm in the next step.

Figure 5, 6 and 7 illustrates the Expert's Choice tool [28] where features are compared using the AHP technique based on urgency and time.

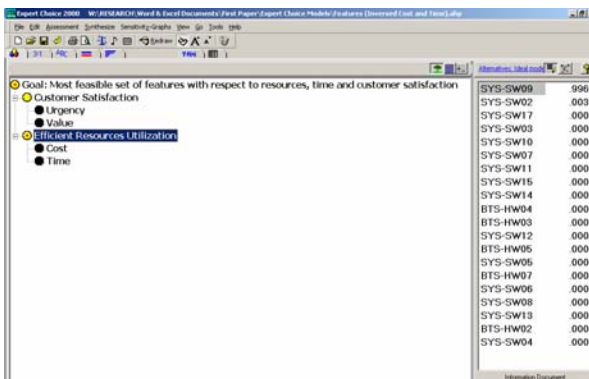


Figure 5: The Analytical Hierarchy Process in the tool

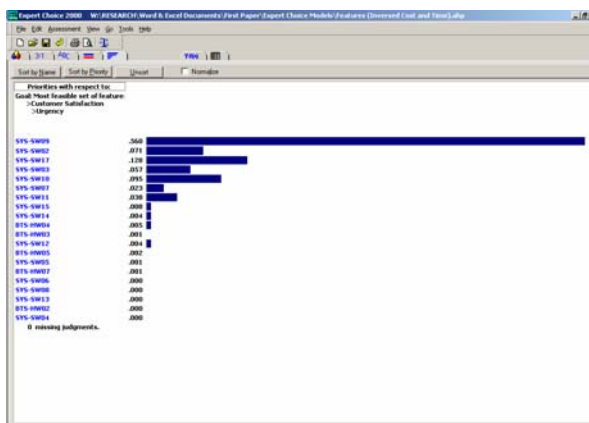


Figure 6: Pair wise comparison based on urgency (time needed to deliver a feature to the market)

**4.2.6 Replanning Releases.** The information gained from the AHP pair-wise comparison process is used to select a new set of features F' to be assigned to the new (replanned) release.

Selecting the most promising set of features F' to form a replanned release is a combinatorial

optimization problem. This problem is known as a type of Knapsack problem where given a set of features, we try to maximize the Weighted Average Satisfaction (WAS) while maintaining the available effort and time capacity. The algorithm to solve such problems are time consuming and of a high complexity [29].

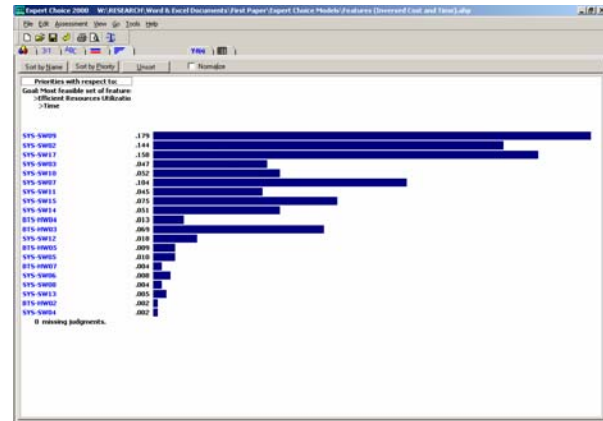


Figure 7: Pair wise comparison based on the time estimation Time ( $f_i$ ) of each feature

In our process model, we adopt a lightweight greedy algorithm which does not give an optimum solution, yet it gives an approximation of it. The greedy approximation algorithm was proposed by Martello and Toth. For learning more about this algorithm, we refer the reader to [30].

A simple pseudo code of the Greedy Replan algorithm can be expressed as follows:

1. Define an objective Function  $F(f_i) = 0$ .
2. Define  $F' = \emptyset$  and empty set of features.
3. Define Effort = 0, Time = 0 (Effort and Time Capacity).
4. Sort all features with the best attractiveness ratios gained from the AHP comparison in a descending order.
5. While Effort and Time are less than the release capacity C do steps 6 - 9.
6. Add feature  $f_i$  to  $F'$
7.  $F(f_i) = F(f_i) + WAS(f_i)$
8. Effort = Effort + Effort ( $f_i$ )
9. Time = Time + Time ( $f_i$ )
10. Return  $F'$ ,  $F(f_i)$ , Effort, Time

## 5. Case Study: Re-planning product releases in a Telecommunication project

To demonstrate the contribution of the proposed process model we conduct a case study in on a project from the telecommunication using a web-based tool called ReleasePlanner<sup>®</sup> [28]. The tool is implemented

based on the hybrid intelligent approach proposed in [12]. The project has 7 stakeholders with different perspectives and degrees of importance.

On  $t_0$ , the project starts with 15 features. At  $t_1$ , the project receives change requests from different stakeholders while the team is still developing release 1. We categorize features as described in Section 4.2.2.

At the time changes arise ( $t_1$ ), we have 3 implemented features (Feat.), 5 ongoing, 7 planned for, and 4 added. Table 1 shows this categorization with each feature ID.

According to each feature category, we re-estimate the capital and time resources required as described in section 4.2.4.

Table 1: Features categorization with each feature ID

N #	Belong to Set	Feat. Type	Feat.	Feature ID
1	F	Impl.	$f_1$	BTS-HW01
2			$f_2$	BTS-SW01
3			$f_3$	SYS-SW01
4	F	On going	$f_4$	SYS-SW09
5			$f_5$	SYS-SW07
6			$f_6$	SYS-SW06
7			$f_7$	SYS-SW08
8			$f_8$	SYS-SW04
9	F	Planned For	$f_9$	SYS-SW02
10			$f_{10}$	SYS-SW17
11			$f_{11}$	SYS-SW03
12			$f_{12}$	SYS-SW10
13			$f_{13}$	SYS-SW11
14			$f_{14}$	BTS-HW07
n=15	$\Delta F$	Added	$f_{n+1}$	BTS-HW02
16			$f_{n+2}$	SYS-SW15
17			$f_{n+3}$	SYS-SW14
18			$f_{n+4}$	BTS-HW04
19			$f_{n+4}$	BTS-HW03

Then, we compare the old set of features (F) with newly added ones ( $\Delta F$ ) using AHP. During the AHP comparison process, we use the Weight Average Satisfaction (WAS) assigned to old features in the previous release planning process and compare it with the level of attraction assigned to the new features ( $\Delta F$ ).

As we apply the Greedy Replan algorithm (described in section 4.2.6) the new (replanned) release is formed. Comparing the new (replanned) release with the old release with respect to the WAS, capital and time usage, we have:

Table 2 shows the Old release compared to the replanned one with respect to the WAS, while Table 3 shows the Old release compared to the replanned one with respect to Capital and Time.

Table 2: The old and new (replanned) releases structures

Old Release		Replanned Release	
Feat.	WAS	Feat.	WAS
$f_1$	--	--	--
$f_2$	--	--	--
$f_3$	--	--	--
$f_4$	616	--	--
$f_5$	1632	$f_5$	1632
$f_6$	2004	$f_6$	2004
$f_7$	654	--	--
$f_8$	749	--	--
$f_9$	1485	$f_9$	1485
$f_{10}$	1325	$f_{10}$	1325
$f_{11}$	1296	$f_{11}$	1296
$f_{12}$	723	--	--
$f_{13}$	305	--	--
$f_{14}$	967	--	--
$f_{15}$	1114	$f_{15}$	1114
--	--	$f_{n+1}$	1962
--	--	$f_{n+2}$	1911
--	--	$f_{n+4}$	1593
Sum	12870	14322	

Table 3: The old and new (replanned) releases Capital and Time usage

Old Release			Replanned Release		
Feat.	Capital	Time	Feat.	Capital	Time
$f_1$	--	--	--	--	--
$f_2$	--	--	--	--	--
$f_3$	--	--	--	--	--
$f_4$	0	800	--	--	--
$f_5$	160	432	$f_5$	160	432
$f_6$	400	560	$f_6$	400	560
$f_7$	400	856	--	--	--
$f_8$	800	980	--	--	--
$f_9$	25	1150	$f_9$	25	1150
$f_{10}$	50	830	$f_{10}$	50	830
$f_{11}$	100	2200	$f_{11}$	100	2200
$f_{12}$	300	1600	--	--	--
$f_{13}$	500	1150	--	--	--
$f_{14}$	750	2045	--	--	--
$f_{15}$	1500	2130	$f_{15}$	1500	2130
--	--	--	$f_{n+1}$	0	550
--	--	--	$f_{n+2}$	50	650
--	--	--	$f_{n+4}$	200	300
Sum	4985	14733		2485	8802

It is obvious that the new replanned release gained a higher WAS compared to the old release. Moreover, the replanned release requires less capital and time in order to be delivered. However, the number of features included in the replanned release is less than the old release.

It is obvious that the old release had more features planned for implementation. However, our new (replanned) release has a higher Stakeholder satisfaction level with lower resource usage.

We refer the reader to our electronic version of the empirical case study for more details [31].

## 6. Conclusions and Discussion

The lightweight replanning process model provides a basis for incorporating changes instantly into the development lifecycle. The lightweight factor is beneficial since it will reduce the cost of introducing changes in the development cycle.

The light weight Replanning process model can be applied instantly to incorporate changes in the development cycle as soon as possible. This is recognized as one of the fundamentals used for handling changes [24].

Integrating the process of categorization of changes facilitates the re-planning process phases where we select features with higher value while not exceeding effort capacity available.

The model is of a low computation complexity compared with the release planning process proposed in [12]. This is considered good where the replanning must be achieved quickly especially in dynamic markets. Furthermore, light weight replanning enables performing the process at any time. This is considered to be common where the rate of change is often high [22] particularly in market driven products [2].

Some open issues to be explored include validating the lightweight replanning approach empirically. Also, deciding when we should replan and when we should not, possibly by defining a threshold, is an issue that needs to be explored. Another related issue is discerning how frequently the replanning process should take place. Another area of future work involves integrating an impact analysis process to quantify the change effects on the existing implemented features.

The Lightweight Replan model best fits products that are market driven where maintaining a higher stakeholder satisfaction is desired. However, the model and the presented results are limited to the data and the empirical study we conducted so far. Thus, there is a need for real world industrial experiments to validate the model and the results.

## Acknowledgement

We thank the Ministry of Higher Education of Saudi Arabia and Alberta Informatics Circle of

Research Excellence (iCORE) for their financial support of this research. We also thank Jim McElroy and the anonymous reviewers for their detailed and valuable comments.

## References

- [1] G. Ruhe and M. O. Saliu, "The art and science of software release planning," *IEEE Software*, vol. 22, pp. 47-53, 2005.
- [2] P. Carlshamre and B. Regnell, "Requirements lifecycle management and release planning in market-driven requirements engineering processes," In Proceedings IEEE International Workshop on the Requirements Engineering Process: Innovative Techniques, Models, and Tools to support the RE Process, pp. 961-5, Greenwich, UK, 2000.
- [3] T. L. Saaty, *The analytic hierarchy process*. New York: McGraw-Hill, 1980.
- [4] J. Karlsson and K. Ryan, "A cost-value approach for prioritizing requirements," *Software, IEEE*, vol. 14, pp. 67-74, 1997.
- [5] J. Karlsson, S. Olsson, and K. Ryan, "Improved practical support for large-scale requirements prioritising," *Requirements Engineering*, vol. 2, pp. 51-60, 1997.
- [6] P. Carlshamre, K. Sandahl, M. Lindvall, B. Regnell, and J. Natt och Dag, "An industrial survey of requirements interdependencies in software product release planning," In Proceedings 5th IEEE International Symposium on Requirements Engineering, pp. 84-91, Toronto, Ont, 2001.
- [7] A. I. Anton, "Successful software projects need requirements planning," *Software, IEEE*, vol. 20, pp. 44, 46, 2003.
- [8] D. A. Penny, "An estimation-based management framework for enhanceive maintenance in commercial software products," In Proceedings ICSM International Conference on Software Maintenance, pp. 122-130, 2002.
- [9] B. A. Nejme and I. Thomas, "Business-driven product planning using feature vectors and increments," *Software, IEEE*, vol. 19, pp. 34-42, 2002.
- [10] A. J. Bagnall, V. J. Rayward-Smith, and I. M. Whittle, "The next release problem," *Information and Software Technology*, vol. 43, pp. 883-90, 2001.

- [11] J. Ho-Won, "Optimizing value and cost in requirements analysis," *Software, IEEE*, vol. 15, pp. 74-78, 1998.
- [12] G. Ruhe and A. Ngo-The, "Hybrid Intelligence in Software Release Planning," *International Journal of Hybrid Intelligent Systems*, vol. 1, pp. 99-110, 2004.
- [13] J. Momoh and G. Ruhe, "Release Planning Process Improvement - An Industrial Case Study," *Software Process: Improvement and Practice*, vol. 11, pp. 295-307, 2006
- [14] L. Karlsson, B. Regnell, and T. Thelin, "Case Studies in Process Improvement through Retrospective Analysis of Release Planning Decisions " *International Journal of Software Engineering and Knowledge Engineering*, June, 2006 (in press).
- [15] A. M. Davis, "The art of requirements triage," *Computer*, vol. 36, pp. 42-49, 2003.
- [16] N. H. Madhavji, "Environment evolution: the Prism model of changes," *Software Engineering, IEEE Transactions on*, vol. 18, pp. 380-392, 1992.
- [17] M. M. Lehman, "Feedback in the software evolution process," *Information and Software Technology*, vol. 38, pp. 681-6, 1996.
- [18] W. Lam and V. Shankararaman, "Managing change in software development using a process improvement approach," In Proceedings 24th EUROMICRO Conference, pp. 779-786, Vasteras, Sweden, 1998.
- [19] S. D. P. Harker, K. D. Eason, and J. E. Dobson, "The change and evolution of requirements as a challenge to the practice of software engineering," In Proceedings IEEE International Symposium on Requirements Engineering pp. 266-272, San Diego, CA, USA, 1992.
- [20] D. Rowe, J. Leaney, and D. Lowe, "Defining Systems Evolvability - a Taxonomy of change," In Proceedings International Conference and Workshop on Engineering Computer Based Systems, pp. 45-52, Jerusalem, Israel, 1998.
- [21] W. Lam and V. Shankararaman, "Requirements change: a dissection of management issues," In Proceedings 25th EUROMICRO Conference. Informatics: Theory and Practice for the New Millennium, pp. 244-251, Milan, Italy, 1999.
- [22] G. Stark, A. Skillicorn, and R. Ameen, "An Examination of the Effects of Requirements Changes on Software Maintenance Releases," *Journal of Software Maintenance: Research and Practice*, vol. 11, pp. 293-309, 1999.
- [23] S. A. Higgins, M. De Laat, P. M. C. Gieles, and E. M. Geurts, "Managing requirements for medical IT products," *IEEE Software*, vol. 20, pp. 26-33, 2003.
- [24] R. C. Sugden and M. R. Strens, "Strategies, tactics and methods for handling change," In Proceedings IEEE Symposium and Workshop on Engineering of Computer-Based Systems, pp. 457-463, Friedrichshafen, Germany, 1996.
- [25] T. Gilb, *Principles of Software Engineering Management* Addison-Wesley, England, 1988.
- [26] G. Kotonya and I. Sommerville, *Requirements Engineering: Processes and Techniques*: John Wiley & Sons Ltd, 1998.
- [27] L. C. Briand and I. Wiecek, "Resource Estimation in Software Engineering " in: *Marciniak JJ (ed) Encyclopedia of software engineering (2nd edition)*. New York: John Wiley, 2005.
- [28] <http://www.expertchoice.com/>.
- [29] A. Ngo-The and G. Ruhe, "Optimized Resource Allocation for Software Release Planning," *Software Engineering, IEEE Transactions on*, 2005 (Submitted).
- [30] M. R. Garey and D. S. Johnson, *Computers and intractability : a guide to the theory of NP-completeness*. San Francisco: W. H. Freeman, 1979.
- [31] T. AlBourae. Calgary, AB: <http://sern.ucalgary.ca/%7Ealbourae/>, 2006.

# A Cost-Based Approach to Software Product Line Management

Holger Schackmann, Horst Lichter

{schackmann, lichtner}@informatik.rwth-aachen.de

RWTH Aachen University. Software Construction Group. Ahornstr. 55, 52074 Aachen, Germany

## Abstract

*The evolution of a software product line requires different product management practices compared to a single product since the diverging requirements of different customers must be coordinated to preserve the common product line architecture. Allowing too much variability leads to substantial follow-up costs during the lifecycle. This paper describes the challenges of product management for an evolving software product line. A costing approach is proposed to enable more transparency on the costs of variability, support sound decisions on the appropriate amount of variability and gain control on the development process.*

## 1. Introduction

The development of a software product line (SPL) aims at exploiting commonalities between the products in all phases of the development process. What are the differences between product management for an SPL and product management for a single product?

The management of a product within an SPL can not be detached from the management of the other products due to dependencies between development artifacts and use of the same resources. Thus product management in SPLs is characterized by a large number of stakeholders that directly or indirectly exert influence on product management and can be affected by its decisions.

Furthermore, products may be situated in different stages of their lifecycle. While a set of products will be established as product line members with regularly maintenance releases, there will be ongoing development of new products and possibly the integration of formerly independent products.

Thus product managers must constantly aim at aligning the diverging needs of different products towards the SPL approach; otherwise the promised synergies can not be achieved.

The remainder of this paper is organized as follows. Section 2 depicts the difficulties imposed upon management of an SPL. Section 3 draws comparisons to variability management in production engineering. Section 4 then proposes an approach to variability management for SPLs based on the application of activity based costing. Section 5 gives an outlook on further work and section 6 provides a summary.

## 2. Challenges of product management for software product lines

In the following we will depict a list of challenges for SPL management, also based on observations with industrial cooperation partners.

### 2.1. Insufficient lifecycle scoping

Scoping is a part of domain analysis for SPLs. According to Schmid the task of scoping is planning and bounding what should be made reusable [1]. Based on an analysis of the commonalities and variabilities of the underlying domain, it must be decided which features should be present in all products, which should be implemented as optional or alternative parts, and which features should be developed in a product specific way.

While existing scoping approaches are focused on initial scoping (see [1] for an overview), the problem of scoping during maintenance and evolution is rather unexplored. In practice we encounter that SPLs are introduced gradually by exploiting commonalities between products that had been developed in customer specific projects. Thus an exhaustive scoping was not performed or even no systematic approach to scoping was used. Moreover scoping decisions may not be documented, or lost during further development.

### 2.2. High coordination efforts

During maintenance and evolution of an SPL, new or changed requirements have to be integrated which

do not fit to the initial scope and therefore require the adoption of existing core assets as well as introducing additional variation points. Since these changes may affect other existing or planned products, each change must be coordinated with other development projects within the SPL. Moreover, different customers may impose conflicting constraints on delivery dates. This requires considerable efforts for coordination.

### **2.3. Hindrances to systematic reuse**

Since software development takes place under continual deadline pressure it may often be faster to produce a customer specific solution instead of spending additional effort on investigating chances for reuse and coordinating necessary changes induced by an urgent customer request with other products.

Within an SPL there may typically be products which have a bigger impact on business success, need larger resources and undergo a more dynamic development. These products have stronger impact on core asset development whereby requests related to other products may not be considered sufficiently. Hence developers of less important products may again be coerced to construct product specific solutions.

### **2.4. Insufficient product communication**

The many temptations to product managers to introduce new customer specific features or adaptations of existing features, lead to a large amount of variability accompanied by an increasing technical and cognitive complexity.

The offering of features within an SPL can therefore not completely be communicated to the customer. Thus aligning customer requests with existing features of the SPL as well as identifying chances for reuse becomes difficult. Either it is not known to the analyst that there exist other potentially reusable features, or the similarities to features requested by the customer are not recognized. Similar features may be developed independently several times. If this is perceived later, it will require considerable efforts to merge these features into shared assets. But if no counteraction is performed, the scarce resources of the development organization will increasingly be loaded with coordination efforts.

### **2.5. Lifecycle costs of variability**

Added initial development costs due to missed chances for reuse are not the only problem caused by large variability. Each customer specific feature must be maintained, integrated in subsequent releases, tested

separately and possibly considered during deployment, user training and customer support. Requirements engineering for new and existing products becomes more complex and therefore costlier.

Summarized, variability is a cost driver during the whole SPL lifecycle.

### **2.6. Misunderstood customer orientation**

The concept of customer orientation may guide product managers to fulfill many new customer requests with new customer specific features. But with regard to the follow-up costs, it must be examined whether a new feature offers a corresponding business value to the customer, or if the request can also be fulfilled with a similar existing feature. In this case it must be pointed out to the customer that an adaptation of the requirements can result in a better fit to the SPL. The customer can profit by better quality, better support and reduced maintenance costs, if his product relies more on the shared assets.

### **2.7. Lack of economic incentives**

The mentioned difficulties indicate that managing variability is the core task of product management for SPLs.

One problem is that there is no sound basis of decisionmaking. Development costs of a customer specific feature can possibly be estimated. But neither the resulting follow-up costs will be transparent; nor the business value of a specific feature will usually be assessed. Therefore there is no clarity on the effects of variability on costs and accordingly a lack of economic incentives to guide managerial decisions.

We believe that this is a major barrier for successful further development of an SPL in the long term, which not had been sufficiently addressed by SPL research yet. Existing cost models for SPLs rather follow a top down approach (see [2] for an overview) and support SPL investment decisions on a high level. Developing an approach to gather the costs of variability more accurately will provide complementary input to enable better estimations of future costs and sound variability management decisions.

## **3. Variability management in production engineering**

The depicted challenges resemble a situation perceived in manufacturing industry during the eighties. Due to competitive pressure manufacturers increased the variety of their product portfolio with the objective of product differentiation. Only later it was



recognized that the expected economies of scale were overestimated, and the costs of additional variants (diseconomies of scope) were not considered sufficiently [3]. The added complexity leads to increased costs in all phases of the product lifecycle, including development, production, sales and customer support.

### **3.1. Deficiencies in costing systems**

A large part of these costs can not directly led back to corresponding product variants. In traditional cost accounting systems these overhead or indirect costs are allocated to products on a per-unit basis. This leads to an unconsciously cross-subsidization. More exotic product variants will be sold below their actual costs, thus leading to further competitive disadvantages for standard products whose prices are charged with increasing overhead costs. A collateral effect of complexity in the product portfolio is the deterioration of market power due to a declined effectiveness of the distribution system, longer reaction rates on the market and cannibalism in the product portfolio.

### **3.2. Variability management**

Due to these problems variability management went into the focus of product management. It was realized, that the maximum benefit neither lies in arbitrary widening the product portfolio nor in radical avoidance of new variants. The costs of additional variants must be balanced with the value offered to the customer. The achievable additional customer satisfaction does only increase on a degressive scale with the number of possible variants. Therefore product management has to find the economic optimum in the number of variants.

### **3.3. Activity based costing**

The depicted deficiencies of traditional costing methods led to the development of new approaches like activity based costing [4]. This approach seeks to identify the activities in the product lifecycle and to determine their costs. Cause and effect relationships must be analyzed to find out how to allocate activity costs to products, services or customers. So most of the former overhead costs can be allocated more accurately. This enables the identification of unprofitable products, services, or customer relations. Product managers can react in many different ways to establish profitable customer relationships, e.g. by changing the prices, reconfiguring or replacing products, improving production processes, changing

the business strategy or eventually abandon a product completely.

### **3.4. Analogies to software product line management**

Can these approaches be transferred to the management of SPLs? Analogously the costs of developing and maintaining an SPL are for the greater part indirect or overhead costs, which can not easily be allocated to a certain product or customer.

But since there are no or only marginal production costs in software development, cost structures are totally different to production engineering. Costs are mostly independent from the number of sold units of a product variant. Therefore one can not easily divide between standard products and exotic products in the product portfolio and assume the latter generate more overhead costs. With an adequate product line architecture those exotic products might be easily configured based on the core assets.

But as described before, the number of customer specific features has a large impact on lifecycle costs. Therefore one can analyze which features are standard features that are relevant for most customers and which features can be seen as more exotic features, only included in products for one or a few customers. Gathering the costs caused by these features more accurately, would help to attain more clarity on the influence of variability on costs.

The prevailing approaches to costing in software development have the limitation that they are unable to gather the costs caused by variability. Costs are usually allocated either to customer projects, maintenance projects or internal development projects. This may suffice for accounting and budget control. But with multiple reuse the relationship between direct labor hours that went into development and the costs of software breaks down [5]. Fichman and Kemerer therefore propose the adoption of activity based costing to component based software development. Since systematic reuse is the core of SPL development, an activity based costing approach will presumably be suited for the information needs of managing SPLs.

## **4. Variability management for software product lines**

In this section an approach to variability management is proposed, based on activity based costing and the assessment of the customer value of the variability. Subsections 4.2 to 4.4 depict how this information can be utilized to achieve the following goals:

- Estimation of future costs of variability
- Guidance of scoping decisions and strategic steering of variability
- Process improvements in SPL development and customer support

#### 4.1. Activity based costing for software product lines

The application of activity based costing to software development can be based on several existing techniques. Defined development processes aid in identifying the relevant activities. Change request management systems, task management systems or time registration systems basically enable a detailed gathering of labor hours for most activities.

But the allocation of activity costs to customers and software components respectively raises many difficulties. It is not clear how development and maintenance costs of core assets can be distributed to customers. The use of software components as cost objects is problematic when there is no direct relation of an activity to a certain component, e.g. activities like requirements engineering or system testing. Therefore we propose the use of features as the primary cost object for activity based costing in SPL development.

Feature modeling was established as a technique for modeling commonalities and variabilities in requirements engineering for SPLs [6][7]. Basically feature models present a hierarchical structuring of features and contain domain relations like alternative or optional features and furthermore dependencies between features.

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

The cost allocation from features to customers must be based on an assessment of the importance of the provided features to the different customers (Figure 1). In case of features that are shared between different customers, the costs have to be divided. In order to do this, the value of the feature for each customer must be assessed independently. The distribution of the costs can then be based on proportions of these assessment values.

Hence an integral part of this approach to variability management must be the analysis of the value a feature provides to customers. It might not be possible to express this value in monetary units, but it suffices to assess the contribution of a feature to user satisfaction

on a simple scale of values. Techniques like the Kano method [8] or Quality Function Deployment [9][10] may be applied to this purpose.

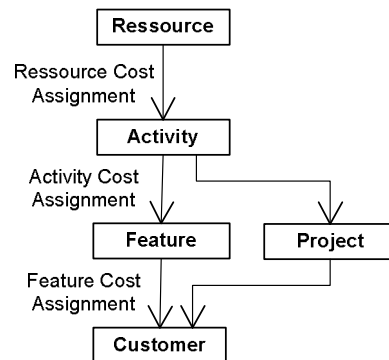


Figure 1: Features as cost objects

#### 4.2. Estimation of variability costs

Gathering experience with accurate cost information of features will enable an analysis of the relation between initial development costs of a feature and its follow-up costs.

A better understanding of the cost implications of variability will allow better estimations of future maintenance costs. This can guide decisions on the architecture of the product line, e.g. on the use of certain variability mechanisms or the merging of existing product variants or features. Furthermore cost estimations can be employed for quotation costing and negotiations during early requirements engineering phases.

#### 4.3. Guidance of scoping decisions

Bringing together cost information based on features as cost objects and the assessment of the customer value of features provide a sound guidance for scoping decisions (figure 2).

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

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

must be some estimation where and how much money is lost.

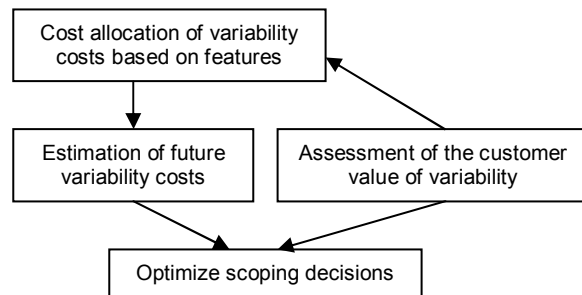


Figure 2: Approach to variability management

#### 4.4. Process improvement

Cost information will not only help to decide on the range of offered features and their pricing. It can also be utilized to identify weaknesses in the internal processes and guide respective improvements. As an example it might be identified that there is a high effort for testing certain features in each subsequent product release. This can justify investments in flexible testing technologies. Other examples might be a high effort for customer support or for bug fixes related to certain features.

#### 5. Further work

In our further work we aim at developing a detailed model for a costing approach based on features as cost objects. Within the context of an ongoing industry cooperation project we will conduct a case study on gathering cost information in an SPL development process, in order to validate if this approach can help to evaluate product management decisions and support cost estimations. To provide appropriate tool support, the collection of costing information should be integrated with our existing tool set for feature modeling [11].

#### 6. Summary

Managing a software product line imposes novel challenges to software product management due to multiple stakeholders with diverging needs, and complex dependencies between the products. The cognitive complexity of the variability of features leads to high coordination efforts, insufficient scoping decisions and a suboptimal exploration of the reuse potential. Difficulties in steering the increase of feature variants within the product line are evidenced by increasing maintenance costs.

This resembles the significant increase of overhead costs caused by the growth of product variants in manufacturing industry. Variability management was therefore recognized as a core task of product management. A key to gain control on product variability was the application of new costing systems to obtain more accurate cost information.

We believe that intransparent cost structures are a major barrier for successful management of software product lines. The presented approach to variability management for software product lines aims at closing this gap by utilizing an activity based costing approach to software development with features as primary cost objects. Features provide a natural basis for allocating costs and are anchored in existing variability modeling techniques. The gathering of more accurate cost information is combined with an assessment of the customer value of features within the product line. This approach can support better estimations of future costs of variability as a basis for customer negotiations and enable a strategic steering of the variability within the product line.

#### References

- [1] Schmid, K. (2003): Planning Software Reuse – A Disciplined Scoping Approach for Software Product Lines. PhD Theses in Experimental Software Engineering, vol 12, Fraunhofer IRB Verlag, Stuttgart.
- [2] Clements, P.C., J.D. McGregor, S.G. Cohen (2005): The Structured Intuitive Model for Product Line Economics (SIMPLE). CMU/SEI-2005-TR-003, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania.
- [3] Schuh, G. (2005): Produktkomplexität managen – Methoden, Strategien, Tools. (in German) Hanser Verlag, München.
- [4] Kaplan, R.S., R. Cooper (1997): Cost and Effect. Harvard Business School Press, Boston, Massachusetts.
- [5] Fichman, R., C. Kemerer (2002): Activity Based Costing for Component-based Software Development. Information Technology and Management, vol 3 (1/2), 137-160. Springer, Netherlands.
- [6] Kang K., S. Cohen, J. Hess, W. Novak, A. Peterson (1990): Feature-Oriented Domain Analysis (FODA) Feasibility Study. CMU/SEI-90-TR-021, ADA235785, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania.
- [7] Lee, K., K.C. Kang, J. Lee (2002): Concepts and Guidelines of Feature Modeling for Product Line Software Engineering. In Gacek C. (Ed.) Proceedings of the 7th International Conference on Software Reuse: Methods,

Techniques, and Tools, ICSR-7, Austin, Texas. 62-77. LNCS 2319, Springer, Berlin.

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

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

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

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

# Towards Context-Aware Product-Family Architectures

Mohammed Salifu    Bashar Nuseibeh    Lucia Rapanotti  
*The Open University, Milton Keynes, UK*  
{M.Salifu, B.Nuseibeh, L.Rapanotti}@open.ac.uk

## Abstract

*The product-family paradigm is predicated on the definition of a general product architecture from which a number of different architectures can be derived, each addressing the needs of a separate market segment, with a high level of component reuse. The way this is achieved is through the identification of variability points within a generic product architecture, from which variant product architectures are generated, each product with a specific range of functionality.*

*This position paper proposes an approach to variability for software product-families that extends traditional product-family variability mechanisms to deal with context awareness. The approach is based on the identification of relevant relationships between context properties and user requirements on the one hand, and architectural variability types on the other; and the definition of design notations and principles for the representation and trade-off analysis between different variability types within a product-family architecture. In broad terms, the approach links requirements to software architectures using the product-family paradigm as the underlying organising principle. This is envisaged to provide context-aware product-family system developers a conceptual means to capture requirements and in linking such requirements to architectural design choices in the form of variability point types. This will contribute to software systems design and management during development and subsequent evolution.*

## Keywords

Product-line architectures; variability points; static adaptability; dynamic adaptability

## 1. Introduction

The notion of product-line informally describes a situation in which the supplier of a product seeks to maximise profit by identifying and supplying different

categories of the product to different segments of consumers of the product [5, 8, 25]. The process of identifying or defining segments of consumers is referred to as market segmentation [8].

The product-line paradigm is predicated on the definition of a general product architecture from which a number of different architectures can be derived, each architecture addressing the needs of a separate market segment, with a high level of component reuse. In order for the generic architecture [30] to support different market segments [5, 8], variability points [31, 37] are identified in the architecture at which the differences are realized. The list of possible architectural elements (components, connectors, constraints on them) from which a variability point will ultimately be assigned value(s) are referred to as variants [17, 18, 35, 36]. When the decision as to which variant to use at a variability point is taken and all other variants previously available at this point are discarded, the point is said to be bound or closed [18]. (An unbound variability point is referred to as an open point.) The realisation of individual products is achieved by statically binding the variability points before they are delivered to consumers. We refer to this type of variability points as static and the process of realising the individual products as static adaptability.

On the other hand, context-aware devices are expected to change their behaviour in response to changes in their operating environments or contexts. Reasons for changes in the operating environment are many, from fluctuating resources upon which the device relies (e.g., reduced bandwidth for a mobile phone) to different operating locations (e.g., a mobile user travelling long distance) or the presence of other devices (e.g., Bluetooth-enabled phones) [4, 12, 29, 34]. Changes may also be caused by users' preferences; for example, users of a mobile phone may require a particular set of features to be available to them while at work and a different set of features while at home. Adaptation to a changing environment may well require the restructuring of a device's software

architecture as different bindings of components may be involved. We refer to the restructuring of an application's architecture in response to contextual changes while the application is in operation as dynamic adaptability and the binding or unbinding of architectural elements as dynamic binding or unbinding.

There is an increasing expectation for software intensive devices to be context-aware, and many consumers' devices, such as mobile phones, which are developed as product families, are expected to follow this trend. To overcome the limitation of static binding, there is a need for approaches to product-family design which allow for both type of variability to be dealt with at the architectural level, as this is considered to be more efficient and cost effective [28]. Appropriate design notations and analytical tools will be required to enable trade-offs between static and dynamic variability points depending on characteristic of operating contexts and their associated requirements. This would allow appropriate variants to be chosen at variability points depending on the context for which they are intended. We argue that the definition of an architectural notation and a reasoning framework for dealing with various types of variability points and their instantiation, using a problem-oriented approach (such as problem frames [14]), is a necessary prerequisite to the successful design of product-families of context-aware software devices.

To the best of our knowledge, a framework which deals with these issues has yet to be developed. Therefore, we propose an approach that extends traditional product-family variability mechanisms to deal with context awareness. The approach is based on: the identification of relevant relationships between context properties and user requirements on the one hand, and architectural variability types on the other; and the definition of design notations and principles for the representation and trade-off analysis of different variability types within a product-family architecture.

The rest of this paper looks at background and related work and briefly describes our proposed approach.

## 2. Background & Related Work

This section briefly discusses current classification and representation of variability points and dependency relations between variants. This will be followed by a discussion of current approaches to reconfiguring product families.

### 2.1 Variability points and dependencies

Bachmann and Bass [2] have identified six sources of variability points. These are variations in function or features, such a word-processor with a voice-recognition system (high-end) and one without (low-end); variation in the data used by components for communication, such as using a stack or queue and variation in control flow such as two components communicating using a publish-subscribe communication pattern [33] or a dedicated communication channel. Other sources of variabilities are variations in technology, variations in quality requirements and variations in the target operating platforms. Irrespective of the source of variability, variants available at variability points have been largely classified into three main types [2, 24, 37]: optional, single and multiple variants. An *optional* variant is included in some product members but not in others, for instance including a camera in a mobile phone. In the case of *single* variant, exactly one variant is chosen from a candidate set and all others discarded. In the case of *multiple* variants, more than one variant must be selected from a candidate list, for example dual communication protocols for a mobile phone, thereby giving it a wider area of operational access. There are various combinations of these basic types reported in [2]. This rigid classification is unsatisfactory in the presence of contextual changes as functionalities in a given set and for a single device may be classified as single variant (i.e. mutually exclusive) in one context and multiple variant (i.e. to be selected together) in another context.

Jaring and Bosch [17] have looked at the classification of variants and argued that whether a variant is optional or not changes based on relational dependencies. This is inline with earlier observation by [3] when they looked into using use cases [6] to communicate variability in product-family members to customers. In this context, a relational dependency refers to the changing of the type of a variant from optional to mandatory, or vice-versa, following the selection of another variant. That is, the selection of one variant may automatically trigger the inclusion of another variant which was previously classified as optional. For example, in the Nokia phone product-family, as currently implemented, the selection of the FM radio feature will automatically require the inclusion of the series 40 operating platform [26], as it is the only platform that supports this functionality. Therefore, at design time, it is necessary to consider all dependencies which derive from all contexts within which a product-family member is to operate. These include dependency relations between variants at a

given variability point and among variants at different variability points. Conflicts also need to be resolved.

The representation of variability points in software product-families, as currently practised, is achieved primarily through feature diagrams [16, 37]. A feature diagram shows the set of features available in the generic architecture from which the features of individual product members are selected. In this context, a feature is a functional unit that is visible to the consumer, such as a camera or FM radio in a mobile phone. The representation of variability point using use cases by Böhne et al. [3] is comparable to those based on feature diagrams. On the other hand, a graphical notation which is not based on feature has been proposed by Bachmann and Bass [2]. This is based on a generic UML (like) structure notation. What all these have in common is that, none of them explicitly consider the properties of the operating contexts and the domains existing in these contexts. We argue that the explicit representation of contextual properties and requirements, using a problem-oriented approach [14], is a prerequisite to successful development of context-aware product-family architectures.

In addition, all these notations are aimed at product-families whose architectures are statically bound and therefore made up of static variability points. Part of our work will seek to explore possible extensions for representing both static and dynamic variability points and their associated variants, using a problem-oriented approach.

## 2.2 Reconfigurable product families

There have been attempts [1, 10, 11, 21] at designing reconfigurable product-family architectures, each of which will be discussed next. The configuration of an architecture refers to its set of components, their interconnections and the constraints defining the behaviour of this architecture [13]. The replacement of such a configuration with a new (or different) one after it has been released or during the operation of the applications based on it, is referred to as reconfiguration [22, 28].

The work of Gomaa and Hussein [10, 11] is based on the use of architectural styles or patterns, such as the client server architectural style, to construct what they refer to as a reconfiguration pattern (based on the style of the generic architecture). A reconfiguration pattern is used to guide the process of automatically deriving one product-line member from a different one. This can be argued to be a generalised form of

parameterisation [30] as all instances of this product-family must conform to the style and different members are instantiated by changing the values of parameters. The focus of Gomaa and Hussein [10, 11] is on managing state transitions from one product member to a different member during reconfiguration, with no explicit discussion of the impact of dynamically changing the operating context on defining and reasoning about static and dynamic variability points. This approach fits the definition of static binding as the reconfiguration takes place as a one-off, and instantiated members do not undergo any dynamic adaptation of their structure during operation. It therefore appears not to be suitable for context-aware applications.

The work of Kim et al. [21] is similar to that of Gomaa and Hussein, hence exhibits the same limitations. The key difference is that Kim et al. provide an architectural description language for describing architectures and modifications to be applied to them during reconfiguration. The example in [21] adopts a pipe-and-filter architectural style and could therefore be argued to be a specialised case of the work of Gomaa and Hussein with the addition of a means to describe the architecture and its modifications. Again, as with Gomaa and Hussein, the focus of this approach is on managing the state transition from one product member to a different member during reconfiguration. Hence, it also fits the definition of static binding as members do not undergo any dynamic adaptation of their structure during operation. Therefore, this approach too appears not to be suitable for context-aware applications.

The work of Apel and Bohm [1] is based on the use of a layered architectural style to design a reconfigurable middleware for a context-aware environment. In this work, context-aware environment refers to an operating environment with network bandwidth fluctuations, connection interruptions, device mobility and resource-constrained devices. The services provided by this architecture are operating system-based and largely limited to the network infrastructure. In designing the reconfigurable middleware architecture, Apel et al. have adopted the product-line paradigm and produced a generic architecture from which specific architectures tailored to different environments are produced as and when the context requires, during runtime. Hence, this approach fits the definition of dynamic binding as the general architecture undergoes dynamic adaptation during operation. However, note that the whole generic architecture undergoes the dynamic adaptation and not the individual product members. This distinction is

significant as it implies that the entire generic architecture is loaded onto each target device and each adaptation represents a different product-family member satisfying the requirements of a different context. Therefore, this approach only applies to devices that are capable of running the entire generic architecture. To summarise, there is only one target context-aware generic architecture, loaded in all devices and the only notion of variance is in terms of differences in operating contexts.

We argue that this approach has severe limitations for developing consumer electronic devices, such as mobile phones, with which the success of the application of the product-family paradigm is largely associated. For example, using this approach to transform the current Nokia static product-family architecture into one that is context-aware, will require all the 1000 or so different handsets [23] to carry the same generic architecture (including its complete set of features). This is not realistic as different devices have different capabilities, such as memory size, screen size, processor speeds, operating platforms and hardware features. Hence, the architecture of every device must be optimised to take full advantage of its hardware resources. This explains why Nokia has at least five different platforms each with several versions [26] aimed at optimising the hardware resources of every device. It is also a requirement by the Open Mobile Alliance community-OMA (a consortium of mobile phone applications developers and operators, such as Nokia and Phillips) [27] that no single architecture is delivered to different devices with different hardware resources and features as this leads to some devices under utilising their resources while others are over burdened with software resources they never use. This requirement is stated as an architectural principle (#4) by the OMA as:

“It is not acceptable for OMA Architecture and Specifications to take a lowest common denominator approach that prevents use of advanced features in more capable devices and networks. At the same time, users of less capable devices should enjoy the best possible experience that those devices can provide...”[27]

We therefore argue the need to (partially) statically bind individual product-family members (containing only the specific required architecture) before they are delivered to consumers, and to dynamically bind these architectures during runtimes in response to changing contexts. Hence some variability points should be statically bound (e.g. defining the specific type of phone and architecture) while others dynamically

bound during operation in response to changes in context (i.e. adapting the specific architecture for the given context). The overall aim of our research is product-family architectures capable of supporting a mixture of both static and dynamic variability points.

### 3. Towards Context-Aware Product Family Architectures

In this section, we briefly describe the main objectives of our proposed research. These are:

1. to classify and represent variability point types and their application to context-aware devices;
2. to elicit, define, and represent relationships between context properties and requirements, and types of variability (points);
3. to develop conceptual tools for trade-off analysis in the choice of different types of variability points within a product family architecture;
4. to evaluate the approach through relevant exemplars and case studies.

#### *Classification of variability points and their application to context-aware devices*

This will be based on the exploration of existing work on variability points such as the works of Bachmann and Bass [2], Svahnberg et al [37], Bühne et al [3] and Jaring et al. [18], which are currently based on statically bound generic architectures, in view of possible extensions to cover dynamic variability points.

#### *Variability points representation*

This will require the exploration of current notations, such as that proposed in [2], used for static variability points representation, in view of a possible extension. Alternatively, a customisation of a UML structural notation may be adopted. The aim is a notation suitable for representing both static and dynamic variability points.

#### *Definition, capture and representation of relationships between context properties and requirements, and types of variability (points)*

To this end, we are proposing to adopt a problem-oriented approach based on Jackson’s problem frames [14] and their extensions (such as [32]). This would provide a conceptual framework for capturing context information and requirements, and relating them to architectural solution structures (in particular, types of variability points in the architecture). This is because of their underlying principle of separation between the problem domain (the context of a software solution and



the requirement) and the solution domain (where software resides) [15]. Key strengths of the approach could be argued to be its adoption of a multi-paradigm approach to problems and solutions descriptions [7] and a clear separation of indicative from optative descriptions; that is the separation of what is true of the development world with or without the presence of the software to be constructed and that of the desired behaviour wished upon the operating environment. The problem frames approach is preferred over other approaches such as goal-based [38] or scenarios [6], as these do not allow for an explicit representation of context and would therefore not be suitable in addressing the kind of contextual information we need to represent.

#### *Definition of conceptual tools for trade-off analysis*

We will explore current (generic) architectural techniques for trade-off analysis such as ATAM [20] or SAAM [19] for possible application to the analysis of variability points trade-offs.

#### *Evaluation of the approach through relevant exemplars and case studies*

To this end, we will elicit case studies found in the product-family literature such as those published as part of the “Engineering Software Architectures, Processes and Platforms for System-Families (ESAPS)” [9] European project (and where possible solicit industrial partner support) and use them both in the development and evaluation of our techniques.

## 4. Summary

Our research proposes an approach that extends traditional product-family variability mechanisms to deal with context awareness. As can be seen from the discussion so far, the proposed research touches three key research areas. These are (1) requirements and contextual properties impacting (or constraining) their satisfaction, (2) software architectures in the form product-families architectures and variability points mechanisms and (3) architecture level trade-off analysis. Due to space limitations, the related work considered is largely based on product-families and variability points, in terms of architectural design space. However, we argue that for a context-aware product-family development tool or conceptual framework to be useful, it must cover these three key areas.

As the research is in its early stage, it is impossible to say if the outcome will lead to an entirely new product-family development paradigm. However, we argue that at the very least, context-awareness will increase the

complexity of product-family based systems development. Even though the increased complexity may not be visible in all phases of software system management, it is likely to significantly affect the development and management of such systems during requirement specification through to implementation. It is these phases of system development that a reasoning framework, such as the one envisaged, will provide a means in dealing with the increased complexity.

## References

1. Apel, S. and K. Böhm. *Towards the Development of Ubiquitous Middleware Product Lines*. in *Software Engineering and Middleware: 4th International Workshop, SEM 2004*. 2005. Linz, Austria.
2. Bachmann, F. and L. Bass. *Managing Variability in Software Architectures*. in *SSR'01*. 2001. Toronto, Ontario, Canada: ACM Press.
3. Bühne, S., G. Halmans, and K. Pohl. *Modelling Dependencies between Variation Points in Use Case Diagrams*. in *Proceedings of the 9th International Workshop on Requirements Engineering Foundation for Software Quality (REFSQ)*. 2003. Klagenfurt, Austria.
4. Chen, G. and D. Kotz, *A Survey of Context-Aware Mobile Computing Research*, in *Technical Report TR2000-381*. 2000, Dartmouth Computer Science.
5. Claycamp, H.J. and W.F. Massy, *A Theory of Market Segmentation*. *Journal of Marketing Research*, 1968. **5**(4): p. 388-394.
6. Cockburn, A., *Writing effective use cases*. 1st ed. 2001, Longman, Upper Saddle River, NJ: Addison-Wesley. 1-304.
7. Cox, K., Jon G. Hall, and L. Rapanotti, *A Roadmap of Problem Frames research*, in *To appear: Journal of Information and Software Technology*. 2005.
8. Dickson, P.R. and J.L. Ginter, *Market Segmentation, Product Differentiation, and Marketing Strategy*. *Journal of Marketing*, 1987. **51**(2): p. 1-10.
9. ESAPS, *Overview*. 2002, Engineering Software Architectures, Processes and Platforms for System-Families: <http://www.esi.es/esaps/overview.html>.
10. Gomaa, H. and M. Hussein, *Dynamic Software Reconfiguration in Software Product Families*. Lecture Notes in Computer Science, 2004. **3014/2004**: p. 435 - 444.
11. Gomaa, H. and M. Hussein. *Software Reconfiguration Patterns for Dynamic Evolution of Software Architectures*. in *Proceedings of the Fourth Working IEEE/IFIP Conference on Software Architecture (WICSA'04)*. 2004: IEEE CNF.
12. Harter, n., et al., *The Anatomy of a Context-Aware Application*. *Wireless Networks*, 2002. **8**(2-3): p. 187 - 197.
13. Hoek, A.v.d. *Configurable Software Architecture in Support of Configuration Management and Software Deployment*. in *INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING*. 1999.

14. Jackson, M., *Problme Frames: Analyzing and structuring software development problems*. 1st ed. 2001b, New York, Oxoford: Addison-Wesley. 390.
15. Jackson, M. *The World and the Machine*. in *ICSE'95*. 1995. Seattle, Washington, USA: ACM.
16. Jaring, M. and J. Bosch. *Representing Variability in Software Product Lines: A Case Study*. in *SPLC2 2002, LNC2379*. 2002: Springer-Verlag Berlin Heidelberg.
17. Jaring, M. and J. Bosch. *A Taxonomy and Hierarchy of Variability Dependencies in Software Product Family Engineering*. in *Proc. of the 28th Annual International Computer Software and Applications Conference (COMPSAC'04)*. 2004: IEEE CNF.
18. Jaring, M., R.L. Krikharr, and J. Bosch, *Representing variabilities in a family of MRI scanners*. Software-Practice And Experiance, 2004. **34**: p. 69-100.
19. Kazman, R., et al. *SAAM: A Method for Analyzing the Properties of Software Architectures*. in *Software Engineering, 1994. Proceedings. ICSE-16., 16th International Conference*. 1994.
20. Kazman, R., Mark Klein, and Mario Barbacci. *The Architecture Tradeoff Analysis Method. in Engineering of Complex Computer Systems, 1998. ICECCS '98. Proceedings. Fourth IEEE International Conference*. 1998.
21. Kim, M., J. Jeong, and S. Park. *From Product Lines to Self-Managed Systems: An Architecture-Based Runtime Reconfiguration Framework*. in *Workshop on the Design and Evolution of Autonomic Application Software (DEAS 2005)*. 2005: ACM Press.
22. Kramer, J. and J. Magee, *The evolving philosophers problem: dynamic change management*. IEEE Transactions on Software Engineering, 1990. **16**(11).
23. Maccari, A. and A. Heie, *Managing infinite variability in mobile terminal software*. SOFTWARE—PRACTICE AND EXPERIENCE, 2005. **35**(6): p. 513–53.
24. Mannion, M., et al. *Representing Requirements on Generic Software in an Application Family Model*. in *ICSR*. 2000: Springer-Verlag.
25. Moorthy, K.S., *Market Segmentation, Self-Selection, and Product Line Design*. Marketing Science, 1984. **3**(4): p. 288-307.
26. Nokia, *Nokia Architecture - Platform Architectures*. 2005, Nokia - [http://europe.nokia.com/nokia/0\\_62611\\_00.html](http://europe.nokia.com/nokia/0_62611_00.html): Online. p. 1-2.
27. OMA, O.M.A., *OMA Architecture Principles VI.2*. 2004, Open Mobile Alliance - OMA Ltd. p. 1-10.
28. Oreizy, P., et al., *An architecture-based approach to self-adaptive software*. Intelligent Systems and Their Applications, IEEE [see also IEEE Intelligent Systems, 1999. **14**(3)].
29. Pascoe, J. *Adding Generic Contextual Capabilities to Wearable Computers*. in *Digest of Papers. Second International Symposium on...* 1998. Pittsburgh, PA, USA.
30. Perry, D.E. *Generic Architecture Descriptions for Product Lines*. in *Lecture Notes in Computer Science*. 1998: Springer Berlin / Heidelberg.
31. Pohl, K., G. Böckle, and F.J.v.d. Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*. 1 ed. 2005: Springer. 468.
32. Rapanotti, L., et al. *Architecture-driven Problem Decomposition*. in *Proceedings of the IEEE RE 04 conference*. 2004.
33. Rozanski, N. and E. Woods, *Software Systems Architecture*. 2005, London: Addison-Wesley Profession. 27-37.
34. Schilit, B.N., N. Adams, and R. Want. *Context-Aware Computing Applications*. in *Workshop on Mobile Computing Systems and Applications*. 1994.
35. Sinnema, M., S. Deelstra, and J. Bosch. *COVAMOF: A Framework for Modelling Variabilities in Software Product Families*. in *SPLC'04*. 2004: Springer-Verlag.
36. Sinnema, M., et al. *Modeling Dependencies in Product Families with COVAMOF*. in *Proceedings of the 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems (ECBS'06)*. 2006: IEEE CNF.
37. Svahnberg, M., J.v. Gulp, and J. Bosch, *A taxonomy of variability realization techniques*. Software: Practice and Experience, 2005. **35**(8): p. 705-754.
38. Van Lamsweerde, A., R. Darimont, and P. Massonet. *Goal-directed elaboration of requirements for a meeting scheduler: problems and lessons learnt*. in *Proceedings of the Second IEEE International Symposium on*. 1995.

# Ten Misconceptions about Product Software Release Management explained using Update Cost/Value Functions

Slinger Jansen, Sjaak Brinkkemper  
Information and Computing Sciences Institute  
Utrecht University  
Utrecht, the Netherlands  
{s.jansen, s.brinkkemper}@cs.uu.nl

## Abstract

*The decision for a young product software vendor to release a version of their product is dependent on different factors, such as development decisions (it feels right), sales decisions (the market needs it), and quality decisions (the product is stable). Customers of these products, however, are much more cost oriented when deciding whether to update their product or not, and will look mainly at the cost and value of an update. Product software vendors would gain tremendously if their release package planning method was supported by a similar cost/value overview. This paper presents cost/value functions for product software vendors to support their release package planning method. These cost/value functions are supported by ten misconceptions encountered in seven case studies of product software vendors that these vendors had to adjust during their lifetime. Finally, a number of cost saving opportunities are presented to enable quicker adoption of a release and thus shorten release times and customer feedback cycles.*

## 1 Introduction

Product software release planning has been characterised as a “wicked” [4] and “complex” [1] problem for which no perfect solution exists. One part of release planning, release package planning, is often underestimated due to its seemingly innocent and uncomplex nature. Product software vendors that do not have much experience in release planning often publish their release packages because a team of experts within the organization deems the release good-enough, which results into some releases that are hardly adopted by customers, whereas others are much more popular.

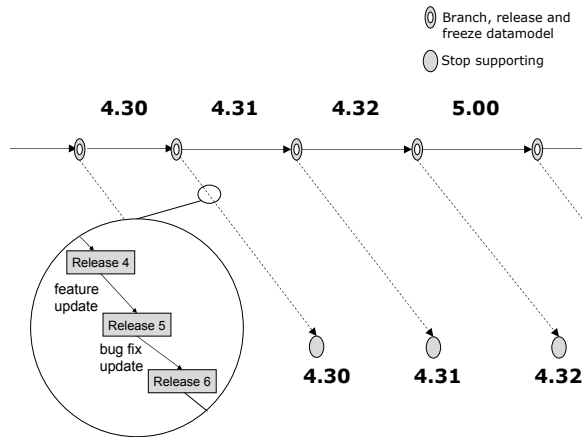
Simultaneously, release packages are created often during the lifecycle of a product, which suggests that processes such as release package creation, release package publication, informing the customer of a new release, and updating

are repetitious processes that must be automated as much as possible, to ease both customer update effort and vendor release package creation effort. Decreasing this effort results into customers that are more willing to update, and vendors who are more willing to release regularly, as suggested by the agile development methods, such as extreme programming [2]. However, from a number of case studies performed in the past it is found that product software vendors generally do not sufficiently plan their releases [10].

We define *Software product release management* as the storage, publication, identification, and packaging of the elements of a product. *Release package planning*, which is part of the release planning process, is the process of defining what features and bug fixes are included in a release package and the process of identifying these packages as bug fix, minor, or major updates, taking into account releases that have been published in the past and the possible update process required to go from one release of the product to another release. To illustrate, figure 1 displays a release snapshot from a recent case study [7], in which major, minor, feature, and bug fix releases are shown.

An *update package* is a package that promotes a customers configuration to a newer configuration. Secondly, a *bug fix update package* contains only bug fixes, a *feature update package* contains only new features, and *minor* and *major update packages* contain both bug fixes and new features. The distinction between minor and major update packages is usually that major update packages change structural parts of a product, such as the architecture or the data model. Our view of software evolution described here is similar to Rajlich and Bennet’s staged model [3], which addresses evolutionary changes (minor and major update packages) first, and then continues to see patches (bug fix packages) until a release is phased out and closed down.

The objective of this paper is to create release package planning awareness within software product management research. This is achieved by the presentation of cost/value functions that support misconceptions found in seven real-life cases about software product release management. Two



**Figure 1. Typical Versioning Example**

complementary cost/value functions are presented that enable a product software vendor to estimate whether a release package will actually be downloaded and installed by its customers. Also, two complementary Cost/Value functions are presented to help a software vendor decide whether the next release package will be marked a bug fix, minor, or major release.

The presented Cost/Value functions provide an extra check before publishing a release package for product software vendors. In that, the presented decision method is a useful extension to product road mapping methods, such as the one presented for small product software businesses [15], the method that supports the product software knowledge infrastructure [17], and other methods that support release planning [14]. Section 2 presents and describes the Cost/Value functions. Section 3 describes ten misconceptions encountered in seven case studies that support the Cost/Value functions as a valid release package planning method. In section 4 methods are described to save either customer update costs or vendor side release costs. Finally, in section 5 we discuss the presented method and describe the conclusions.

## 2 Defining the Cost/Value functions

This section describes the Cost/Value functions for both the customer (see figure 2) when updating its software and the vendor (see figure 3) when releasing a new version. These functions separately describe the cost of an update for a customer, the value of an update for a customer, the value of a new release for a vendor, and the cost to create the release package for the vendor. The functions are based on case studies performed at seven organisations [10]. Also,

the customer functions are based on different papers from Enterprise Resource Planning (ERP) application updates and migrations [13], and a recent case study we performed at a content management systems vendor that also does updates and migrations for customers [8]. The Cost/Value functions are similar to the profit functions developed by the Research Triangle Institute [16]. However, these profit functions are used to calculate the impact of software testing inadequacies to the software business and not specifically for update release timing.

### 2.1 Customer Functions

A customer will base its decision to update a software product on a number of factors. First and foremost, the customer is interested in the value the update represents for her. This value can be of many different forms, such as the addition of a new level to a game providing the customer with more entertainment or a complete new production planning module to an ERP package saving the customer many millions. Simultaneously the customer will take the cost of updating into account. Such cost can be the downright effort of downloading and installing the new level for the game or downtime of the ERP system during the update costing the company many millions.

A customer's value of an update is defined as  $Cval$  (1). The function defines the value of an update to a customer as the value of new features the customer will use plus the value of the removal of previous workarounds. The new features include those features that have been added to the new release, but also those that simply did not work in the previous release and for which a workaround was not available.

Before a customer decides to update, however, the customer will calculate the cost of an update to see if it is really worth it. The cost function  $Ccost$  (2) defines the cost of an update as the cost of downtime of a product, the cost of training for the people using the new/changed functionality, the cost of effort put into the update process, the cost of functionality that was removed from the release or the cost of customisations that can no longer be used after the update, and finally the cost of the payments to the vendor for the update.

For a customer to make the update decision,  $Cval$  must exceed  $Ccost$  (3), especially when taking into account that the resources that are required to perform the update normally perform other value adding tasks. It is quite surprising to see that many vendors do not invest structurally into reducing these costs for the customer, especially since in most of our case studies up to 70 percent of revenue was coming from existing service contracts and only 30 percent from new customers. Also, when a vendor sees that  $Ccost$  exceeds  $Cval$  for a large number of customers, releasing an

$$Cval(update) = value(newFeatures) + value(removalOfWorkarounds) \quad (1)$$

$$Ccost(update) = \begin{cases} cost(downtime) & + cost(training) & + cost(updateEffort) & + \\ cost(lostFunctionality) & + cost(paymentToVendor) & & \end{cases} \quad (2)$$

$$Cval(update) > Ccost(update) \quad (3)$$

**Figure 2. Customer Cost/Value Functions**

$$Vval(newUpdatePackage) = \begin{cases} newCustomers * priceNewRelease & + \\ oldCustomers * priceOfUpdate & + costReduction(support) \end{cases} \quad (4)$$

$$Vcost(newUpdatePackage) = \begin{cases} cost(development) & + cost(updateCurrentCustomers) & + \\ cost(increasedSupport) & + cost(marketing) & + \\ cost(deliveryToCustomers) & + cost(packageCreation) & \end{cases} \quad (5)$$

$$Vval(update) > Vcost(update) \quad (6)$$

**Figure 3. Vendor Cost/Value Functions**

update becomes essentially useless, unless the vendor hopes to attract a large number of new customers. This seems improbable though, since if current customers are not interested in the product, why would new ones be?

## 2.2 Vendor Functions

For a vendor the value of an update is much harder to calculate, especially because it involves estimating how many new customers are attracted with the new release and how many customers are actually prepared to update. A vendor's value of a new release are new customers attracted by the release that specifically targets a new market, the reduction in support calls due to a bug fix to a commercial operating system, or a customer that pays the vendor for an update of their ERP product.

The function for a vendor's value  $Vval$  (4) describes the value of a new release as the number of new customers times the price of a new release, the current customers who are prepared to update against reduced cost, and finally the cost reduction in support calls due to the fixes in the new release package. Calculating the  $Vval$  is hardest, mostly because it involves estimating the number of new customers and estimating how many customers are willing to update from any previous version, which might introduce different prices for the different updates. If the release package contains a large number of bug fixes there might be a cost reduction in support costs. However, if many new features have been introduced, this reduction might be cancelled out by the cost increase in support.

The  $Vcost$  (5) function is defined as the cost of development of the functionality and bug fixes for the new release package, the cost of updating the current customers, the cost of increased support questions relating to the new release, the cost of marketing, the cost of delivering the new

release package to customers, and finally the cost of packaging the release. The cost of updating current customers includes such things as update tools [11], renewing their licenses, and possible support questions that arise during the update process. The cost of marketing includes informing current customers of the new release, the marketing campaign, creating release notes, and maintaining the product's website. The cost of delivering the update to customers encompasses the creation of the delivery medium (CD, DVD, floppy, USB-stick, website, etc.), the assembling of all artifacts, the possible translations of the products language files, and completeness checking of the release.

The  $Vcost/Vval$  functions are used to evaluate whether it is time to create a release package. This is generally the case when  $Vcost$  exceeds  $Vval$  (6), i.e., when the potential value of releasing an update is higher than the cost that was required to create the update. Automation of the processes that make up release package creation and publication can potentially reduce  $Vcost$ , enabling a software vendor to release more often. This is similar to condition (3), where automation of the delivery and deployment processes can decrease  $Ccost$ , thus making it more attractive for customers to update.

The functions shown in this section tend to change largely when looking at either a bug fix, a minor, or a major release package. In the case of a bug fix package that is released on-line, contrary to a major release, no new storage media need to be created by a vendor. The decision to release either a bug fix, minor, or major release package can be made using these functions. If the reduction in support costs justifies the effort put into fixing a number of bugs, a bug fix release is justified. If the reduction in support costs does not justify the effort put into fixing a number of bugs and the addition of functionality, you might want to earn it back by making the next release a minor release. If the ven-

vendor feels that the next release should generate more revenue from new customers and old customers as well, this might be a justifiable case for a major release package. Of course this is not a hard science. Especially in the case a bug cost a disproportionate amount of time to fix, it might not be justifiable to publish a minor release package. A question the vendor must ask itself then is whether it was worth it to try and fix the bug in the first place.

### 3 Ten Misconceptions about Product Software Releasing

All product software vendors undergo series of paradigm shifts during their lifetime leading to radical changes in earlier established principles [6]. These misconceptions are generally strategic misconceptions that beginning software vendors can easily have about product software management and release management specifically. Here ten misconceptions are presented that were encountered in seven case studies of product software vendors. These product software vendors have been the subject of study from 2004 until 2006, and include Dutch software organizations with between 60 and 1500 employees [7] [8]. The main focus of research were the vendors' release, delivery, and deployment processes. For a further description on how the case studies were undertaken we refer to the case study reports and a paper describing all seven cases [10]. The value/cost formulas presented in this paper support the lessons learnt presented here.

**1. Customers want to stay up-to-date** - It is important to realize that a customer of a software product uses it only to make life better. If a newer release package does not provide the customer with new functions, why would she update? When, for instance, was the last time you updated a computer game? Or your ftp client? To quote one of the case study participant's customers "Their software supports our business process perfectly. Some of the workarounds are strange, but as long as we don't have to invest in the ghastly process of updating, we're happy." This is a clear example of where  $C_{cost}$  exceeds  $C_{val}$ .

**2. Customers must stay up-to-date** - To guarantee success of a product software vendor it is often assumed that customers must stay up to date. The misconception is demonstrated by the example of a content management system product software vendor, where customers use versions from years back who never updated due to the large number of customisations and complex update process. These customers, however, don't feel limited in their use of the product however, and will update when they require new functionality. Once again  $C_{cost}$  exceeds  $C_{val}$ . The difference between the first and second misconception is that they are discovered at different times in the product lifecycle. The first misconception is discovered once a new version of a

product is released and is not adopted at all by customers. The second misconception is discovered once a vendor has many different versions out in the field, without encountering life-threatening problems.

**3. Release  $n + 1$  is better for a customer than release  $n$**  - Many of a bookkeeping software vendor's customers were still using the MS-DOS based version of their product until 2005 when the vendor declared it would no longer support the DOS version. When attempting to update all these small entrepreneurs to a GUI based version the main complaint was that the graphic interface was *less* intuitive than their previous DOS versions. The bookkeeping software vendor ended up implementing all the same keystroke combinations that were typical of the DOS era, into their GUI based client. Even though from the vendor's point of view their update to the GUI based version was necessary, customers could have worked with the DOS version for at least the next ten years and considered  $C_{cost}$  to be larger than  $C_{val}$ .

**4. Fixes can be postponed to the next major release** - A typical mistake to make is to postpone bug fixes for later releases, hoping to save the effort of having to implement the fix into multiple releases. This works fine if customers are eager to update, and the next major release is around the corner. However, in one of the case studies performed in 2004 we encountered a vendor who postponed many bug fixes to its next major release package. The major release package, planned for early 2005, still has not been released mid 2006. Many of the bug fixes had to be back ported to keep customers satisfied. This is a clear example where  $V_{cost}$  seemed to be lower than  $V_{val}$ , but actually was not.

**5. Workarounds must be avoided at all costs** - Once again, as long as  $C_{cost}$  exceeds  $C_{val}$ , workarounds are a nice solution to a problem that would otherwise require a large investment from an organisation or person. An example of this is the Internet Explorer workarounds for style sheets. Quite often style sheets will look different on Internet Explorer 6 than other browsers, due to a bug. It is common knowledge, however, that Internet Explorer's interpreter can be fooled by adding specific characters to the code of a style sheet. Microsoft has chosen not to fix this bug until Internet Explorer 7, mainly due to the fact that everyone is aware of the workaround and too many customers would need to be updated.

**6. Customers always want new features** - This common misconception is that any release package can contain new features, since the customer should be happy with (possibly) free new features. An example encountered is a point of sale product software vendor, whose users typed more or less blindly into the system and checked only every ten seconds to see if the screen was showing the desired result. The simple displacing of a button in the user interface raised so many complaints ( $C_{cost}$  exceeds  $C_{val}$ ) that they decided

to freeze the user interface to their application in between minor releases as much as possible.

**7. Releasing too often is bad** - The aforementioned bookkeeping product software vendor started releasing on a weekly basis at some point, to shorten the feedback cycle to developers. The vendor did receive more bug reports, but product experience in general, declined. The vendor decided that this was not caused by the fact that they released too often, but that they released to their final customers too often. The frequent releases were maintained, but only for internal use, quality assurance, and pilot customers. Also, customers are required to stay up to date to reduce the number of support calls.

**8. A quiet customer is a happy customer** - An informal survey amongst a number of customers of a plug-in software vendor showed that customers who contacted the helpdesk in the early phases of its use were much more content with the product than those customers who had not called the helpdesk in the early adoption phases. Another example encountered was a software vendor who called up a customer for a yearly check-up, and heard that they had recently decided to buy a competitors product, even while the customer still had a contract with the current vendor. This demonstrates the importance of regular customer contact. The customer would still have been a customer if the vendor had made the customer aware of the fact that *Ccost* is smaller than *Cval*.

**9. Customers read release notes** - Especially system managers of large software products are well accustomed to browsing through release notes, trying to find that one fix to a bug or that one new feature that justifies a customer's investment into updating the product. Clearly, this is a proactive customer that is looking to optimize the value of the software product's latest release package. These system managers, however, would be much more interested in information about new releases that specifically targets them. One software vendor [12] is currently experimenting with a system that filters release notes for specific customers, such that they do not receive information that is irrelevant to them. An example of this is a bug fix to a component a customer has not purchased.

**10. Having many different releases out in the field is bad** - The earlier example of the content management systems product software vendor shows us that having many different releases out in the field is not necessarily a bad thing, as long as it is part of the business model. This vendor, for instance, charges its customer for all services in the form of a service contract, especially to those customers with very old versions. To the software vendor these customers present more of a knowledge management problem, since many of the solutions built in the past have to be reused for customers experiencing similar problems now. The vendor does agree that this is only possible due

to its small "manageable" number of customers. The difference between the second and this misconception is that the second misconception addresses the "happy" customer, whereas this misconception concerns the successful product software vendor.

Some other misconceptions encountered were "our next release must contain less bugs than our previous release to satisfy customers" and "we shouldn't build an automatic updater because the customer will feel they're not in control". These misconceptions are proven wrong by our Cost/Value functions as well, but we simply encountered them less often than the ten mentioned here. It is our firm belief that taking the profitability approach with regards to release package planning in a commercial environment is the way to go. An interesting question of validity is whether this type of anecdotal evidence is enough to prove that our Cost/Value functions are correct. It is part of our future work to further evaluate the validity of the Cost/Value functions based on historical (cash flow) results from both software implementations at customers and software release history.

## 4 Reducing Costs of Release Management

Besides using the Cost/Value functions for daily decisions, they allow us some thought experiments. Product software vendors generally adhere to bug fix/minor/major release scoping. When looking solely at version numbers, an open source project such as Mambo/Joomla, has had three major releases since 2001, approximately 10 minor releases, and approximately 120 bug fix releases. These numbers show that bug fix updates are released much more often than major updates. Also, when looking at customer behaviour, they are more inclined to regularly update to a new bug fix release package than they will perform a costly major update.

When looking at bug fix updates and the functions presented earlier the Cost/Value calculation impact factors change compared to major updates. In the case of a major update, the cost of development will largely exceed all other costs, making those less important from a financial point of view. For a major release, for instance, the completeness checking of artifacts will be a relatively small step in the release package creation project. When looking at a bug fix project, however, the development might have taken only a couple of days developing effort, whereas the creation of the release package might take an equal amount of time and effort. If we then take into account that these bug fix releases generally do not generate profit and only improve product quality and reduce the number of support calls, other costs are suddenly much more drastic.

Besides the scope of a release, the number of customers who update to a new release determines how much effort must be put into reducing the cost of release management.

For Exact Software and its 160,000 customers [9], for instance, the reduction in cost by introducing a combined software configuration management system and customer relationship system was huge. By combining these two systems they enable customers to automatically download and deploy bug fix and minor updates. However, if a vendor only serves twenty customers and is not planning to extend their customer base beyond one hundred customers, it must consider whether it is worth investing much into automatically releasing, delivering, and updating releases at its customers.

A product software vendor can reduce its costs in a number of areas. This cost reduction in turn enables a vendor to release more often. Releasing more often generates feedback about new releases quicker, which enables a vendor to improve its product and make better informed decisions on development and fixing plans. Clearly, this theory supports the agile camp, in its “Release early and often” viewpoint [2].

#### 4.1 Vendor Side Cost Reduction

To begin with a vendor must strive to **release often**, if not continuously [18]. The more a product under development is in the shape it will be in when finally released, the less chance there is for errors to be introduced during release package creation. After all, any party within the vendor organization, be it pilot customers, other developers, or the quality assurance department, will use this latest release for internal evaluation. The parties responsible for the final release will also have less work in the final stages of release package creation, a process that takes place often. This process is hampered by a product that supports different languages, since quite often these language files are translated shortly before the final release date.

The process of **release package creation must be automated** as much as possible to eliminate simple (error sensitive) manual tasks. If a release package is checked for completeness automatically each time a release package is created, it does not need to be checked extensively by quality assurance, eliminating a large part of this process.

The cost of software delivery is greatly minimized if **all delivery is done through a network** instead of expensive media, such as CDs or DVDs. The releases stored on these media are never as up-to-date as the ones stored in the vendor’s release package repository, which could be accessible through a network or secure Internet connection.

#### 4.2 Customer Side Cost Reduction

Whereas the vendor might be reducing costs internally, it must invest in making the deal to update to a new version as attractive as possible. Though this seems like a large investment at first, the payoff comes quickly when customers

become more eager and better informed with regards to releases a vendor offers.

Software deployment costs can be reduced for the customer by **automating the update process**. This requires the software vendor to seriously invest into an update tool and to develop its architecture in such a way that customisations remain functional after an update. Even though this seems like a large investment up front, it makes the decision for a customer to update easier, and as such makes them more eager to update often. The same holds for the reduction of downtime, since customers will be much more eager to update if downtime is reduced to a minimum.

Before customers can update to a new release, however, they need to be informed about the new release package. Currently, most product software vendors inform their customers through information news letters, customer days, e-mails, and many other ways. A higher rate of release penetration can be reached, however, if the vendor **uses the software itself to inform the customer**. This can range from a small pop-up when the application starts up, to an automatic pull of an update, such as Mozilla’s Firefox currently does.

With regards to informing customers, release notes are an essential part of release management. When customers are looking for a bug fix, for instance, they will browse through the release notes looking for that specific piece of information. Clearly these **release notes need to be indexable**, such that customers who previously requested information concerning a problem are informed as soon as a fix for that problem has become available.

### 5 Discussion and Conclusions

This paper presents cost and value functions that product software vendors can use to evaluate whether it is profitable to release a version of their software. Simultaneously, functions are provided that assist a customer in making the decision to update a vendor’s software product. These functions support ten changed viewpoints that were encountered in seven case studies. Finally, these functions show that costs can be saved for both product software vendors and customers on commonly occurring patch and minor updates, which can shorten feedback cycles from end-user to product software developer.

The process of release package planning is greatly simplified with the use of the provided Cost/Value functions. These functions also defend that product software vendors invest into automating processes such as release package creation, release package publication, informing the customer of a new release, and updating. The fact that this does not happen in practice raises a number of questions, such as why the vendors do not invest more into these processes. An answer often given when product software vendors were confronted with this question was that they are



busy creating their specific software solution already, but that they would be happy to buy a tool that helps automating these tasks.

A weakness of the Cost/Value functions is that being obsessed with short-term profits will lead any product software vendor without a long-term vision to the abyss. Vendors must take into account customers will always be prepared to offer large amounts of money to small vendors if they just build one little feature that is extremely valuable to them. The vendor must always keep in mind that it is creating software for a market and not one particular customer. The functions must only be used once the prioritization of requirements for the next couple of releases has been finalized.

These calculations provide a decision method for updating and releasing, but only in case all costs and prognoses are exact. Knowing that this is impossible, we leave it to the practitioner to perform data gathering [5] and implement a risk factor for unforeseen costs (and unforeseen value). Currently the Cost/Value functions are still in an experimental state even though we feel they are of great value to the field of release package planning. Thus it belongs to our future work to evaluate the functions in real world scenarios with historical release and cash flow data. We do recommend using a currency as the unit of measurement, since both sales and full time employment units can be expressed in money.

Part of the work thus is to find methods and tools that assist product software vendors in automating the tasks of release creation, release publication, informing the customer of a new release, and updating a customer's configuration. In earlier work the lack of tools for software deployment was identified [11] and possible solutions were presented. With respect to continuous software releasing the tool Sisyphus was built to support product software vendors with automatically creating their software releases [18]. Work recently has started on the PHEME prototype, a communication infrastructure that assists product software vendors in sharing software, data, feedback, licenses, and commercial information with its customers.

## References

- [1] A. J. Bagnall, V. J. Rayward-Smith, and J. M. Whittle. The next release problem. In *Information and Software Technology*, volume 43, pages 883–890, 2001.
- [2] K. Beck and M. Fowler. *Planning Extreme Programming*. Addison-Wesley, 2001.
- [3] K. Bennet and V. Rajlic. A staged model for the software lifecycle. In *IEEE Computer*, July, 2000.
- [4] P. Carlshamre. Release planning in market-driven software product development: Provoking an understanding. Springer-Verlag, 2002.
- [5] C. Ebert, R. Dumke, M. Bundschuh, A. Schmietendorf, and R. Dumke. Chapter 1. In *Best Practices in Software Measurement*, 2004.
- [6] I. Heitlager, S. Jansen, S. Brinkkemper, and R. Helms. Understanding the dynamics of product software development using the concept of co-evolution. In *Second International IEEE Workshop on Software Evolvability (at the International Conference on Software Maintenance)*. IEEE, 2006.
- [7] S. Jansen. Software Release and Deployment at Planon: a case study report. In *Technical Report SEN-E0504*. CWI, 2005.
- [8] S. Jansen. Software release and deployment at a content management systems vendor: a case study report. Institute of Computing and Information Sciences, Utrecht University, Technical report UU-CS-2006-0XX., 2006.
- [9] S. Jansen, G. Ballintijn, S. Brinkkemper, and A. van Nieuwland. Integrated development and maintenance for the release, delivery, deployment, and customization of product software: a case study in mass-market erp software. In *Journal of Software Maintenance and Evolution: Research and Practice*, volume 18, pages 133–151. John Wiley & Sons, Ltd., 2006.
- [10] S. Jansen and S. Brinkkemper. Definition and validation of the key process areas of release, delivery and deployment of product software vendors: turning the ugly duckling into a swan. In *proceedings of the International Conference on Software Maintenance (ICSM2006, Research track)*, September 2006.
- [11] S. Jansen, S. Brinkkemper, and G. Ballintijn. A process framework and typology for software product updaters. In *Ninth European Conference on Software Maintenance and Reengineering*, pages 265–274. IEEE, 2005.
- [12] S. Jansen and W. Rijsemus. Balancing total cost of ownership and cost of maintenance within a software supply network. In *proceedings of the IEEE International Conference on Software Maintenance (ICSM2006, Industrial track)*, Philadelphia, PA, USA, September, 2006, 2006.
- [13] C. S. P. Ng, G. G. Gable, and T. Chan. An erp maintenance model. In *36th Hawaii International Conference on System Sciences (HICSS)*, 2003.
- [14] J. N. och Dag, V. Gervasi, S. Brinkkemper, and B. Regnell. A linguistic-engineering approach to large-scale requirements management. *IEEE Software*, 22(1):32–39, 2005.
- [15] K. Rautiainen, C. Lassenius, J. Vahaniitty, M. Pyhajarvi, and J. Vanhanen. A tentative framework for managing software product development in small companies. In *Proceedings of the 35th Hawaii International Conference on System Sciences*, 2002.
- [16] Research Triangle Institute. The economic impacts of inadequate infrastructure for software testing. National Institute of Standards and Technology, 2002.
- [17] I. van de Weerd, S. Brinkkemper, R. Nieuwenhuis, J. Versendaal, and L. Bijlsma. Towards a reference framework for software product management. In *Proceedings of the 14th International Requirements Engineering Conference (Accepted for publication)*, 2006.
- [18] T. van der Storm. Continuous release and upgrade of component-based software. In *Proceedings of the 12th International Workshop on Software Configuration Management (SCM-12)*, 2005.

# Towards Comprehensive Release Planning for Software Product Lines

Muhammad Irfan Ullah and Guenther Ruhe

*Laboratory for Software Engineering Decision Support, University of Calgary, Canada*  
*{ullah, ruhe}@cpsc.ucalgary.ca*

## Abstract

*Release Planning (RP) plays an important role for the success of incremental product development. Proper planning includes consideration of stakeholder preferences, resources and their capacities, as well as product and business objectives. The complexity of this process is getting even larger when looking for releases of software product lines (SPL). SPL is considered as a viable and important software development paradigm allowing companies under certain conditions to realize order-of-magnitude improvements in time to market, cost, productivity, quality, and other business drivers.*

*In this paper, we present ongoing research in the process to develop a comprehensive and formalized model for planning and optimizing releases for SPL. We have identified key issues that are unique to RP of SPL. Some of them are highlighted by an example modified from literature for illustrative purposes.*

## 1. Introduction

A software product line (SPL) is a set of software-intensive systems that share a common, managed set of features satisfying 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 [1]. A software release is a collection of new and/or changed features that form a new product.

Release planning (RP) for incremental software development assigns features to releases such that most important technical, resource, risk and budget constraints are met. Without good release planning ‘critical’ features are jammed into the release late in the cycle without removing features or adjusting dates. This might result in unsatisfied customers, time and budget overruns, and a loss in market share [2].

Release planning for software product lines (RP-PL) involves a number of new aspects, which are not applicable in traditional systems. It has to consider

multiple products being developed by multiple teams for different customers. The research question is to go beyond planning and managing just one, but a couple of different products integrated into a product family.

Currently, there is no RP-PL model that allows addressing the problem in a rigorous formalized manner generating optimal solutions or at least qualified solutions. Most of the popular SPL development methodologies focus on the scoping process. Riebisch et. al. [3] include release planning as part of the scoping process for SPL. They have proposed Decision Tables to prioritize features for assignment to releases. Their approach is semi-formal and does not support quantitative evaluation of release plans. Release Matrix approach proposed by Tabora [4] provides a holistic view of SPL release management however, lacks the formalism needed to generate optimal solutions. For the case of one product, the hybrid approach called EVOLVE\* was designed for exactly that purpose. It improves existing methods for release planning by combining the strength of mathematical models with the subtleness of experts’ knowledge [5]. It uses computationally intelligent techniques such as evolutionary computing and principles of multi-criteria decision support in combination with human intelligence [5]. The challenge is to extend this approach to RP-PL.

In this paper, we present ongoing research in the process to develop a comprehensive and formalized model for planning and optimizing releases for SPL. The paper is subdivided into five sections. In the following section 2, we present some terminology and existing results in the context of RP-PL. Section 3 is devoted to discuss the formal model of [5] and mapping of RP-PL problem to reuse existing functionality of the ReleasePlanner™ decision support system [6] developed for optimized release planning of one product. An illustrative example for the problem statement and its solutions is given in Section 4. Finally, Section 5 gives an outlook to future research.

## 2. State of the Art

### 2.1. One Product Release Planning

One of the most prominent issues involved in incremental software development is to decide upon the most promising software release plans while taking into account diverse qualitative and quantitative project data. This is called release planning. Release planning considers stakeholder priorities and different types of constraints. The output of the release planning process is a set of candidate assignments of features to releases. They are supposed to represent a good balance between stakeholder priorities and the shortage of resources. In each release, all the features are executed following one of the existing software development paradigms including analysis, system design, detailed design, implementation, component testing, system testing, and user testing.

An overview of existing methods for release planning of one product was given in [7]. The following deficits were analyzed:

1. There is no major focus on addressing system constraints. The attempt in [8] assumes operational risk of system failure can be given probabilistic values, without deriving such information from the architecture, code base, and other historical data of the system.
2. There are not enough decision support tools that are fully developed and based on sound formalism, except ReleasePlanner™.
3. Release planning has been largely focused on “fixed release intervals” and no current work on planning with flexible time intervals.

### 2.2. Scoping vs. Release Planning for SPL

What is the difference between RP-PL and the scoping activity being an established part of any PL development approach? In general, release planning prioritizes features for assignment to releases while scoping focuses on identification and principal categorization of features. In the strict sense [9], scoping identifies the individual products, features assigned to products and features assigned to core assets of all products. RP deals with prioritizing the features (common and variable) and assigning them to releases. Scoping develops the product portfolio [10] without considering the technological constraints, resource capacities and consumptions. Because of the inherent complexity, we suggest to make this process more rigorous and optimized. Based on the output of scoping process, this will allow generating qualified

alternative release plans in a transparent and repeatable manner.

### 2.3. Release Planning – Conventional Systems vs. Software Product Lines

RP-PL presents some new challenges, which do not exist in the context of conventional systems. We provide a list of new questions. For some of them, our proposed approach will show how to handle them in a more formal manner.

1. Managing delays in development of core assets due to conflicting requirements of products so that it does not affect product development.
2. Evolving core assets with competing requirements of various products.
3. Synchronizing different product cycles.
4. Allocating resources across product teams.
5. Improving quality, productivity and time-to-market goals while maximizing stakeholders’ satisfaction.
6. Moving excess resources across product development teams to increase resource utilization.
7. Quickly reacting to opportunities for new product development based on existing core assets.

At this stage our proposed RP-PL model can handle points 1 and 2 mentioned above with the help of what-if analysis. For point 3, our model is based on the assumption that all the products have same release dates. For point 4, our model develops the release plans with an assumption that only one team is developing all the products. Our model uses stakeholder preferences in a quantitative manner to develop the release plans thus it fulfills point 5. Since we are considering only one team therefore our RP-PL model does not address point 6. Our model can handle any number of products and therefore it is possible to introduce new products and generate release plans for them based on the existing core assets. Consequently, it satisfies point 7.

## 3. Modeling and Problem Statement

Ruhe and Ngo-The [5] have presented a hybrid release planning model for conventional product development. It is able to produce optimal release plan solutions by combining the strength of computational intelligence and the knowledge and experience of human experts [5]. In this section we will present the formal notation used by the model [5] and later map the release planning problem for software product lines

on it. In a SPL there are two types of features, i.e. core asset features ( $F_n$ ) and product specific features ( $F_m$ ). In our model we are assuming that these two types of features are already decided during the scoping process and core asset features are used by all the products while each product specific feature is required by only one of the products. Therefore, the total set  $F$  of features in the SPL is given as  $F = F_n + F_m = \{f(1), f(2), \dots, f(n+m)\}$ . These features can be assigned to one of the  $K$  possible release options. This is described by decision variables  $\{x(1), x(2), \dots, x(n+m)\}$  where  $x(i) = k$  if we assign feature  $i$  to release option  $k \in \{1, 2, \dots, K\}$  and  $x(i) = 0$  if feature  $i$  is postponed.

The model considers two types of dependencies amongst features: coupling and precedence. Coupled features must be released together while the precedence relationship imposes to release features in a certain order [5].

We assume an available resource capacity  $Cap(k)$  for each release  $k$ . For simplicity we only consider a single type of resource although the model [5] considers capacities for different types of resources. If feature  $f(i)$  requires an amount  $r(i)$  of development resources then each release plan  $x$  must satisfy

$$\sum_{x(i)=k} r(i) \leq Cap(k) \quad (1)$$

Stakeholders are extremely important in release planning. Assume a set of stakeholder  $S = \{S(1), \dots, S(q)\}$ . Each stakeholder  $p$  can be assigned relative importance  $\lambda(p) \in \{1, \dots, 9\}$ .  $S(p) = 1$  indicates the lowest and  $S(p) = 9$  the highest degree of stakeholder importance. Each stakeholder prioritizes every feature based on two criteria using a nine point scale. The first criterion is represented by  $value(p, i)$  and the second criterion is represented by  $urgency(p, i)$ .

The objective is to maximize a function  $F(x)$  among all release plans  $x$  subject to satisfaction of resource constraints described in equation 1. The function  $F(x)$  depends on a number of factors like value of the release, importance of stakeholders, urgency of a feature and its value to stakeholders.

$$F(x) = \sum_{k=1 \dots K} \sum_{i: x(i)=k} WAS(i, k) \quad (2)$$

with

$$WAS(i, k) = \xi(k) [\sum_p \lambda(p) \cdot value(p, i) \cdot urgency(p, i)] \quad (3)$$

In (3),  $\xi(k)$  expresses the relative (normalized to 1) importance of release  $k$ . This release planning model is the core functionality of the decision support system ReleasePlanner™ [6]. The goal of our research is to extend its applicability to allow also release planning

for SPL. As a first step in this direction, we have mapped RP problem for SPL on [5] and used ReleasePlanner™ [6] to generate optimal release plans. For that, we have made the following assumptions.

1. Core asset and product specific features along with their mapping on the products are available from the scoping process.
2. All the products in the SPL have the same release dates for respective releases.
3. Only one team is developing all the products in SPL.
4. No customization effort is required to use a core asset in a product.

#### 4. Hypothetical Case Study

We have developed an example by looking at a number of case studies presented in literature and adapted it to highlight some issues relevant to RP. The purpose of this example is to illustrate the mapping of RP problem for SPL on [5] and generate optimal release plans using ReleasePlanner™ [6].

Consider a SPL with two products, Product A and Product B. There are ten candidate features in the product line. The assignment of features to products and core assets (C.A.) is shown in figure 1. The graphical representation shown in figure 1 is adapted from [11] but here we are only interested in precedence and coupling types of interdependencies.

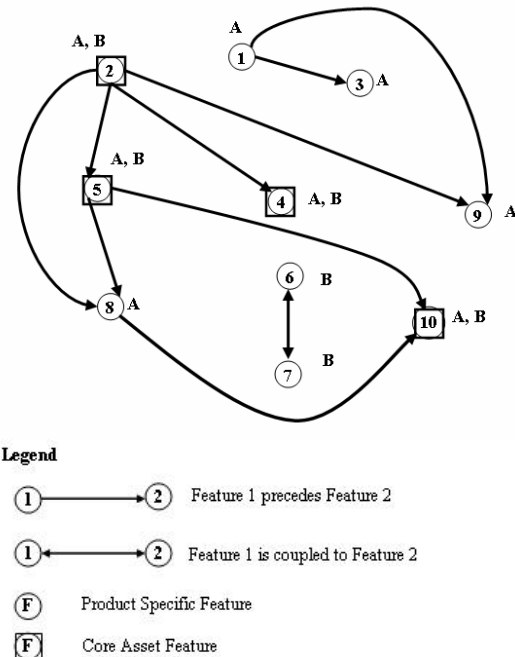


Figure 1: Graphical view of features in the SPL

In this example we are considering  $K = 2$  releases. There is one type of development resource and its capacity is  $Cap(1) = 55$  person days and  $Cap(2) = 45$  person days for release 1 and 2 respectively. The relative importance of releases are  $\xi(1) = 0.7$  and  $\xi(2) = 0.3$ , respectively. The coupling and precedence constraints amongst the features are illustrated in figure 1. There are two stakeholders  $S(1)$  and  $S(2)$  with relative weights  $\lambda(1) = 8$  and  $\lambda(2) = 5$ . The resource requirements for the ten features are given in table 1. Stakeholders' votes on all of the features for value and urgency criteria are also shown in table 1.

Table 1: Effort estimates and stakeholders' votes

Feature $f(i)$	Effort $r(i)$ (p. days)	Stakeholder $S(1)$		Stakeholder $S(2)$	
		Value(1, $i$ )	Urgency(1, $i$ )	Value(2, $i$ )	Urgency(2, $i$ )
f(1)	5	6	7	4	2
f(2)	14	5	8	7	6
f(3)	7	6	7	5	6
f(4)	16	4	2	6	7
f(5)	15	7	6	6	4
f(6)	10	6	7	6	3
f(7)	5	5	7	7	7
f(8)	8	6	6	5	8
f(9)	13	3	1	4	3
f(10)	10	5	3	7	7

ReleasePlanner™ presents the decision maker with up to five qualified alternative solutions. The term “qualified” here refers to solutions proven to be at least 95% optimal for the stated objective function. We have selected two solutions provided by the ReleasePlanner™ to discuss them in more detail.

#### 4.1. Solution 1

Table 2 presents the structure of solution 1 generated by the ReleasePlanner™. It lists down the features and the releases in which they will be developed. A core asset feature will only need to be developed once for all the products. In this release plan feature f(3) of Product A is postponed (PP).

Table 2: Release plan for SPL: Solution 1

Product A			Product B	
R1	R2	PP	R1	R2
f(2)	f(1)	f(3)	f(2)	f(4)
f(5)	f(4)		f(5)	f(10)
f(8)	f(9)		f(6)	
	f(10)		f(7)	

#### 4.2. Solution 2

Table 3 presents structure of the second alternate solution provided by the ReleasePlanner™.

Table 3: Release plan for SPL: Solution 2

Product A			Product B		
R1	R2	PP	R1	R2	PP
f(2)	f(1)	f(10)	f(2)	f(4)	f(10)
f(5)	f(3)		f(5)		
f(8)	f(4)		f(6)		
	f(9)		f(7)		

#### 4.3. Analysis

Both of the release plans presented above are within 97% quality range with respect to objective function, however, they differ in their structure. One of the major differences is in terms of postponed features. In solution 1, f(3) is postponed while in solution 2, f(10) is postponed. Postponement of f(10) affects both the products since it is a core asset. ReleasePlanner™ also provides comparison of the solution alternatives in terms of resource utilization, stakeholders' satisfaction and performance with respect to prioritization criteria. Based on the requirements of the product, their respective importance, time-to-market and stakeholders, the project manager can then select one of the solution alternatives.

The strength of our technique is based on its ability to combine human knowledge and expertise with computational power to generate different scenarios for supporting pro-active release planning. It also supports generation of what-if scenarios by varying resource capacities, release weights, stakeholder weights or by pre-assigning features to specific releases. One can imagine the complexity of this task when there are tens of products in the product line with hundreds of features to assign to releases while satisfying the constraints. On top of this, the release plan has to maximize the satisfaction of stakeholders' of multiple products having conflicting interests. With the help of ReleasePlanner™ decision support system, release planning of a product line with any number of products and features can be performed efficiently.

#### 5. Work-in-Progress

Some of the assumptions mentioned in section 3, under which we have mapped the RP-PL problem on [5] are restrictive and do not necessarily reflect most of the real world situations. We are currently working to

removing these assumptions by extending [5]. Extensions will include:

1. Allowing stakeholders to vote on developing a feature as core asset or product specific.
2. Ability to generate release plans with multiple teams, each developing one product of SPL.
3. Ability to move excess resources across product and core assets development teams to improve quality of release plan solutions.
4. Assigning relative weights to products in the product line based on their time-to-market requirements, economic benefits and stakeholders, etc.
5. Evaluate impact of different organizational structures i.e. separate core assets team vs. product teams developing core assets on release plan solutions.

## Acknowledgements

We thank the Alberta Informatics Circle of Research Excellence (iCORE) and the Higher Education Commission of Pakistan (HEC) for their financial support of this research. We also thank the anonymous reviewers for their detailed and valuable comments.

## References

- [1] <http://www.sei.cmu.edu/productlines>.
- [2] D. Penny, "An Estimation-Based Management Framework for Enhance Maintenance in Commercial Software Products," In Proceedings of the *International Conference on Software Maintenance*, Montreal, Canada, 2002, pp. 122-130.
- [3] M. Riebisch, D. Streitferdt and I. Philippow, "Feature Scoping for Product Lines," in Proceeding of *PLEES'03, International Workshop on Product Line Engineering The Early Steps: Planning Modeling and Managing*, Erfurt, Germany, September 2003.
- [4] L. Taborda, "Generalized Release Planning for Product Line Architectures," in Proceedings of the *SPLC 2004, The Third Software Product Lines Conference*, Boston, USA, August 2004, pp. 238 – 254.
- [5] G. Ruhe and A. Ngo-The, "Hybrid Intelligence in Software Release Planning," *International Journal of Hybrid Intelligent Systems*, Vol. 1(2004), pp. 99 - 110.
- [6] <http://www.releaseplanner.com>
- [7] O. Saliu and G. Ruhe, "Supporting Software Release Planning Decisions for Evolving Systems," in Proceedings of the *29<sup>th</sup> IEEE/NASA Software Engineering Workshop*, Greenbelt, MD, USA, April 2005, pp. 14 – 24.
- [8] D. Greer, "Decision Support for Planning Software Evolution with Risk Management," in Proceedings of the *16<sup>th</sup> International Conference on Software Engineering and Knowledge Engineering*, Banff, Canada, 2004, pp. 503 – 508.
- [9] K. Schmid, "A Comprehensive Product Line Scoping Approach and Its Validation," in Proceedings of the *24<sup>th</sup> International Conference on Software Engineering*, Orlando, USA, May 2002, pp. 593 – 603.
- [10] K. Schmid, "Scoping Software Product Lines – An Analysis of an Emerging Technology," in *Software Product Lines – Experience and Research Directions*, P. Donohoe, Ed. Kluwer Academic Publishers, 2000.
- [11] P. Carlshamre et. al., "An Industrial Survey of Requirements Interdependencies in Software Product Release Planning," in Proceedings of the *Fifth IEEE International Symposium on Requirements Engineering*, Toronto, Canada, 2001, pp. 84 – 91.