Mälardalen University Press Dissertations
No. 67

TOWARDS A CAPABILITY MODEL FOR RELEASE PLANNING OF SOFTWARE INTENSIVE SYSTEMS

Markus Lindgren

2008

School of Innovation, Design and Engineering

Copyright © Markus Lindgren, 2008
ISSN 1651-4238
ISBN 978-91-86135-06-5
Printed by Arkitektkopia, Västerås, Sweden

Mälardalen University Press Dissertations
No. 67

Towards a Capability Model for Release Planning of Software Intensive Systems

Markus Lindgren

Akademisk avhandling

som för avläggande av Teknologie Doktorsexamen i Datavetenskap vid Akademin för Innovation, Design och Teknik kommer att offentligen försvaras fredagen den 21:a november, 2008, kl. 09.00 i sal Gamma, Mälardalens Högskola, Västerås.

Fakultetsopponent: Docent Tony Gorschek, Blekinge Tekniska Högskola

Akademin för Innovation, Design och Teknik

Abstract

Release planning is an early product development activity concerned with deciding which features and quality improvements that should be pursued in development projects, i.e., it is an activity which to a large part decides how the development budget of a company is allocated.

This thesis investigates release planning for long-lived software intensive systems; systems which contain software, electronics, and mechanics, and have a life-cycle of 10-20 years. In performing release planning for these systems, the existing system, including its architecture, often represent a large investment which has impact on which features and quality improvements that are cost-efficient to include in a release. However, in industry today, little attention is given to the existing system during planning, resulting in decisions being based on uncertain information, and thereby increasing the risk of problems in the development projects.

This thesis is based on a multiple case study involving seven industrial companies developing and producing long-lived software intensive systems. There are several contributions in this thesis, aimed at understanding and improving the release planning process: (1) validation of previous research related to key-aspects for release planning including identification of short- and long-term planning as a new key-aspect; (2) the capture of state-of-the-practice for release planning; (3) a proposal for a capability model for release planning, which can be used to assess the capabilities of a company's release planning process, but also for identifying process improvement possibilities; and (4) a process for finding a balance between investments in features and quality improvements, developed based on the practices used at two of the most capable companies in the study. Finding such a balance is important since adding new features may attract new potential customers, while improving the quality for existing customers can reduce costs of poor quality.

ISSN 1651-4238

ISBN 978-91-86135-06-5

Abstract

Release planning is an early product development activity concerned with deciding which features and quality improvements that should be pursued in development projects, i.e., it is an activity which to a large part decides how the development budget of a company is allocated.

This thesis investigates release planning for long-lived software intensive systems; systems which contain software, electronics, and mechanics, and have a life-cycle of 10–20 years. In performing release planning for these systems, the existing system, including its architecture, often represent a large investment which has impact on which features and quality improvements that are cost-efficient to include in a release. However, in industry today, little attention is given to the existing system during planning, resulting in decisions being based on uncertain information, and thereby increasing the risk of problems in the development projects.

This thesis is based on a multiple case study involving seven industrial companies developing and producing long-lived software intensive systems. There are several contributions in this thesis, aimed at understanding and improving the release planning process: (1) validation of previous research related to key-aspects for release planning including identification of short- and long-term planning as a new key-aspect; (2) the capture of state-of-the-practice for release planning; (3) a proposal for a capability model for release planning, which can be used to assess the capabilities of a company's release planning process, but also for identifying process improvement possibilities; and (4) a process for finding a balance between investments in features and quality improvements, developed based on the practices used at two of the most capable companies in the study. Finding such a balance is important since adding new features may attract new potential customers, while improving the quality for existing customers can reduce costs of poor quality.

List of Included Papers

- *Importance of Software Architecture during Release Planning*, Markus Lindgren, Christer Norström, Anders Wall, Rikard Land, In Proc. Working IEEE/IFIP Conference on Software Architecture (WICSA), IEEE Computer Society, Vancouver, Canada, February 2008.

- *Key Aspects of Software Release Planning in Industry*, Markus Lindgren, Rikard Land, Christer Norström, Anders Wall, In Proc. 19^{th} Australian Software Engineering Conference, IEEE Computer Society Press, Perth, Australia, March 2008.

- *Towards a Capability Model for the Software Release Planning Process — Based on a Multiple Industrial Case Study*, Markus Lindgren, Rikard Land, Christer Norström, Anders Wall, In Proc. 9^{th} International Conference on Product Focused Software Process Improvement, Springer (LNCS), Rome, Italy, June 2008.

- *A Method for Balancing Short- and Long-Term Investments: Quality vs. Features*, Markus Lindgren, Anders Wall, Rikard Land, Christer Norström, In Proc. 34^{th} Euromicro Conference on Software Engineering and Advanced Applications, IEEE Computer Society, Parma, Italy, September 2008.

- *An Initial Capability Model for the Software Release Planning Process — Based on a Multiple Industrial Case Study*, Markus Lindgren, Rikard Land, Christer Norström, Anders Wall, submitted to the Journal of Software Process Improvement and Practice, for publishing in 2009.

- *Deriving Worst-Case Execution Time by Measurements*, published in Markus Lindgren's Licentiate thesis, MRTC report ISSN 1404-3041 ISRN MDH-MRTC-26/2000-1-SE, Mälardalen Real-Time Research Centre, Mälardalen University, December 2000.

- *Deriving Reliability Estimates of Distributed Real-Time Systems*, Markus Lindgren, Hans Hansson, Christer Norström, Sasikumar Punnekkat, In Proc. of RTCSA'2000, IEEE Computer Society, Cheju Island, South Korea, December 2000.

List of Additional Publications

- *Design and Scheduling of Shared Displays for Embedded Systems*, Hans Hansson, Markus Lindgren, MRTC report ISSN 1404-3041 ISRN MDH-MRTC-1/1999-1-SE, Mälardalen Real-Time Research Centre, Mälardalen University, March 1999.
- *System Development with Real-Time Components*, Damir Isovic, Ivica Crnkovic, Markus Lindgren, In Proc. of the International ECOOP2000 Workshop 22 - Pervasive Component-based systems, Sophia Antipolis and Cannes, France, June 2000.
- *Measurement and Simulation Based Techniques for Real-Time Systems Analysis*, Markus Lindgren, Licentiate Thesis, Uppsala University Printers, December 2000.
- *Using Measurements to Derive the Worst-Case Execution Time*, Markus Lindgren, Hans Hansson, Christer Norström, Sasikumar Punnekkat, In Proc. of RTCSA'2000, IEEE Computer Society, Cheju Island, South Korea, December 2000.
- *Release Planning in Industry — Interview Data*, Markus Lindgren, MRTC report ISSN 1404-3041 ISRN MDH-MRTC-219/2007-1-SE, Mälardalen Real-Time Research Centre, Mälardalen University, November 2007.

Acknowledgements

All journeys have a beginning. I can not really recall when mine started, but somewhere along my journey a friend of mine showed me the magical (?) world of computer games. In the beginning I was quite amused by playing games, however, I soon wondered how games were developed. Over the years the focus on computer games changed a bit, but somehow I ended up studying Computer Engineering at Mälardalen University where it turned out that computers could be used for useful things (besides playing games).

In my final undergraduate year Christer Norström asked me if I wanted to become a PhD student. At this time I did not really know where my journey was heading, but I figured it would not hurt to learn more. In my first research years, 1998–2000, Hans Hansson was my main supervisor with much appreciated support from Christer, Henrik Thane, and Sasikumar Punnekkat; you all are exceptional people whom I am very greatful for having the opportunity to work with! This research eventually led to the defense of my Licentiate thesis in 2000, after which I started working as a consultant. Still, I occasionally took some courses in case I would sometime aim for a PhD. After a few years in industry Christer, again, asked me if I wanted to do more research. Well, I figured it would not hurt to learn some more...

In my second period of research, 2005–2008, Christer Norström, Anders Wall, and Rikard Land acted as my supervisors. One thing is for certain, this work would never have been completed without your excellent support. Many thanks go to you! Still, I do not consider the idea of having my PhD defense in Second Life as a good idea; the idea was entertaining though. You all three are highly skilled and I really enjoy your company, both when doing research and when having pointless (?) discussions. Most things in life are easier to deal with when having fun!

Work is also more fun when shared with good colleagues. During these years lots of pasta has been shared with Thomas Nolte and Daniel Sundmark, joined by many laughs and occasional (mental?) training sessions, which aid in digesting research. Lots of good cooperation has been established within the BESS research group, e.g., Stig, Håkan, Peter, Jocke, Stefan, Pia, and many more people at the School of Innovation, Design and Engineering (none mentioned, none forgotten). I hope this cooperation can continue for many years to come.

My colleagues at ABB, both old and new, have shown great patience in allowing time for me to complete my research. I am almost surprised that none of you have even tried to "steal" any of my belongings at work. I can actually not recall a single time when any of you have questioned me spending time at the university, even though there has been lots and lots of work to do. We will be seeing each other more frequently now.

This work has been partially funded by the KK Foundation via the research school RAP. Gratitude should also be expressed to ABB Force Measurement for allowing me to conduct research as part of my work. In addition, this thesis would not have been possible to complete without the cooperation from the anonymous companies, and people, involved in this study via interviews.

My research journey has passed through many interesting places over the years, such as Berlin, Paderborn, Madrid, York, Cannes, Cheju Island, Vancouver, Perth, Rome, and now finally, Parma. However, all journeys must come to an end, but my journey is far from the end.

Markus Lindgren
Parma, September, 2008

Graduate School
Intelligent Systems for
RAP
Robotics, Automation
and Process Control

Preface

My research education has been conducted in two different phases between the years 1998–2000 and 2005–2008, with work in industry between these phases. In this chapter I (i) describe the background to my research education and the focus in each of these two time periods and (ii) present a summary of my research conducted in the years 1998–2000 and relate it to the research in this thesis.

Research Education Background

My research education has been conducted over the years 1998–2000 and 2005–2008. The focus of my first research effort was on predicting properties of real-time systems, which was concluded by a successful defense of my licentiate thesis [Lin00] in the year 2000. After completing my licentiate thesis I started working as a consultant in industry, where examples of assignments included: evaluation of using object-oriented design and implementation for train control systems, development of a concept for a next generation train control communication infrastructure, and design and development of a flexible software architecture for mechanical control as part of a flatness control system operating in cold-rolling mills. During these 5 years, working at different companies developing software-intensive products I quickly realized that my prior research efforts would have little impact in improving these products.

In 2005 I started working as an industrial PhD student at ABB with initial aim of progressing research within the area of software architecture. In the beginning the research effort was devoted to (i) studying literature within the area of mainly software architecture, and (ii)) attempting to formulate a suitable research problem. In rough terms, in

industry I observed products which I considered would benefit if their software architectures were improved. However, improving the architecture of an existing system is often associated with relatively large costs. Hence, to be able to allow for such architectural improvements there must be a part of the budget assigned to address these problems. Rather soon this line of reasoning comes across what is most relevant to do from a business point of view. Since the budget normally is limited, this could mean that if a software architectural improvement is performed, then there could potentially be no budget left for adding more features; features which potentially could increase revenues. This is, in short, how my research area for the years 2005–2008 became release planning; with much focus on improving the release planning process. As will be explained, in Chapter 1, it is relevant to consider the existing system during release planning including its software architecture.

Given this background it is quite natural that the two parts of the research have different direction. Or rephrased, it would be strange if the research would have identical focus after 5 years in industry, since this would imply that I would have learned little during these years.

Relation to Licentiate Thesis

This section summarizes the research efforts previously published as part of my licentiate thesis [Lin00]. The efforts were mainly in two areas:

- A method for estimating the worst-case execution time (WCET) of a program.
- A method for deriving reliability estimates for distributed real-time systems.

For the sake of completeness, the two main publications from my licentiate research are included in Appendix A and Appendix B, respectively. A summary of the licentiate research follows in the next two sections. We conclude this preface by describing some notable differences in research method and research area between my pre and post licentiate research.

My licentiate thesis was devoted to predicting properties of a single software architecture quality attribute (see Chapter 2.7 for a description of quality attributes) namely *performance* in the context of real-time systems. Real-time systems are often distributed systems where multiple computers, possibly using different kind of processors, and possibly

connected with different kind of network technologies. It is worth noting that all companies being part of the study presented in Paper A–E of this PhD thesis develop and produce real-time systems. Furthermore, this PhD thesis focuses on considering all quality attributes of an architecture, rather than a specific attribute, as was done in my licentiate research.

Worst-Case Execution Time by Measurements

Appendix A, originally published in my licentiate thesis [Lin00], presents a method for deriving the worst-case execution time of programs by performing measurements on the target hardware. As a basis for the method, we use a well known result by McCabe [McC76] and others, namely, the observation that there is a set of independent program paths from which all paths within a program can be expressed. The core of the method is to derive linear equations describing the execution time for executed paths as the sum of the visited independent paths. This procedure is repeated for a set of input data to obtain a system of equations. The solution to this equation system gives execution times for the individual sub-paths, which is one part of the problem of deriving WCET estimates.

The benefits of the approach are that: (i) measurements can be performed on the *whole* program; there is no need to perform measurements on isolated parts of the program, (ii) the target hardware can be used for the measurements, which eliminates the tedious and error prone work of creating an accurate timing model of the target system. Both these issues are becoming increasingly important since the timing of many processor instructions are becoming context dependent. For instance, to derive the execution time for a basic block on a pipelined processor, it is important to know which basic block is executed before and after the block. Therefore measurements can no longer be performed on isolated parts of the program, the *whole* program must always be taken into account if we need accurate results. Our method avoids the need for exhaustive measurements of program paths by measuring only the independent sub-paths, which often are order of magnitudes fewer than the total number of paths.

Deriving Reliability Estimates for Distributed Real-Time Systems

Real-time scheduling techniques have not gained widespread use in industry yet. One of the reasons for this is that the current methods seldom match how systems are actually implemented, nor do they cope well with the hardware that is used. We should, of course, mention that there are areas in which real-time scheduling is used successfully, but also that there is a large class of applications for which it is not directly applicable, mainly those with soft real-time requirements. It is also important to note that the hard real-time requirements which exist in schedulability models do not exist in the real world, hence, these models often deal with "artificial" requirements.

Appendix B presents a novel simulation technique for distributed real-time systems. From simulations of a system we observe when errors occur, in particular violations of timing constraints, and then we make statistical predictions on the overall reliability of the system (in terms of meeting timing constraints). The major benefit of this is that the analysis can be performed on all kinds of systems. The drawback is that we use statistics, and hence, cannot provide strict guarantees about the timing behavior of the application (except in special cases). However, industrial applications often require a reliability estimate, rather than timeliness guarantees.

Notable Differences in Research

The two periods of my research, 1998–2000 and 2005–2008, have had different direction. In the first period focus was on development of methods for predicting real-time properties, while the second period has had a main focus on processes and how people cooperate to develop technical solutions. In this section I comment on some notable differences when conducting research in these two different areas.

In 1998–2000 I used traditional computer science research methods, i.e., quantitative research methods as described in Chapter 3.1, where I in essence followed the following process:

1. Develop a method, or model, using existing research and experience as an aid, i.e., the creative part.

2. Quantitatively evaluate the method, for example, by benchmarking a set of similar methods and compare a set of known variables. For example, for WCET estimations we can compare the pessimism of the estimation on a set of programs with *known* WCET, and we can compare the time for computing WCET estimates.

3. Evalutate research results. If a new method shows to be better in any of the evaluated variables we basically have a research result worth publishing. From a scientific point of view this is solid work, however, but from an industrial perspective such a result does not necessarily provide any immediate value.

In 2005–2008 my research focused on improving the release planning process. Compared to the work in 1998–2000 there are some rather distinct differences:

- The number of relevant variables is unknown.
- It is hard to study improvement of release planning processes in a constructed lab setting. Instead this research must capture and observe practices being used in real context, e.g., at companies. One way of capturing such data is by interviews. That is, interviews is a useful method for collecting data; data which in, e.g., the WCET research relatively simply can be measured.
- All relevant variables can not be quantitatively measured. In particular, data collected during interviews are typically qualitative.
- How can we compare and evaluate different release planning processes? What really matters for release planning is basically building good business, but this depends on many things, e.g., market demand. We can of course evaluate other aspects, such as number of decisions made, but with the drawback of such aspects possibly not being good indicators of a good release planning process.

Another way of looking at these two research areas is that my research in 1998–2000 was focused on developing and evaluating technical solutions, while my later research focused on developing and evaluating processes, i.e., focus on how people work, or should work. In the pre licentiate period, the main challenge was in the technical work, not in applying the research method, whereas the main challenge for the research in the post licentiate period has been related to the methodology, which requires more training and is much harder to master.

Contents

I

Thesis

Chapter 1

Introduction

Remaining profitable is becoming increasingly hard due to tough competition and globalization. For companies basing their business on selling products this means that their products must be developed, produced, and marketed using lower and lower cost. At the same time these products need to higher degrees exceed customers' expectations on quality, cost, and functionality to beat the competition. Failure to consistently be profitable can result in going out of business. Product development is concerned with identifying and transforming market needs and opportunities into products, products which later (hopefully) can be sold to customers and generate profit to the producing company.

This thesis addresses an early phase of product development during which it is decided what should be included in the next version of a product. Our focus is product development of long-lived, i.e., in the range 10–20 years, large software intensive products, for which product development is truly a non-trivial task. Examples of such products are: telecommunication systems, train control systems, and automation systems.

There are several reasons to why product development of these kinds of products is a non-trivial task, which we divide into *internal* and *external* factors:

Internal factors refer to properties of the products themselves and factors existing within these kind of companies. A company typically has some degree of control of these factors.

Complexity The products themselves typically contain significant complexity.

Diverse competencies Software intensive products typically contain software, electronics, and mechanics. Hence, development of these products requires cooperation among people within diverse competence areas.

Scale Not only are the products complex, but in addition the development organization is usually large and often distributed. That is, development of these products involves many people, which in itself is a contributor to development complexity. Typically there is not single individual within these kinds of companies that understand the product in full. Hence, product development is largely a cooperative effort.

Evolution These products are developed in steps during the products' life-cycles, where each step builds on earlier steps. From a technical point of view this can become problematic, since over the years there will be requirements changes that potentially make the original technical solution inappropriate. Furthermore, it is not certain that the people that developed the original technical solution still remain at the company. Over the years there can also be significant changes in the competence required to develop these products using emerging technology.

External factors refer to factors outside the control of the company.

Legislation & standards New legislation and standards may become mandatory to follow. For example, in the car industry there are regular updates to the allowed emission rates, which forces development of more efficient engines; actually quite a lot of software is required in today's car engines in order to be able to comply with legislation. Inability to comply with a country's legislation will rule out the possibility of selling products in that country, which can lower revenue possibilities for a company. New standards can place new requirements on the processes used in developing these products, e.g., the new machine directive 2006/42/EG in Sweden which is compliant with the European Union directive. Safety standards

also place requirements on development processes, such as the IEC 61508.

New technology The rapid evolution of processors and network technology enable new functionality to be developed, which earlier was realized as pure electrical or mechanical solutions, i.e., software is becoming increasingly important in today's products. As a consequence, the software in these products is becoming more complex. Such innovations include anti-lock brake systems (ABS) and anti-skidding systems in cars.

Integration Today's technology allows integration of more systems than earlier. Again, this increases the complexity in these products. For example, consider today's mobile phones. Earlier we used our phones for the sole purpose of calling other people. Today, these devices support SMS, MMS, internet surfing, reading of e-mails, cameras, music players, etc. etc. Actually, some of today's mobile phones contain functionality previously only found in desktop computers.

Changing expectations During these products' life-cycle there can be significant changes in customer expectations, and also in the available technology, which further complicates the problem. Hence, to be attractive on the market it is normally required to constantly update the product offering. Again, consider a mobile phone as an example. What were your expectations on functionality 10 years ago? What are your expectations today? What price are you willing to pay to obtain this functionality (before and now)? Much can happen in 10 years...

As mentioned, this thesis addresses an early phase of product development during which it is decided what should be included in the next version of a product. Typically, this phase is referred to as *release planning* or *product planning* [RS05, Car04, BAW06]. Release planning can be seen as a company-wide optimization problem where the goal is to maximize utilization of the often limited resources, such as budget and developers, of a company and turn them into business benefit. Normally the cost of implementing all proposed needs[1] is larger than the budget

[1] A *need* is a proposal for a change that normally is negotiable to some extent; needs later become project requirements. Needs can be proposed by both external stakeholders, e.g., customers, as well as internal stakeholders, such as developers.

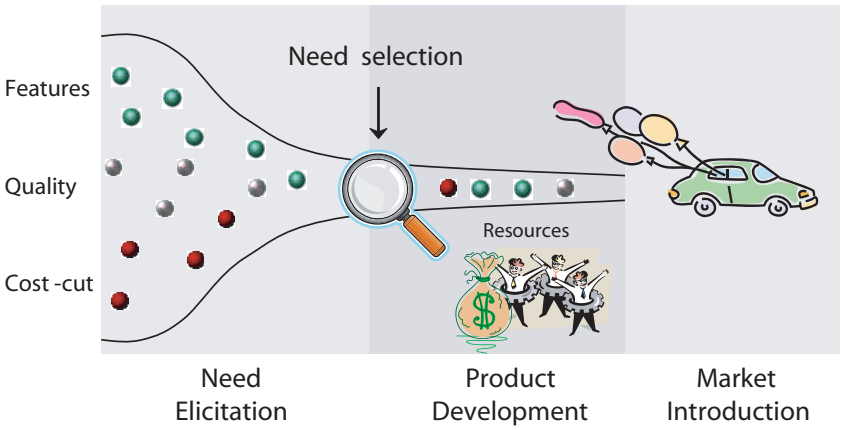


Figure 1.1: Overview of need prioritization during release planning.

allocated to a release, therefore a decision needs to be made of what to include in a release and what to postpone. Also, the *set of needs* has to be prioritized in order to maximize business value of the needs included in a release.

As input to release planning are a set of *needs* that, when implemented as part of a product, provides some business/customer value. A need is a proposal for a change that normally is negotiable to some extent; in later phases needs are transformed into product requirements. A need typically can be classified as being either a *feature*, *quality improvement*, or *cost-cut* request. Quality improvements are concerned with improving some existing part of a product, for example, making a product more user-friendly or improve its performance. Features extend the product in some way, e.g., by adding new functionality. Cost-cut initiatives are typically aimed at reducing the production cost, but can also be related to process improvements, e.g., improve the order handle flow within a company. Cost-cut initiatives can result in further feature or quality improvement needs. For example, by using less material during production it may be possible to lower the production cost. However, when using less material it is possible for the mechanical construction to be weakened. In some cases this can be compensated for by using more advanced control algorithms. Hence, in such a case a cost-cut initiative

also result in, secondary, new feature needs. Figure 1.1 illustrates how the different kinds of needs are selected and prioritized before they reach product development, where a need is symbolized by a grayed circle; different shades of gray for different kinds of needs.

This thesis is concerned with improving the release planning process, for which there are several motivations:

- From an *industrial perspective* release planning is a significant contributor to deciding what features, quality improvements, and cost-cut activities that product development should focus on. Hence, it is a major contributor to deciding what will be released to the market, i.e., it is concerned with building the company business. Typically there are many stakeholders involved in making decisions on the needs to include in a future release. The selection of what needs to include is a complex task, which can not be computed mathematically, since there basically is no right or wrong choice; only better or worse choices. Still, it is not desired to have such a decision be influenced by irrelevant factors, such as own personal gain, since in large parts these decisions will affect the future company business.

- From an *academic perspective* release planning has received relatively little attention, at least when considering its importance for company success. There are sub-problems within the area that have received attention, however, few approach the problem in its entirety.

Release planning is an early product development activity, as indicated in the description above, but often it is also a continuous activity lasting the entire product development project(s) assigned to implement the prioritized needs. The reason for this are uncertainties and risks associated with implementing new needs as part of product(s), as is typical in most product development, which may require replanning.

There are several problems involved in performing release planning for these kinds of products. One particular such problem with software for long-lived products is that *software ages*, as is pointed out in Software aging [Par94] and the Laws of Software Evolution [Leh80]. This aging occurs due to, e.g., changes in customer expectations, the developers making changes do not understand the original design concept, or the original design solution not being suitable for new needs. Furthermore,

as the software grows in size it becomes more complex and as a result even harder to modify. As is pointed out by Spinelli in [Spi07]: "We haven't yet come to terms with the idea that software ages, often beyond salvation." However, redesigning parts of, or entire, industrial software systems is associated with large costs and risks [MWN+04], which can potentially cause seemingly sound investments made in software become poor investments. This is one reason for companies often being reluctant to embark on large redesign efforts. Another reason is that software quality improvements to existing systems are often considered being investments with relatively long time before a sound return-of-investment can be reached, and therefore it can be especially hard to decide when, and which, quality improvements to include in a release [Lin07].

Whether we like it or not, software deteriorates over time [Lan02], and as a consequence it needs to be maintained. However, sometimes redesign is required to improve the software's state. In practice, redesign decisions typically are hard to make since it basically means throwing away an investment already made. The decision is hard since in the short-term it can be more cost effective to do a quick fix that solves the problem, which typically results in a *technical debt* [FBB+99]. However, such a solution often makes it harder to extend the software further. Hence, in the long-term it can be more cost effective to redesign the software. Furthermore, there are often many risks and uncertainties in making this decision.

The aim of this thesis is to develop a process that can be used for improving the release planning process of an organization, with a specific focus on how to find a balance between short- and long-term investments, which can be translated into how to balance investments in feature growth, quality improvements, and cost-cut initiatives.

1.1 Research Motivation

In this section we discuss the motivation for this research. As mentioned in the beginning of Chapter 1 release planning is concerned with identifying the needs providing greatest return of investment [SR05b, SR05a, RS05, Car04]. More loosely expressed, release planning is concerned with identifying the needs, that if implemented as part of a product, provides possibilities for fulfilling the needs by both new and existing customers as well as other stakeholders having interest in the product,

Project Success Factors	% of Responses
1. User involvement	15.9%
2. Executive management support	13.9%
3. Clear statement of requirements	13.0%
4. Proper planning	9.6%
5. Realistic expectations	8.2%

Table 1.1: Top 5 project success factors [Sta95].

Project Impaired Factors	% of Responses
1. Incomplete requirements	13.1%
2. Lack of user involvement	12.4%
3. Lack of resources	10.6%
4. Unrealistic expectations	9.9%
5. Lack of executive support	9.3%

Table 1.2: Top 5 project impaired factors [Sta95].

thereby hopefully contributing to increased sales; assuming that those needs attract customers willing to buy the product. From a business perspective the goal of release planning is to optimize profit.

Product development, of which release planning is one of the early tasks, is required for realizing the plans decided during release planning. Sadly, based on the results presented in the Standish Group Report [Sta95] one can conclude that software development is a rather immature engineering discipline, since only 16.2% development projects are completed on time and on budget and in larger companies only 9% deliver on time and budget. Table 1.1 presents the factors identified as contributed most to project successes, while Table 1.2 presents the factors identified as contributing most to projects being cancelled. Most of the factors in Table 1.1 and Table 1.2 are related to release planning.

A more recent publication by the British Computer Society [oE04] indicates that the situation still is in need of improvement, although it may have improved. For example, in one survey conducted by the British Computer Society only 3 out of more than 500 development projects were

considered successful[2]. Furthermore, in their summary they conclude that "... the most pressing problems relate to the people and processes involved in complex IT projects" [oE04].

Today most product development is required to be based on existing systems for economical and time-to-market reasons [MWN+04]. This forces existing systems to be evolved by adding features or improving existing features. During such evolution the existing system often has impact on which extensions and improvements are easy, hard, and possible to perform [BCK03], thereby also possibly having impact on what extensions and improvements provide greatest return of investment. The importance of considering evolution aspects is further motivated by "... the software development community is coming to grips with the fact that roughly 80 percent of a typical software system's cost occurs *after* initial deployment" [BCK03]. Furthermore, within the software engineering community it is known that as a software system ages the cost of change will increase [Spi07, Par94]. It is also known that software aging is inevitable, just as it is for humans. Parnas writes that "our experience with software aging tells us that we should be looking far beyond the first release to the time when the product is old" [Par94].

Software architectural concerns should be addressed early on in order to enable fulfillment of business critical quality attributes [BCK03]. Quality attributes are basically determined by non-functional requirements, such as performance, safety, and security. Failure to fulfill critical quality attributes results in a product useless to its intended customers. Furthermore, to modify the quality attributes of an existing system often is associated with high costs. It would therefore seem natural, from a business point of view, to address these issues early on. However, from a software architectural point of view there are more problems with product evolution occurring over long time, in the range 10–20 years, which complicates the problem even further. For example, during such time spans customer expectations may radically change, new technology may become available enabling new kinds of products to be developed and/or reducing cost of product development, and there may be changes in business goals. These changes can have impact on the software architecture required to support these changes.

To summarize the motivation:

[2]By successful they refer to projects which meet expectations on cost, quality, and delivery time.

- Release planning is an important activity aiming at identifying the needs providing greatest return of investment.
- In industry most product development is performed by evolving existing systems, consequently the existing system needs to be considered during release planning.
- The software architecture of the existing system can be an enabler to new features, but it can also place constraints on what can be done cost-effectively.

1.2 Research Questions

This section presents the research questions addressed by this thesis. We are using a research approach based on grounded-theory [SC98], i.e., we develop models/methods based on observed data. In grounded-theory research it is common to not have the full set of research questions defined before commencing on the research effort, instead these questions are formulated as part of the research. This is the case in this research as well. More details concerning the research method used during this research are presented in Chapter 3, including more information on the formulation of the research questions. However, the aim of this research is to improve the release planning process[3]. As a consequence the research questions have been formulated as questions that need to be answered in order to obtain insights and knowledge that enable release planning process improvements to be developed.

As a starting point in this research we need to understand what aspects of release planning industry considers being key, and consequently what aspects are relevant to improve, we first need to identify what these aspects are. The first research question (RQ) therefore is:

> **RQ1:** *What are the key aspects of software release planning in industry?*

As an aid in answering this question we have used the release planning key aspects proposed by Saliu and Ruhe [SR05a] as a reference model,

[3]Initially the aim was to develop a method useful in deciding the balance between investments in features and quality improvements, however, while conducting the research the focus has shifted more towards process improvement; the research method is described in Chapter 3.

i.e., we also attempt to validate their results. After gaining knowledge concerning what industry considers being key, we can start studying the practices used in industry during release planning, i.e., how do current practices enable fulfillment of the key aspects:

> **RQ2:** *What are the (observed) best-practices for release planning in industry?*

To answer RQ2 we need to capture the current state-of-the-practice and to among a set of practices identify the best-practices. Companies usually have different motivation for using a specific set of practices, understanding these motivations and the problems these are aimed at addressing provide valuable insights. The aim of this research is to improve the release planning process of an organization, and specifically how to decide the balance between investments in features and quality. Our third research question is:

> **RQ3:** *Given the answers to RQ1 and RQ2, how should a company find a balance between investments in features and quality?*

By answering research questions RQ1 and RQ2 a number of valuable insights will be gained for release planning in industry. These insights aid in developing methods that improve the release planning process, RQ3. In Chapter 4 we will return to the research questions and discuss their answers, and the relation to Paper A–E.

1.3 Scope

The focus of this research is primarily aimed at improving release planning for software intensive products with long-life cycles, typically in the range 10–20 years. Software intensive products are not pure software products, instead these contain mechanics and/or different kinds of electronics. However, today most feature growth is enabled by software and software has a significant contribution to the functionality of the product. Products existing in this domain are, for example, automation systems, telecommunication systems, and transportation systems. These kinds of products are normally developed in steps, i.e., in a series of releases, where the existing system typically represents a large investment, which has impact on what features and improvements are cost-effective to perform.

There exist different types of releases for software products, each with different purpose. There are variations between companies in terms of which types of releases they use and how these are denoted. For example, Case 7 in [Lin07] uses three kinds of releases: *main release*, *service pack*, and *bug fixes*. Bug fix releases contain corrections to problems (released frequently, say once per month), service releases contain corrections to problems and sometimes new functionality, and the main release introduces mainly new functionality (released more seldom). This research is focused on release planning for major releases, i.e., the type of release typically adding most new features and improvements The main releases studied in this thesis are usually released every 6 to 24 months.

A comment just to avoid possible confusions, the concept of release planning is also used within the Agile community [Agi07], where it refers to planning of smaller incremental releases. We are using the term release planning since this is used in research where the existing system is taken into consideration [SR05b, SR05a], and we refer to planning of mainly major releases, rather than smaller incremental releases.

Agile development, which is briefly discussed in Chapter 2.2, generally is considered to be well suited for small development teams, up to 15–20 developers [Boe02, Con01]. In this work we are primarily investigating release planning for products being developed by more developers than considered appropriate for Agile development. The products studied in this thesis are typically developed in some 20–30 sub-projects, which together form the major release. However, it may be possible to use Agile development practices in some of these sub-projects. Based on this one can conclude that the companies studied in this thesis are relatively large. However, due to non-disclosure agreements we cannot reveal any absolute numbers concerning their size.

This thesis is not concerned with portfolio management, i.e., the task of managing a set of products. Instead the focus of the thesis is mainly on improving an existing product within an existing market. However, it may be the case that some of the results in this thesis are also applicable outside this scope.

Assessment is required for identifying the current state of the release planning process within an organization, since without knowing the current state it is hard to identify what to improve. However, this is not within the scope of this research.

Apart from being a large and interesting area for a large audience, our personal motivation for having the described scope for this thesis

is threefold, (i) a significant and important part of Sweden's industrial companies are operating in this context, (ii) we have existing contacts into some of these companies which make them relatively accessible for studies, and (iii) it is in our personal area of interest.

1.4 Thesis Overview

This thesis is published as a *collection of papers*, hence, it consist of two main parts. The outline for the first part of the thesis is as follows:

Chapter 1 This chapter, introduces release planning, the research questions, scope of research, and provides an overview of the thesis.

Chapter 2 Presents a selection of the related work for release planning.

Chapter 3 Presents the method used in conducting this research, as well as a discussion concerning threats to validity of the research.

Chapter 4 Presents the research results resulting from this thesis and provides answers to the research questions stated in Chapter 1.2.

Chapter 5 Summarizes the contributions of this thesis and discusses possibilities for future work.

The second part of the thesis contains the collection of papers included in this thesis. Note that these papers have been reformatted according to the thesis template, however, the content of the papers are identical to the published versions of these papers. The outline for the second part is as follows:

Chapter 6, Paper A "Importance of Software Architecture during Release Planning", focuses on how software architecture is considered during release planning and specifically on how the role of a software architect is influenced. Results are based on a multiple case study involving 7 industrial companies. Among the findings are: product management has insufficient knowledge to address architectural issues, still the process employed for release planning at a company can have possibilities of addressing architectural issues. Presented at Working IEEE/IFIP Conference on Software Architecture (WICSA), Vancouver, Canada, February 2008.
Authors: Markus Lindgren, Christer Norström, Anders Wall, and Rikard Land. [LNWL08]

Chapter 7, Paper B "Key Aspects of Software Release Planning in Industry", uses the release planning key-aspects proposed in [SR05a] as a reference model and then, using a multiple case study, validates some of the key aspects proposed in [SR05a]. One new key aspect for release planning is identified, short- and long-term planning, and two of the key-aspects proposed in [SR05a], scope and prioritization mechanism, are determined not to be key aspects for release planning. Presented at the 19^{th} Australian Software Engineering Conference (ASWEC), Perth, Australia, March 2008.
Authors: Markus Lindgren, Rikard Land, Christer Norström, and Anders Wall. [LLNW08a]

Chapter 8, Paper C "Towards a Capability Model for the Software Release Planning Process — Based on a Multiple Industrial Case Study", presents a starting point for a capability model for release planning, which is based on generalizing the work in Paper A, Paper B, and Technical Report A. Based on [Lin07] it is clear that companies release planning processes differ quite much. The general idea of the model is to aid companies in improving their release planning process. The model is based on experiences gained from the earlier multiple case study. Presented at the 9^{th} International Conference on Product Focused Software Process Improvement (PROFES), Rome, Italy, June 2008.
Authors: Markus Lindgren, Rikard Land, Christer Norström, and Anders Wall. [LLNW08b]

Chapter 9, Paper D "A Method for Balancing Short- and Long-Term Investments: Quality vs. Features", further builds on the results in Paper A and Paper C. This paper motivates and proposes a method for balancing investments in quality improvements and feature growth. Presented at 34^{th} Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Parma, Italy, September 2008.
Authors: Markus Lindgren, Anders Wall, Rikard Land, and Christer Norström. [LWLN08]

Chapter 10, Paper E "An Initial Capability Model for the Software Release Planning Process — Based on a Multiple Industrial Case Study", is an extended version of Paper C where parts of Paper D have been integrated; Paper C was invited for journal publication

after its presentation at PROFES'08. **Submitted to the Journal of Software Process Improvement and Practice**.
Authors: Markus Lindgren, Rikard Land, Christer Norström, and Anders Wall. [LLNW09]

Appendix A and Appendix B Contains the two main publications from my successfully defended Licentiate thesis [Lin00], which are provided here for completeness. As has been discussed in the Preface on page vii my research has been conducted in two different time periods, 1998–2000 and 2005–2008, and in two different, but related research areas.

My contributions: I am the main contributor of Paper A–E, which are co-authored by my supervisors. Furthermore, I am the main driver behind the case study design and also the main driver during the interviews in the study. During each interview one of my supervisors has been present with main purpose of taking notes during the interview. After each interview I have written a short report based on the notes taken during the interview, which then has been discussed with the supervisor being present during the interview. The general workflow when writing the above papers is that first I come up with an idea for a paper which I then discuss with my supervisors. Based on the feedback from these discussions I create a first draft of the paper. The draft is proof read by my supervisors after which I adjust the areas I consider being relevant to adjust. The last step of the process is usually iterated a few times for each paper.

Chapter 2

Related Work

This chapter presents a selection of related work for release planning, where our intention is to cover areas most closely related to release planning (as treated in this thesis) in greater depth. However, we also present research in other related areas, although more briefly. We start by a clarification of terminology in Chapter 2.1 and continue with a rather general introduction to product development in Chapter 2.2, which concludes with a short overview of research areas related to this thesis. Chapter 2.3 provides an introduction to quality models. Release planning processes are covered in Chapter 2.4, continuing with research focused on developing release planning algorithms in Chapter 2.5, and with requirements prioritization discussed in Chapter 2.6. Finally, in Chapter 2.7, we first give an introduction to software architecture and continue with a few examples of architecture-centric methods.

2.1 Terminology

Planning can be done at several different levels in an organization [DeG00, LKK05, JS06], ranging from *vision*, *business strategy*, *product strategy*, *release planning*, to *project planning*. Release planning, for example, is concerned with deciding the features, quality improvements, and cost-cut initiatives which should be pursued for a future product release. However, the terminology used for these different levels of planning, and what is actually captured by each term, is not clear-cut. In this section

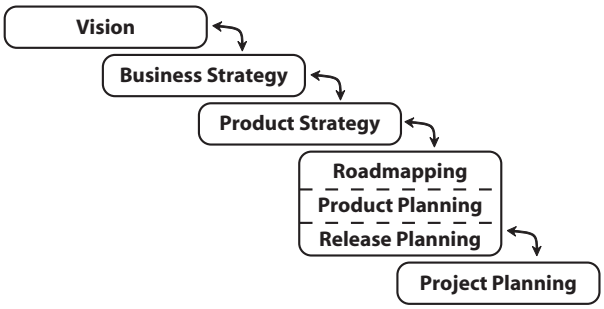


Figure 2.1: Different levels of planning within an organization.

we briefly describe this related terminology and shed some light on where release planning fits in.

An example set of levels of planning are illustrated in Figure 2.1. The vision captures the long-term (visionary) goal for an organization. For example, the vision for a company developing safety equipment for cars the vision could be zero fatalities in traffic. The business strategy, which is linked to the vision (illustrated by the arrows in the figure), would express how to profit from the vision. The product strategy in turn would focus on desirable product properties, which would be linked to the business strategy. The next levels of planning, roadmapping, product planning, and release planning, deals with deciding what and when specific product features and quality improvements should be released. In the literature it is recognized that these terms are sometimes used with different focus, and sometimes they refer to the same level of planning [LKK05, JS06, RSO08, RS05]. Hence, there is a grey zone in terms of what is really meant by these terms.

Terminology in relation to this thesis: We use the term release planning in this thesis, since the term release planning is mainly used when consideration is taken to the existing system [SR05a, RS05, JS06]. We consider product planning being an activity conducted on the same level of planning as release planning, but there seems to be a stronger focus on market and business issues in product planning. Due to this the market and business aspects have less focus in this thesis, but are included via judgments by market and business stakeholders. However, since release planning considers the existing system there is sometimes

also a grey zone in terms of what belongs to release planning and project planning respectively. One way of viewing this is that release planning requires the use of project resources to be able to increase certainty in information used during release planning decisions.

2.2 Product Development

Models for software development have been developed in order to aid in the endeavor of building quality software, with aim of projects meeting both budget and time constraints [Som00]. One well-known and, at least historically, often used development model is the waterfall model, illustrated in the upper part of Figure 2.2. In this model there is a sequence of process steps that need to be carried out during development. It starts by *requirements elicitation* and *requirements analysis*; these activities are focused on *what* to develop. The requirements are then passed to *design*, where the top-level design is defined; the focus here is on *how* to build a product fulfilling the requirements in a cost-effective way. During *coding* the design is refined and implemented as part of the software product, using some programming language. Before releasing a product to the market the product typically goes through different kinds of *testing*, e.g., unit testing, integration testing, and system testing. After the product has been tested it is released to its customers. When a product becomes used by its customers it is common with new requests for new features and improvements, which are dealt with during *maintenance*. There are variations of the waterfall model, for example, with reviews and alterations between the different steps. However, the waterfall model is by many considered being an inflexible way of developing software.

Another development model is the Rational Unified Process (RUP) [Kru00, JBR99], illustrated in the lower part of Figure 2.2. In the waterfall model the result, the product, is not visible until all process steps have been performed, while in RUP there are *iterations* that each produce some piece of (working) software which can be evaluated. Within each iteration there are different phases of development, which each has a different focus. The phases are *inception*, *elaboration*, *construction*, and *transition*. One rather distinct difference when compared to the waterfall model, is that the different activities are no longer strictly performed in sequence. Instead, as is illustrated in Figure 2.2, the amount of time used for requirements, design, implementation (coding), and test

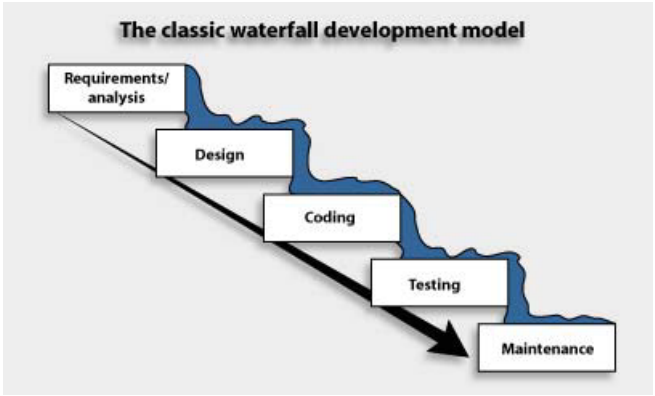
varies in each phase. It is worth noting that the RUP is a general software development model which needs to be tailored to the needs of an organization before being used.

In more recent years Agile [Agi07] development models have been proposed, which are light-weight processes at least when compared to RUP. Agile models are more focused on the code being developed and the users that will use the software, rather than the process used in developing the software. However, in the early days of Agile there were problems of scaling up Agile development teams; up to roughly 15 people worked reasonably well.

Product development in relation to this thesis: In this thesis we are addressing products/systems which are incrementally developed over a sequence of releases, where each release extends/improves an earlier release. What cannot directly be seen in the waterfall model and RUP, see Figure 2.2, is the impact that the existing system has on the desired new features and improvements. A perhaps more faithful view of the problem is illustrated in Figure 2.3. The figure illustrates how the current software architecture of a product enables/constrain the achievable quality for a product. The quality of a product has impact on the features and improvements its users request, hence, the quality of a product (indirectly) has impact on its desired requirements and quality attributes. The requirements and desired quality attributes in turn have impact on the software architecture. In a way one can see a new main release as going through the circle once in Figure 2.3. It is actually quite rare in industry to develop new products without having an existing system to consider [MWN+04], as indicated in Figure 2.2.

Release planning is concerned with eliciting and identifying the needs providing greatest return-of-investment. At the same time, in a software development project there is refinement of these needs into product requirements. Due to risks and uncertainties in development projects there can be, at later stages, need to modify the project scope. Due to this reason some of the related work for release planning can be found in requirements engineering, part of this work is described in Chapter 2.4 and Chapter 2.6. As discussed in Chapter 1, there is need to consider the quality of the existing product(s) during release planning as well as the current architecture for the product, since the architecture have impact on the development cost. A brief overview to quality models is presented in Chapter 2.3 and an introduction to software architecture in Chapter 2.7.

a)



b)

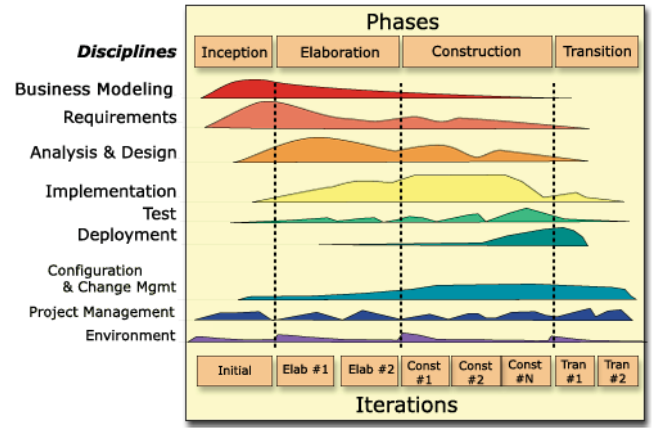


Figure 2.2: Two examples of software development models [picture a) from [Som00] and picture b) from [JBR99]].

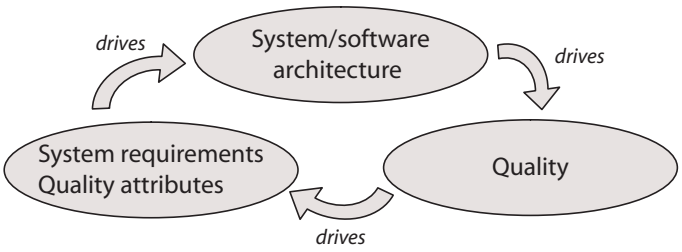


Figure 2.3: The software architecture impact on quality (picture adapted from [Fis98]).

2.3 Quality

In Chapter 1 we mentioned that it is relevant to consider the quality of a company's existing product(s) during release planning, however, we only discussed quality in rather general terms. In this chapter we provide a brief introduction to software quality and software quality models.

As a starting point we need a definition of what is meant by *quality*. According to the survey by Milicic on quality models in [LMW05] there are two different viewpoints on what quality is, namely:

Conformance to specification *Quality that is defined as a matter of products and services whose measurable characteristics satisfy a fixed specification — that is, conformance to an in beforehand defined specification.*

Meeting customer needs *Quality that is defined identified independent of any measureable characteristics. That is, quality is defined as the products or services capability to meet customer expectations — explicit or not.*

We consider these viewpoints to complement each other rather than being in conflict, since from a development project's perspective it is natural to develop a product according to some form of specification. However, from a customer's perspective it is natural to have the second viewpoint. In the end, it is how customers perceive product quality that determine how successful a product becomes. Note that there is one distinct difference between these two viewpoints on quality, the second

viewpoint is *subjective*, in the form of customer expectations, while the first is *objective*; assuming the specification conforms to good practices for software requirement specifications, e.g., as described in IEEE Std. 830 [IEE98].

A *quality model* defines "*the set of characteristics and the relationships between them which provide the basis for specifying quality requirements and evaluating quality*" [ISO01]. That is, a quality model can aid in transforming (subjective) customer needs into an (objective) specification which can be used in evaluating if a product fulfills the specification, and consequently (partially) aid in evaluating if a product meets customer needs. However, due to the problems of transforming customer needs to specifications it is usually necessary to involve end-users/customers during product validation.

Many quality models have been proposed over the years, a survey of different quality models is presented by Milicic in [LMW05]. However, as the survey discusses there are many similarities between the different proposed quality models. In the following description we have selected one of these models, namely the ISO/IEC Std. 9126 [ISO01], as an example.

ISO/IEC Std. 9126 describes three different kinds of quality: *quality in use*, *external quality*, and *internal quality*, which are described as follows in [ISO01]:

Quality in use *is the user's view of the quality of the software product when it is used in a specific environment and a specific context of use. It measures the extent to which users can achieve their goals in a particular environment, rather than measuring the properties of the software itself.*

External quality *is the totality of characteristics of the software product from an external view. It is the quality when the software is executed, which is typically measured and evaluated while testing in a simulated environment with simulated data using external metrics. During testing, most faults should be discovered and eliminated. However, some faults may still remain after testing. As it is difficult to correct the software architecture or other fundamental design aspects of the software, the fundamental design usually remains unchanged throughout testing.*

Internal quality *is the totality of characteristics of the software prod-*

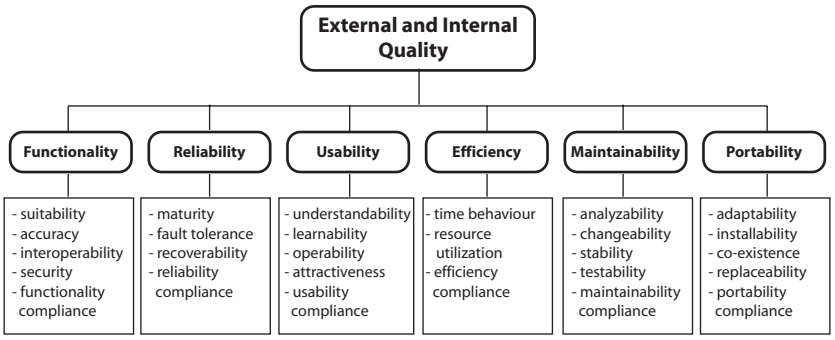


Figure 2.4: Characteristics and sub-characteristics for external and internal quality [ISO01].

> *uct from an internal view. Internal quality is measured and evaluated against the internal quality requirements. Details of software product quality can be improved during code implementation, reviewing and testing, but the fundamental nature of the software product quality represented by internal quality remains unchanged unless redesigned.*

For each of these qualities characteristics and sub-characteristics are defined. Using these characteristics simplifies, e.g., specification of quality requirements. An example of characteristics and sub-characteristics are presented in Figure 2.4. Instead of describing all the parts of the quality model in ISO/IEC Std. 9126 we provide a small example of one of the characteristics.

Example: Consider a company developing and producing ATMs that desires to specify that the ATMs should be "easy to use". This is a subjective statement and consequently not suitable to use as a product requirement, since it can not be objectively verified. There are of course several interpretations of what "easy to use" means for an ATM company and its users. One interpretation of easy to use in this context could be that the ATM should be able to expedite 100 money withdrawal transactions per hour, which is a measure of productivity according ISO/IEC Std. 9126. Having such a requirement can also place requirements on the usability of such a product, since to be able to support 100 money withdrawal transactions per hour it is required that the users can input

their pin code, amount to withdraw etc. within a relatively short time frame. Also, typical users of ATMs do not receive any training in using the ATMs, hence, the use of an ATM needs to be rather intuitive.

To be able to verify a requirement such as the one above it is required to test the ATM on typical users of the ATM, i.e., not the developers developing the machine, and perhaps also users not being familiar with ATMs, in order to validate that the requirement is fulfilled. This task can be simplified by using the usability sub-characteristics and suggested metrics defined in ISO/IEC Std. 9126.

Quality in relation to this thesis: Considering the quality of a company's existing product(s) is relevant during release planning, as a step in finding a suitable balance between investments in feature growth and quality improvements. This thesis argues that quality should be considered in the release planning process, but does not explicitly go into detail on how to do this, i.e., in this sense the thesis only describes *what* to do. However, as indicated in this chapter, using a quality model can aid both in specifying quality requirements and evaluating if quality requirements have been fulfilled.

2.4 Release Planning Process

Research within the area of process improvement is active, where the perhaps most well-known model and most used in practice is the CMMI [CMM06]. CMMI is focused on an organization's capabilities and maturity of running product development project(s). However, CMMI provides little detail on how to perform release planning. The CMMI process areas being related to release planning are *project planning*, *requirement development*, and *requirement management*. Within these process areas there are practices for capturing and managing requirements, but when it comes to how to select which features to include in the next release of a product there is no information or practice. CMMI in several cases merely states "resolve conflicts". However, even though CMMI may be lacking detail in some areas, still it is valuable.

There are also a number of standards which have parts related to release planning, for example, IEEE Std. 830:1998, 1220:2005, 12207:1996, and 15288:2002. IEEE Std. 830 specifies how a complete, correct, and non-ambiguous software requirement specification should be written. IEEE Std. 1220 specifies the tasks required throughout a system's life

cycle to transform stakeholder needs, requirements, and constraints into a system solution. IEEE Std. 12207 specifies a common framework for software life cycle processes, similarly IEEE Std. 15288 defines a framework for systems engineering life cycle processes. As these are standards, they rarely state how to perform a specified required task, instead they state that the task must be performed (somehow), i.e., CMMI and these standards provide little information concerning release planning.

In [BR89] Boehm and Ross describe the Theory-W, which is a theory aiding software project managers in creating win-win conditions among a set of (initially) conflicting goals. The task of a software project manager often is complex with many different stakeholders with different aims, for example, management usually have ambitious goals and do not like overruns nor surprises, customers typically want a quick schedule and a low budget, and the maintainers want well-documented software that is easy to change and no bugs. There are more stakeholders with similar goals, and as can be imagined, often there will be conflicts.

The theory W is "Make Everyone a Winner", i.e., the theory aims at avoiding win-lose and lose-lose situations, and thereby only having win-win situations. In creating win-win situations it is important to (i) separate the people from the problem, (ii) focus on interests, not positions, (iii) invent options for mutual gain, and (iv) insist on using objective criteria. In doing this it is important to understand how people want to win. There are more process steps involved in Theory-W, such as creating win-conditions, creating plans, following up plans. Theory-W can be seen as a negotiating approach which can be used during release planning. In a rather informal comparison in [Mea06], see Chapter 2.6, AHP receives a higher score than Theory-W. For example, Theory-W is determined to be more labor intensive than AHP. Tool support for Theory-W is discussed in, e.g., [BGB01, Grü00].

Another negotiating approach which can be used during release planning is *requirements triage*, i.e., the process of determining which requirements a product shall support given the available time and resources, which is discussed by Davis in [Dav03]. There are three main activities in triage: (i) establish relative priorities for requirements, (ii) estimate the resources needed to satisfy each requirement, and (iii) select a subset of requirements that optimizes the probability of the product's success in its intended market. Furthermore, to successfully perform triage input is required from at least three different kinds of stakeholders: customers, developers, and financial representatives. In the paper Davis presents

14 lessons learned concerning requirements based on experiences from roughly 100 companies, using examples from three of them.

J. Karlsson and Ryan [KR97] present a cost-value approach to prioritizing requirements. There are five steps in the process used in the cost-value approach:

1. Requirements engineers review candidate requirements for completeness and unambiguoity.

2. Customers and users, or suitable substitutes, apply AHP's pairwise comparison method to assess the relative values of requirements; the prioritization mechanism is discussed further in Chapter 2.6.

3. Experienced software engineers use AHP's pairwise comparison to estimate the relative cost of implementing each candidate requirement.

4. Each requirements relative value and implementation cost is then plotted in a cost-value diagram.

5. The stakeholders then use the cost-value diagram as an aid in deciding which requirements to include in the product.

Barney et. al. [BAW06] investigate how software product value is created and understood during release planning in two projects at one Australian outsourcing company. Their main goal is to provide insights into the release planning process used in industry. According to their study the decision making process seems to be a rather unstructured activity. Furthermore, they conclude that the maturity of a product has influence on the criteria used during release planning. Compared to the work in this thesis, we base our work on seven different companies where we, for example, have found relatively large differences in the maturity of the decision making process [Lin07], both structured and unstructured decision making processes have been observed. In addition, we have analyzed our data from several different perspectives, which we present in this thesis. However, our study is restricted to companies with relatively mature products.

In [RKH03] Regnell et. al. develop a theoretical model of the requirements selection process, in order to study: (i) how good a specific organization is in selecting the right requirements, (ii) determine what capacity, e.g., number of people investigating new needs, is required to

obtain a certain decision quality, and (iii) how long it takes to get good ideas to the market. Simulation is used for investigating different scenarios. A survey among product managers and system engineers is also presented in [RKH03] which indicate that less than 50% of products' requirements are correctly selected, which also is an indication of there being a need to improve the requirement selection process and reduce the number of incorrect decisions. In this thesis we basically address the same problem, but we are more process oriented.

Release Planning Processes in relation to this thesis: There is little information to be found in today's standards on how to conduct release planning. Still, release planning is an important area for a company in building a profitable business. In terms of other process related approaches to release planning, such as Theory-W and requirements triage, there are few that consider the entire range of problems involved in release planning and there are few that explicitly consider the existing product(s) during release planning, e.g., quality requirements for existing products. There exists more process-oriented work that have parts which are related to release planning, however, we consider the related work discussed in this section to be representative. Compared to other work we address release planning to a greater extent, perhaps especially how the existing system is considered during release planning. Our study also contains more companies than other similar studies, thereby providing greater possibilities for generalization of results.

2.5 Release Planning Algorithms

One illustrative example of formalizing the release planning problem is the work by Jung [Jun98]. In this approach the release planning problem is formulated as a *knapsack problem* [MT90]. The goal in the original knapsack problem is to maximize the number of items, each with a certain weight, that can fit into one bag on a trip, where the bag's weight capacity is limited. In Jung's approach the goal is to identify the set of requirements providing maximum value at minimum cost. These are the assumptions of the approch:

- Each requirement r_j provides, when implemented, some value v_j.
- Each requirement r_j has an associated development cost c_j.
- The budget for the release is b.

The problem is to assign requirements to a release such that Z_0 is maximized in Equation 2.1.

$$Max\ Z_0 = \sum_{j=0}^{n} v_j x_j \tag{2.1}$$

where x_j is a decision variable set to 1 if requirement j is included in the release, and set to 0 if requirement j is not included in the release. Equation 2.1 needs to be solved such that the constraint in Equation 2.2 is fulfilled, i.e., the selected requirements can be developed without exceeding the release budget b.

$$\sum_{j=0}^{n} c_j x_j \leq b \tag{2.2}$$

Ruhe and Saliu [RS05] has a similar approach as Jung [Jun98] to release planning, where the main difference is that the *value* and *constraint* expressions are more detailed in the work by Ruhe and Saliu. As a consequence it requires more expert judgments. In addition, Ruhe and Saliu considers dependencies between requirements/features and decides the content of the K next releases, instead of just one release, where K refers to the number of releases being planned.

The aim with formalizing the release planning problem often is development of tools. Saliu compares seven different release planning algorithms and their tool support in [SR05a] and evaluate how these consider the key aspects of release planning presented in [SR05a]; the compared algorithms are: PSERM, ENFEM, NRP, OVAC, COVAP, IFM, and EVOLVE. Just as in the case between Jung [Jun98] and Ruhe and Saliu [RS05] the main difference between the algorithms is in how many properties they consider.

Release planning algorithms in relation to this thesis: This line of work has closely related approaches. The main difference is in the refinement of details in the cost and value parameters in Jung's approach [Jun98]. There are also similar approaches which consider the existing system by including that as a parameter as well. The parameters in these approaches typically need to be estimated by experts. Our work is more process-oriented, and starts from the perspective of how things are conducted in industry today.

2.6 Requirements Prioritization

This section discusses different methods for requirements prioritization. An overview of prioritization methods is presented in [Mea06], namely *binary search tree*, *numeral assignment technique*, *100-point method*, *Theory-W*, *requirements triage*, *Wieger's method*, the *analytic hierarchy process*, and the *planning game*. In the following discussion we will touch upon most of these methods, but with a focus on those methods which in evaluations have been determined being most suitable. Some of the mentioned methods are more process oriented, these are covered in Chapter 2.4.

J. Karlsson and Ryan [KR97] presents a *cost-value* approach to requirements prioritization based on the Analytic Hierarchy Process (AHP) [Saa04]. AHP is a decision support method which helps in comparing different alternatives, requirements in this case. AHP uses pair-wise comparisons among a set of alternatives. A bit simplified, in each comparison you ask the question "is A better than B?". By performing a set of such comparisons it is possible to arrange the alternatives in priority order, since via the comparisons it is possible to compute the relative priority of each alternative. In the cost-value approach by J. Karlsson and Ryan they pair-wise compare first the *value* of each requirement, and then a second round of comparisons is made where the *cost* of implementing each requirement is compared.

In [Kar96] J. Karlsson compares two techniques for requirements prioritization, namely *pair-wise comparisons* and *numeral assignment technique*. Pair-wise comparison refers to the method described in the cost-value approach [KR97], while numeral assignment refers to a method where each requirement is assigned a numeral value which represents its value; similar numeral schemes are, for example, used in [KKC00, KAK02]. The two methods are compared in a case study at one industrial company. The conclusion from the study is that pair-wise comparison is faster than numeral comparison and that pair-wise comparison results in more accurate and informative results. For the numeral case the study participants often ended up in discussions concerning what was meant by each numeral, which was not an issue while performing pair-wise comparisons. The results from the study are interesting, however, the study only involved 14 requirements and only 3 people in the project that used both prioritization methods.

The 100-point method is basically a voting scheme mainly used dur-

Evalutation criteria	AHP	Hierarchy AHP	Spanning tree	Bubblesort	Binary search	Priority groups
Required number of decisions	78	26	12	78	29, 33, 38	34, 35, 36
Total time consumption	6	2	1	3	5	4
Time consumption per decision	2	4	5	1	6	3
Ease of use	3	4	2	1	5	6
Reliability of results	1	3	6	2	4	5
Fault tolerance	1	3	6	2	4	5

Table 2.1: Measures from evaluation of prioritization methods [KWR98].

ing brainstorming sessions [Mea06]. Each stakeholder is given 100-points which he or she can distribute among the set of alternatives being considered. The method is for example used in [KKC00], refer to Chapter 2.7 for more details. In essence the method is a variant of the numeral assignment technique. However, the method only works for an initial vote. In a second voting round stakeholders are likely to change their vote distribution such that their favorite alternatives become among the highest ranked [Mea06].

J. Karlsson et. al. [KWR98] compares six different methods for requirements prioritization: *analytic hierarchy process* (AHP), *hierarchy AHP*, *minimal spanning tree*, *bubblesort*, *binary search tree*, and *priority groups*. The study is limited to methods based on pair-wise comparisons, since the conclusion in [Kar96] is that pair-wise comparison is better than numeral assignment. The evaluation results are summarized in Table 2.1. J. Karlsson et. al. conclude that AHP is the most promising approach, since it results in the most trustworthy results based on a ratio scale, it is fault tolerant, and includes a consistency check. However, as they note, AHP may be problematic to scale up. A similar study by Perini et. al. [PSRB07] compares AHP with CBRanking, another prioritization method based on pair-wise comparisons, also finds AHP to have better accuracy.

In [KOR97] J. Karlsson et. al. improves the cost-value approach pro-

posed in [KR97] by (i) proposing ways of reducing the number of required pair-wise comparisons, (ii) extending the method to consider requirement dependencies, (iii) extending the method to support grouping of requirements. These improvements are aimed at making the cost-value approach useful in large-scale commercial development, e.g., the paper addresses the problem with scaling up AHP discussed in [KWR98].

L. Karlsson compares three requirements prioritization methods in [Kar06]. The compared methods are pair-wise comparisons [KR97], tool supported pair-wise comparisons, and the *planning game* (PG). The PG refers to the method in which requirements are prioritized in Extreme Programming (XP) [Bec99]; Extreme programming is an Agile process [Agi07]. In the PG the customer first prioritizes the requirements, or stories containing sets of requirements, using three priority levels: (i) essential functions, (ii) less essential functions but provide signigicant business value, and (iii) those that would be nice to have [Kar06, Bec99]. Developers estimate the effort for implementing each requirement, and sort the requirements by risk into three groups: (i) those that can be estimated precisely, (ii) those that can be estimated reasonably well, and (iii) those that can not be estimated.

The customer then prioritizes the requirements based on the time estimates and decide which requirements should be included in the next release. Releases in XP typically have relatively short release cycles. Hence, this process is repeated, say, every 4 weeks. Still, releases for use at customer location are released more seldom.

The conclusion from L. Karlsson's study [Kar06] is that the methods have similar accuracy, tool supported pair-wise comparison is less time consuming, still most of the subjects in the study preferred the PG. L. Karlsson recognizes the difference in result compared to, e.g., [KWR98], and recommends further study of the subject to settle these differences in results.

Requirements prioritization in relation to this thesis: The work presented in [KR97, Kar96, KWR98, KOR97, KKC00, KAK02] has guided the selection of the prioritization method to use in Paper D, refer to Chapter 9 for more information. The work in [KKC00, KAK02] prefer the numeral assignment technique, since they consider it to be faster than AHP; especially when there are many requirements, but they seem to have no objective comparison if this really is the case. However, there are methods that can reduce the number of required comparisons in AHP [KWR98, KOR97], but with less fault tolerance. There are

results that indicate pair-wise comparisons being faster and more reliable [Kar96, KOR97] than numeral assignment. Still, based on this research there is no strong proof of either of these methods being better than the other; for example, L. Karlsson's results suggest the planning game being better than pair-wise comparisons, still more research is required to confirm this. Based on our literature survey more results seem to be in favor of pair-wise comparisons. Furthermore, we have personal experiences from using pair-wise comparisons, which we consider also to be a good method for fostering discussions concerning alternatives, also discussed in [KWR98, KOR97]. The work in this thesis, specifically Paper D, requires a prioritization method. Based on the above related work we have selected pair-wise comparisons as prioritization mechanism, still it has not yet been shown which method is superior; it may even be the case that it varies depending on context.

2.7 Architecture-Centric Release Planning

In Chapter 1 we mentioned that the software architecture plays an essential role in meeting business critical requirements. In this chapter we provide a brief introduction to software architecture and discuss architecture-centric methods which possibly can be used during release planning.

There are several similarities between software architecture and building architecture. For example, a building architect uses multiple views of a building during discussions with its stakeholders, e.g., floor plans, exterior views, scale models, plus additional views with details for electrical wiring and plumbing [PW92]. Similarly, a software architect needs to express different views of a system to its stakeholders. Still there has been some debate during the last couple of decades concerning the exact definition of what software architecture is. A good, and well cited, compilation of different software architecture definitions is available at the web-site for the Software Engineering Institute [SEI05]. An extract of a relatively modern definition of software architecture follows:

> *The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.*

> *Externally visible properties refers to those assumptions other elements can make of an element, such as its provided services, performance characteristics, fault handling, shared resource usage, and so on.* [BCK03]

The software architecture has a profound impact on the achievable qualities for a product [BCK03]. When discussing software architecture and quality the terminology is slightly different compared to Chapter 2.3, often we refer to *quality attributes* instead. However, the external and internal quality discussed in Chapter 2.3 (see Figure 2.4) has many similarities with the taxonomy typically used for quality attributes, therefore we do not go into further detail on these.

Architectural analysis can be used as an aid in achieving the desired quality attribute requirements for a system. A survey of different methods for architectural analysis is presented by Dobrica in [DN02]. As is concluded in the survey the Architecture Tradeoff Analysis Method (ATAM) [KKC00] is the most well-known and most used method for architectural analysis.

The Architecture Tradeoff Analysis Method (ATAM) [KKC00] is a method for identifying the architectural tradeoffs that exist among requirements on quality attributes. For example, having high performance and at the same time having high flexibility can be conflicting goals, since typically incorporating flexibility is often done at the price of lower performance. Compared to other work discussed in this chapter this work is more technically oriented and not specifically aimed at release planning, still there are some related areas. In a way the ATAM, combines the technical aspects of a product with the business aspects by:

- Systematically eliciting the requirements having significant impact on the software architecture of a product, typically these requirements are requirements on quality attributes.
- Identifying the high priority scenarios, or sets of requirements, which are business critical.
- Identifying tradeoffs among the quality attributes and develop a software architecture concept that can support the quality attribute requirements.

There are nine steps in the ATAM which typically are conducted as workshops over three days, with a break of several weeks after the first

day. A similar approach to ATAM, at least in the respect of eliciting business critical quality attribute requirements, is the Quality Attribute Workshop (QAW) [BEL+03]. Both the ATAM and QAWs explicitly capture quality attribute requirements, these are the requirements which typically have largest impact on the architecture of a system.

More closely related to release planning is the Cost-Benefit Analysis Method (CBAM) [AKK01, KAK02], which builds upon the ATAM [KKC00]. There are nine steps involved in carrying out a CBAM exercise. In summary, scenarios expressing important quality aspects are elicited and prioritized using the 100-point method (see Chapter 2.6). The scenarios are ranked based on the voting, after which the top 50% scenarios are analyzed in more detail. *Utility* is assigned to the top 50% scenarios by estimating their worst-case, current, desired, and best-case quality response level. One of the next steps in CBAM is to develop architectural strategies, i.e., technical solutions, and by interpolation determine the solutions expected response on the utility for each scenario; an architectural strategy may have impact on more than one scenario. Finally, cost and schedule estimation is performed for each architectural strategy and the return-of-investment (ROI) is computed. The architectural strategies are ranked based on their ROI and the highest ranked which can fit within the budget are selected for implementation.

Regnell et. al. [RSO08] present a method on how to consider the quality of the existing system during roadmapping/release planning. One of the primary steps of the method is to capture the benefit of having a certain level of quality within different areas, e.g., performance; see Chapter 2.3 for a discussion on software quality. This is done by estimating different breakpoints in terms for different quality indicators. For example, below a certain level of quality the product will be useless to customers, hence, at some level of quality the product becomes useful. Another quality level is where the product exceeds the competition and at some point there is a level of quality that is excessive to customers, as illustrated in upper left of Figure 2.5.

The next step in the method is to estimate the cost of reaching a certain level of quality for the product, as illustrated in upper right of Figure 2.5. The third step consists of estimating the current level of quality of the product and combining this with the cost and benefit view. These different graphs, for different quality indicators, then guide the release planning decisions; illustrated in lower part of Figure 2.5.

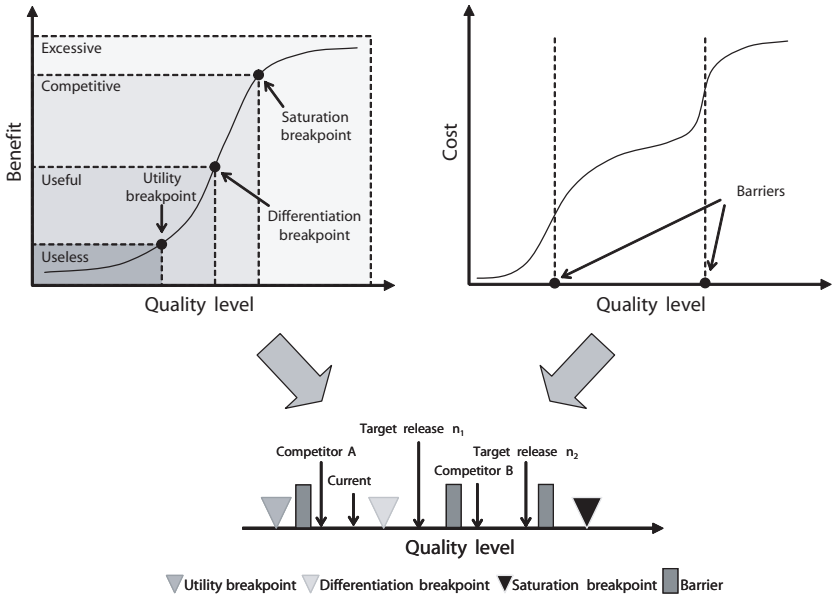


Figure 2.5: Roadmapping using quality indicators [RSO08].

Architecture-centric methods in relation to this thesis: Considering the software architecture of a company's existing product(s) is relevant during release planning, since the existing architecture, e.g., has impact on the effort required for adding new features. From the point of view of this thesis software architecture is relevant to consider in the release planning process, however, we do not require the use of any particular analysis technique or method.

The approaches presented in this section are more technically oriented and specifically address the quality aspects. Compared to the other related work described in this chapter, ATAM and CBAM spends more effort on how to find technical solutions that fulfill elicited quality requirements, while still paying attention to the business perspective. Even though these approaches are not aimed at release planning they probably can be used for part of the problem. However, using these approaches for release planning requires an architecture-centric organization.

Chapter 3

Research Method

This chapter describes the method used while conducting this research. However, before describing the method we first provide an introduction to: research in general, quantitative research, qualitative research, validity, case studies, and grounded theory research. It is not intended for this overview to be complete, but rather it aims at providing a basic understanding of some of the concepts and tradeoffs involved when conducting software engineering research. Hopefully this overview makes it easier to digest the research method presented in Chapter 3.5. We conclude the chapter with a description of how threats to validity have been addressed in our research, Chapter 3.6.

3.1 Traditional Research

The aim of research in general is advancement of human knowledge. Over the years there have been several researchers making significant contributions to human knowledge, such as, Newton (laws of motion and law of gravitation), Kepler (planetary motion), Faraday, and Ohm. There are many different areas of research, yet the scientific method used when conducting research can have, and often do have, many similarities. Historically it seems that scientific methods were originally developed when conducting physical research in the 17^{th}, 18^{th}, and 19^{th} century, which we in the following discussion refer to as traditional research [Rob02].

Traditional (natural science) research is normally quantitative in its nature. For example, in school we are taught how to perform experiments

to verify early research results in physics, for example, from Newton's law of universal gravitation and Newton's law of motion we can derive Earth's gravitational constant g, which varies over Earth's surface; it is roughly $9.8m/s$. In performing experiments with purpose of verifying such results we typically collect a series of samples, where each sample contains a measurement of the relevant variables. After we have collected a sufficiently large sample set we can plot the collected samples in a graph, and (possibly) derive a mathematical formula describing how the relevant variables depend on each other. Furthermore, when measuring the relevant variables there typically is some measurement noise which need to be considered when deriving the mathematical formula, e.g., when measuring time using a manual stop watch our accuracy is limited.

In the above example we use an experiment to validate a model. Hence, before conducting an experiment, whose purpose is to validate a model, there is a constructive phase during which the model itself is built. There are different ways in which models can be built, but typically these are derived part from previous knowledge and experience and part from a creative process. There are also experiments whose sole purpose is to gain experience within an area, so called *exploratory* experiments.

Experiments are generally accepted as being a scientifically sound research method. However, this form of research method is not always suitable. For example, an experiment requires that the relevant variables: (i) are known, (ii) can be controlled, and (iii) can be measured. What if the relevant variables can not be controlled? What if there is no way to quantitatively measure the relevant variables? What if the relevant variables are not known? If the answer is no to all these questions, then we have to resort to *qualitative* methods.

3.2 Qualitative Research

When there are no established quantitative measurements researchers need to resort to qualitative judgements. When judgements come into play there is higher risk of introducing researcher bias. Qualitative research is not always accepted by researchers working with traditional research methods, since some of them consider there being too many subjective judgments that potentially invalidate the research results [Rob02]. On the other hand, researchers working with qualitative research can address this by careful attention to *threats to validity* and by introduc-

ing proper counter measures. The threats to validity can be described as [Yin03]:

Construct validity is concerned with correctly collecting/measuring the relevant variables, i.e., the accuracy and completeness of data.

Internal validity is concerned with how data is analyzed and how relationsships between data are established.

External validity is concerned with establishing the limits within which the results are applicable.

Reliability is concerned with how a research study can be reproduced. To be able to produce new knowledge it is important that other researchers can reproduce the results of each other.

Both quantitative and qualitative research have threats to validity.

One may wonder why there are such differences between how research is conducted in different areas. One explanation is the maturity of the research area itself, as is discussed in [Sha02]. Simply put, each research area goes through different phases. In the beginning there may be anecdotes, next rules of thumb may emerge, and as the area further matures it may be possible to at a later stage use more traditional research methods.

Research in software engineering covers a broad range of topics, where some of the areas are mature and can use quantitative research methods, while some areas are less mature and often require the use of qualitative research methods. For example, performance computing is an area where it is suitable to use quantitative methods, while the research in this thesis in our opinion requires qualitative methods. Several of the areas in software engineering which are less mature are related to people and processes, rather than efficiency of algorithms.

The main part of this thesis is concerned with qualitative judgements, which are supported by systematically collected data and observations, i.e., the focus is on gaining more experience and insights into the area.

3.3 Case Studies

This chapter provides a brief introduction to some of the aspects involved in conducting case study research, more detailed information is

available in [Rob02, Yin03]. Case studies are suitable when studying a contemporary phenomena within some real-life context, where the topic is complex and contains many, and possibly unknown, variables that can not be controlled [Yin03]. Case studies are generally the preferred research strategy when the research question is of the form "how" or "why". In contrast to experiments, case studies do not require control of the relevant variables, i.e., does not require control of behaviour. Historically case studies where first used as a research strategy in social science research [Yin03].

There are three different types of cases studies: *explanatory*, *exploratory*, and *descriptive*. The names of these different types of case studies reflect their purpose. It is possible to have a case study where some elements are explanatory and some exploratory.

The data collected and analyzed in case studies come from six types of sources: documents, archival records, interviews, direct observation, participant-observation, and physical artefacts [Yin03]. When comparing these data sources with sources commonly used in traditional research there are some rather strong differences. For example, when collecting most of these data it is possible for subjective judgements to be introduced, which preferably should be avoided. Still, subjective judgements is not a "bad thing", as long it is possible for different researchers to reach the same, or similar, conclusions. However, it is not good if different researchers reach contradictive conclusions using such data. In addition, it is hard to apply the same analysis methods as used in traditional research for data collected using subjective judgements. Again, if not dealt with properly there is possibility of researcher bias to be introduced.

In traditional (quantitative) research sample size is normally chosen such that statistical confidence can be achieved. Case studies, however, seek to choose a sample set from which general results can be derived, i.e., a representative sample set is desired. In terms of sample size there are two kinds of case study designs: *single case* and *multiple case*. In a *single case* case study it is hard to generalize the results beyond the case being studied, e.g., a company. Still, a single case design may provide in-depth understanding of the particular case. In a *multiple case* case study there are possibilities of fleshing out results generally applicable in the sample set. Furthermore, if the sample set is representative it may be possible that the results are generally applicable. Researchers normally desire results being generally applicable, hence, generality is a

good thing. However, there is a trade-off involved here. When choosing a too small sample set there is risk of the results not being generally applicable. When choosing a too large sample set there is risk of not having time to study each case in sufficient depth to identify the important factors.

The goal with *replicating* case studies is typically to investigate if the results developed in a previous case study is applicable on another sample set. In case the results still holds on this new sample set, then it strengthens the validity of the results.

Case studies are common as a research methodology in software engineering research, as is indicated by these references [HR07, BT03, DTB05, vKR05, BAW06, SA05].

3.4 Grounded Theory Research

Research methods can basically be divided into top-down and bottom-up approaches. A top-down approach first defines a model and then attempts to validate the model using real-life data. To use a top-down approach requires (i) the domain to be relatively mature, and (ii) a simple model or a hypothesis which can be falsified. A bottom-up approach, on the other hand, uses real-life data, analyzes this data, and as analysis and understanding of the data progresses a model is developed. Usually the model is successively refined. *Grounded theory research* [Rob02, SC98] is a bottom-up approach to research.

In building a model using grounded theory research it is important that the data set is sufficiently large such that it contains all the relevant aspects, i.e., the sample set should be representative. Selecting the correct data set is important when conducting case studies, as discussed in Chapter 3.3.

To validate a model built using grounded theory research, it requires the use of new data, i.e., data not belonging to the original data set used in developing the model. This is natural, since per construction the developed model will be valid on the data set used to construct it. Consequently, a new data set is required during validation.

This thesis uses a research approach based on grounded theory. With this we refer to how the models are constructed, i.e., the methods/models presented in this thesis are mainly derived from real-life data. We have not strictly followed all the analysis techniques as required in grounded

theory research [SC98]. More specifically, we have not iteratively collected data and built our model until saturation has been achieved within each category as mandated by grounded theory. Furthermore, we have not followed the three analysis steps required in grounded theory: *open coding*, *axial coding*, and *selective coding*.

3.5 Method

Here we describe the steps carried out throughout this research effort:

Problem Formulation The first phase of this research was to identify a relevant research problem. Initially the goal was to identify a research problem related to improvement of a systems software architecture, and therefore the initial focus was on studying existing architecture literature. However, since architectural improvements usually are expensive it becomes necessary to also consider business aspects of development, e.g., adding new features which potentially can increase revenues.

Eventually we reached a state where we could formulate a research aim, namely to develop guidelines and/or methods that improve release planning by considering the software architecture of an existing system. However, the research questions, as stated in Chapter 1.2 where defined later, but still they reflect the process which has been followed in this research.

Multiple Case Study A multiple case study design involving seven industrial companies was devised whose initial purpose was to confirm the relevance of the stated research aim. To simplify comparison of the release planning processes at the different cases we used the release planning key aspects proposed by Saliu and Ruhe [SR05a], but there were also exploratory parts in the study.

Performing the interviews in the study and their documentation, see [Lin07], was a rather time consuming task. It should be noted that during the analysis phase we discovered that we had collected more useful data than initially expected. There were also larger differences between the companies than initially expected.

Analysis The analysis phase started after completing the data collection phase. In this phase the data was analyzed from different

perspectives. The first result from this analysis was Paper A, see Chapter 6. A natural step, which was not initially thought of, was to validate the key aspects proposed by Saliu and Ruhe, since we had already collected all required data. This analysis resulted in Paper B, see Chapter 7.

In our collected data we captured the state-of-the-practice for release planning. When looking at the different practices used at the different companies we realized differences, for example, some of the practices had difficulties in addressing certain problems while others had not. By ranking the observed practices, partly validated by interviewees participating in our study, we could identify a set of best-practices in Paper C, see Chapter 8.

Release Planning Method Development Now the research entered into a new phase, a constructive phase. Based on the insights and knowledge gained from Paper A–C we now had sufficient understanding to develop a useful and relevant method aiding in performing release planning. The result of this constructive phase is Paper D, see Chapter 9.

PhD Thesis The last step of this research is in writing this PhD thesis. It is not until this step that we put more thought on what actually led us to the possibility of formulating a method. Hence, the research questions in Chapter 1.2 were formulated after the research was complete, but they reflect the actual process that has taken place. That is, by investigating and answering a number of questions in Paper A–C we gained an understanding of release planning that enabled us to develop a method, Paper D.

3.6 Validity

When conducting research there are threats to validity that need to be addressed. In this section we describe what we have done to counter these threats. We have followed the recommendations by Yin [Yin03] for multiple case studies. In our study we have used semi-structured interviews as the primary data collection method, sometimes complemented by documents received from the interviewees. The main alternative, direct observations, has not been used due to the topic being studied

containing company sensitive information and partly due to practical limitations.

We have addressed *construct validity* in multiple ways. First, there have always been two researchers present during each interview in order to reduce risk of misunderstandings. Second, interview notes have been taken during each interview, which have been sent to interviewees for approval. These notes have later been merged to a single description per company [Lin07]. Third, we have had two test interviews to improve our interview setup. In total we have interviewed 16 people (excluding test interviews), typically 2–4 persons per company to achieve data triangulation.

We have considered the possibility of using audio recording equipment during our interviews. However, we have chosen to take notes during interviews instead, since (i) there is risk of interviewees feeling threatened by recording equipment and (ii) transcribing recorded audio is a time consuming task.

To strengthen the *internal validity* of our study we have used multiple researchers when performing analysis, and we have considered rival explanations. To increase *reliability* of our study, all collected data, and derivations thereof, are stored in a database accessible only to the researchers in the study, e.g., interview notes and merged notes per company. In addition, the study design is documented, which includes the interview questions. Using this material it is possible to trace conclusions to collected data, and vice verse.

Thanks to industrial contacts we have been able to find a relatively large number of companies (7) and persons (16) willing to participate in our study, which aids in increasing the *external validity* of our results. Still, there is risk that the selection is not fully generalizable to other domains and/or nationalities/cultures. Choosing the number of companies to study is a trade-off. When using too few companies it can be hard to generalize the results. When using too many companies it can be hard to reach a sufficiently deep understanding for each case, due to resource limitations. In this respect we consider seven companies to be a reasonable trade-off.

To counter *researcher bias* we have been multiple researchers at most of the steps of this study. Furthermore, companies that the researchers in this study are affiliated with are excluded from the study.

Chapter 4

Research Results

This chapter presents an overview of the research results presented in Paper A–E and describes how the results are related to each other, with aim of demonstrating that these papers form one cohesive research effort. More detailed information concerning each paper's results is available in the respective paper chapter (Chapter 6, Chapter 7, Chapter 8, Chapter 9, and Chapter 10). We start this chapter by providing an overview of how the papers are related to each other and conclude by revisiting the research questions stated in Chapter 1.2.

4.1 The Papers in Context

Paper A

Paper A investigates release planning from a software architect's perpective. The study is based on a multiple case study involving seven companies, i.e., the study which the main parts of this thesis is based on. In essence there are three main lessons learned from this paper:

1. The practices used during release planning has impact on the possibilities to early address architectural issues; details in Chapter 6.5.1.

2. The architectural awareness among product management generally is low; details in Chapter 6.5.2.

3. There seems to exist no method, or even rules of thumb, on how to decide the balance between how much to invest in feature growth and quality respectively; details in Chapter 6.5.3.

Based on these results there are a number of implications. First, it is relevant to in more detail study the different practices used during release planning to understand their strengths and weaknesses better (addressed in Paper C). Second, an architecture-centric method for release planning needs to involve more stakeholders during release planning, since product management does not have sufficient knowledge in architectural issues, issues which are essential to address in order to be able to meet business critical requirements (addressed in Paper C and Paper D). Third, based on this work there seems to be a need for a method or guideline for how to decide the balance between investments in feature growth and quality (addressed in Paper D).

Paper B

Paper B investigates the key aspects of release planning. A deep understanding of these aspects is required if we are to develop a useful method that aids in the release planning task. The approach taken in this paper is to use the key aspects proposed by Saliu and Ruhe [SR05a] as a reference model, and attempt to validate those aspects using the cases in our multiple case study and possibly identify new aspects, or remove existing aspects from the model. In terms of our aim to improve the release planning process this paper's main contribution is a better understanding of what is important during release planning.

In summary, the key aspects of release planning are:

Objectives There needs to be some objective of the release planning task, e.g., the direction which we are striving for with the product. These objectives can, for example, be captured in a product strategy.

Resource constraints In all development there are resource constraints that needs to be considered, e.g., the number of available developers. If no consideration is taken to such constraints it is highly likely of ending up with an unrealistic plan.

Technological constraints There may be dependencies between needs which need to be considered during release planning. For exam-

ple, before adding feature x it may be required that feature y is available.

System constraints The existing software, and its architecture, often impacts the effort, risk, and feasibility of adding/improving features.

Time horizon How often releases are made has impact on what can fit within a plan. An organization can either have a fixed or flexible release schedule. In a fixed release schedule releases are released at predetermined intervals.

Stakeholder involvement A stakeholder is any person or organization that has interest in the release. Typically planning can not consider all stakeholders to an equal degree, instead it is important to at least consider the significant stakeholders.

Short- and long-term planning Planning needs to be done so that both short- and long-term aspects are considered. For example, to remain profitable in the long-term it is not good to spend all resources on new features; at least some resources need to be spent on quality aspects.

A more complete description of the cases' release planning processes is available in [Lin07]; in a sense the report contains a description of state-of-the-practice for release planning in industry. What we have observed is that several of the companies in our study strive for their release planning process to provide:

Flexibility Flexibility is required in the release planning process, e.g., enable a short time-to-market by having a release planning process that allows relatively late discovered needs to be included in a controlled manner.

High utilization High utilization of the R&D organization.

Risk reduction Risk reduction by incremental decisions.

Improved planning Improved planning by stepwise refining knowledge concerning proposed needs.

Paper C

This paper proposes a capability model for some of the key ares of the release planning process. Using such a capability model it is possibible for a company to identify process improvement possibilities. In performing this work, again, the reference model by Saliu and Ruhe [SR05a] aided the comparison of different practices used at the companies. Basically the proposed capability model is devised by ranking the observed practices within different key areas based on the understanding gained from Paper A and Paper B, i.e., understanding the key aspects and some limitations of how planning can be conducted. For example, one limitation we reported is that product management has low architectural awareness.

The highest levels in the capability model reflect organizations capable of incrementally refining needs through pre-studies and feasibility studies, with decision points before and after these studies that allow the needs providing greatest return-of-investment to be identified. These studies increase the certainty of estimations made concerning the proposed needs, and thereby aid in improving the objectiveness during decision meetings. Of course, the effort dedicated to these studies needs to be adapted to the estimate of the total effort to realize the needs. That is, less time should be spent on detailing a simple need, when compared to a more complex need. Another important point, is to incrementally plan the release, especially if the release cycle is relatively long, say more than 6 months, since during such time spans new needs may arise which were not initially thought of.

Paper D

Paper D proposes a method for how to decide the balance between investments in feature growth and quality. The earlier results from Paper A–C has strong influence on this method. For example, from Paper A we learned that different stakeholder groups must participate since these have better understanding of their respective area, i.e., product management do not have the required skills to make all required judgements. From Paper C the ranking of the observed practices provides further input. From Paper D it is mainly the highest ranked practices that have influenced the proposed method.

Figure 4.1 illustrates how the results from Paper A–D are related to

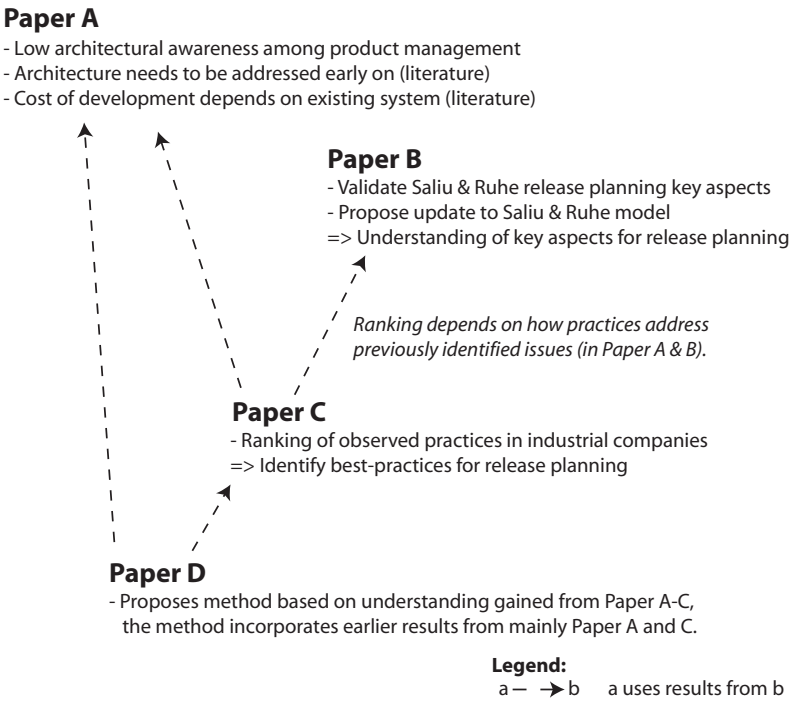


Figure 4.1: Overview of how research results in Paper A–D influence each other.

each other. As can be seen in the figure, the results and experiences from Paper A–C are used in developing the method proposed in Paper D.

Paper E

Paper E is an extended version of Paper C where parts of Paper D have been integrated; Paper C was invited for journal publication after its presentation at PROFES'08. Consequently the main research results from Paper E have already been covered in Paper C and Paper D.

4.2 Research Questions Revisited

In this section we revisit the research questions stated in Chapter 1.2 and provide answers to the questions. As a reminder, the research questions have been formulated such that by answering these questions we will obtain sufficient knowledge for developing a method that can aid in deciding the balance between investments in feature growth and quality.

> **RQ1:** *What are the key aspects of software release planning in industry?*

The first research question is mainly addressed by Paper B. The identified key aspects are: *objectives*, *resource constraints*, *technological constraints*, *system constraints*, *time horizon*, *stakeholder involvement*, and *short- and long-term planning* (see details in Chapter 4.1 and Paper B).

> **RQ2:** *What are the (observed) best-practices for release planning in industry?*

The second research question is primarily addressed by Paper C, with some contributions from Paper A. We have identified best-practices within the following key areas: *need elicitation*, *need documentation*, *decision material*, *product strategy*, and *release plan decision*. Paper C expresses the observed practices as a capability model, where usually each capability level complements the earlier level, i.e., higher capability levels still have elements of the earlier levels incorporated into them. A description of the best-practices within each of the key-areas follows (details are available in Chapter 8.4):

Need Elicitation Needs are elicited in multiple ways, for example, there are more or less adhoc parts of this elicitation phase as well as structured activities. As part of the *adhoc* activities are the needs elicited by product management during (informal) meetings with customers and different stakeholders of the product. In addition to this there should be a *formal path* in the release planning process for the significant stakeholders to inform of needs. Furthermore, before delivering the set of needs to product management the needs should be prioritized according to the product strategy by the stakeholders. As a result, product management will receive a set of prioritized need lists; one from each stakeholder group. The task of product management is to identify a suitable balance between these lists.

Need Documentation Needs should be documented according to a common company template, and these should be stored in a tool such that the people involved in need elicitation and release plan decisions can access them. Needs should be classified as being either *new feature*, *quality improvement*, or *cost-cut*.

Decision Material The process used for developing decision material concerning proposed needs is key in being able to step-wise improve certainty in information concerning needs, and to aid in risk reduction. Needs should be refined using both *pre-studies* and *feasibility studies*, but before any such studies are commenced a first round of need prioritization shall be performed by product management (which reduces the needs being candicates for going through pre-studies). Pre-studies are started for the needs with highest priority with purpose of detailing the need and refining cost and business implications. The result of a pre-study is returned to product management after which a second round of prioritization can be performed. The needs which still are found interesting to proceed with go through feasibility studies, which are more focussed on technical solutions and further refinement of cost estimates. After the feasibility study a third round of prioritization is done resulting in the final set of needs for implementation. Preferably, pre-studies and feasibility studies should be carried throughout the release project, thereby resulting in step-wise decisions concerning the release plan.

Product Strategy In terms of product strategy we have not collected

sufficient information to develop reasonable capability levels. Still, having well-defined and clearly communicated product strategy often helps employees within an organization to know what to strive for. In our study Case 1 and Case 2 has the clearest communicated product strategies.

Release Plan Decision The release plan decision is in large parts based on the earlier developed decision material, the product strategy, and the business strategy. It seems that the use of decision support tools, such as Focal Point [Foc08], aid in reducing, e.g., political factors, during these decisions.

RQ3: *Given the answers to RQ1 and RQ2, how should a company find a balance between investments in features and quality?*

The third research question is primarily addressed by Paper D, which is heavily influenced by the results and experiences from Paper A–D. Several of the steps in the method originate from practices used at two of the companies in our multiple case study, namely those companies that were considered to have the most capable release planning process in Paper C. The outline of the proposed method is as follows:

1. Each stakeholder group elicit needs within their areas. For example, architects elicit/identify software architectural activities aimed at improving the quality of the product, and product management elicit (mainly) new product features.

2. Each stakeholder group independently prioritize their needs. This can be seen as a *local* prioritization of needs.

3. Harmonize the prioritized set of needs from the stakeholder groups, if such a need exists.

4. Upper-management makes a conscious tradeoff between the set of (locally) prioritized needs, also considering the business strategy. This can be seen as *global* prioritization of the proposed needs which in essence decides how much resources to invest in each area.

Details of the method can be found in Chapter 9.5.

Chapter 5

Conclusions

The goal of this thesis is to improve the software release planning process of an organization, with a specific focus on how to decide the balance between investments in features and quality. Release planning is concerned with the task of deciding what features and improvements to include in future release(s) of a product, basically with the goal of optimizing company profit in both short- and long-term.

To develop products effectively, and be profitable, one must: (i) correctly identify different stakeholders' needs, (ii) identify those needs which have greatest return-of-investment, and (iii) to efficiently be able to transform the need statements to project requirements and realize those as part of a product. Sadly, in today's development of software intensive products this process often is performing worse than desired. This thesis is focused on improving some of the earlier parts of this process, often by both considering business and technical aspects.

This thesis is in large parts based on a multiple case study involving seven industrial companies. The publications included in this thesis basically present research results that have be obtained by analyzing the case study data from different perspectives, and/or by refining results from earlier papers. From an industrial perspective the main contributions of this thesis are: (i) a proposal for a capability model for release planning and (ii) a method aiding in deciding the balance between investments in features and quality. Each of these contributions are summarized below:

1. *Capability model.* In this thesis we propose a capability model for release planning processes, which both can aid in assessing the

capabilities of an organization's processes and in identifying improvement possibilities.

2. *Decide balance between features and quality.* We propose a method for deciding the balance between investments in features and quality. First, of all such methods or even rules of thumb are lacking in industry, hence, it fills a missing gap. Second, this method is in large parts based on observed data and therefore is probably easy to apply in industrial companies. Yet, validation of the method remains; the method can not be validated using the same set of data which it was derived from.

From an academic perspective the main contributions of this thesis are: (i) validation of release planning *key aspects* proposed by Saliu and Ruhe [SR05a], (ii) identification of product management generally having low architectural awareness, and (iii) capture of state-of-practice. We have validated six of the key aspects proposed by Saliu and Ruhe and one additional key aspect is identified (short- and long-term planning). We conclude that product management generally has low architectural awareness. Hence, to address architectural issues early the release planning process must involve technically oriented stakeholders, e.g., software architects. Via our relatively large set of cases in the study we have captured state-of-the-practice for release planning in industry, which is useful to other researchers.

This research effort has been conducted using sound and rigorous research approaches suitable in this area of research. This includes development of a case study design, properly following data collection methods for interviews, documentation of interviews, and followed by proper analysis. One of the strengths of this work is the relative large number of companies involved in the study, which increases the study's validity. Even though the study is primarily targeting companies developing software intensive and long-lived products, the results are of such kind that they are probably generalizable outside this product domain. Still, there is a dominance of Swedish/western cultures in the study, which may possibly render the results less useful in other cultures.

Finally, to summarize this research effort, through our multiple case study on release planning in industry we have reached an in-depth understanding of the relevant problems, and methods for addressing the problems in an industrial setting. The research represents a significant effort in this area.

5.1 Future Work

Here we present a sample set of possible future works resulting from the work in this thesis. The results from Paper A, and especially the conclusion that the architectural awareness among product management is low, requires a quantitative follow up study to improve the validity of these results. Possibly there can be further relevant key-aspects which remain to be identified. In addition, there may possibly be differences in what is considered being key in different product domains.

The capability model proposed in Paper C can be further refined, e.g., detailing how needs should be documented and detailing desirable results from pre-studies. Furthermore, the capability model is mainly developed using data from companies not applying Agile development models, refinement of the model to also include Agile development methodologies would be interesting. The capability model also remains to be validated, which can not be performed on the same cases as it was derived from, i.e., more cases studies are required to validate the results.

It is possible that architecture-centric approaches, such CBAM, can be used as an aid for release planning. However, these require architecturally mature organizations in order to be adopted. For immature organizations it may be interesting to investigate what is required for an organization to adopt architecture-centric approaches.

The main data from the case study is so far only published as a technical report, with extracts being used in Paper A–E, still it may be interesting to rework this material and publish it in a different forum such that the current state-of-the-practice for release planning receive more attention in the research community. For example, there are findings related to late detection of project dependencies in release projects, in this study a release project can consist of 20–30 sub-projects, which we as of yet has not published as a finding. The method proposed in Paper D remains to be validated, even though it has been derived from observed data; again, the same cases can not be used in validating the results.

Release planning is concerned with identifying the needs providing greatest return-of-investment, yet there are few, if any, metrics which can be used in measuring the success of this process. There are several metrics which are used for development projects, which for example can be used in determining the value of project managers' work; both for identifying good and poor project managers. But how do we tell if a product manager is doing a good job or not?

Bibliography

[Agi07] What Is Agile Software Development? Web-site: http://www.agilealliance.org, 2007.

[AKK01] Jayatirtha Asundi, Rick Kazman, and Mark Klein. Using Economic Considerations to Choose Among Architecture Design Alternatives. Technical Report CMU/SEI-2001-TR-035, Carnigie Mellon — Software Engineering Institute, 2001.

[BAW06] Sebastian Barney, Aybüke Aurum, and Claes Wohlin. Quest for a Silver Bullet: Creating Software Product Value through Requirements Selection. In *Proc. 32nd Euromicro Conference on Software Engineering and Advanced Applications*. IEEE Computer Society, 2006.

[BCK03] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice 2nd edition*. Addison-Wesley, 2003.

[Bec99] Kent Beck. *Extreme Programming Explained — Embrace Change*. Addison-Wesley Professional, 1999.

[BEL+03] Mario R. Barbacci, Robert Ellison, Anthony J. Lattanze, Judith A. Stafford, Charles B. Weinstock, and William G. Wood. Quality Attribute Workshops (QAWs). Technical Report CMU/SEI-2003-TR-016, Carnigie Mellon — Software Engineering Institute, August 2003.

[BGB01] Barry Boehm, Paul Grünbacher, and Robert Briggs. Developing Groupware for Requirements Negotiation: Lessons Learned. *IEEE Software*, 18(3), 2001.

[Boe02] Barry Boehm. Get Ready for Agile Methods, with Care. *Computer*, 35(1), 2002.

[BR89] Barry W. Boehm and Rony Ross. Theory-W Software Project Management Principles and Examples. *IEEE Transactions on Software Engineering*, 15(7), 1989.

[BT03] Tomas Berling and Thomas Thelin. An Industrial Case Study of the Verification and Validation Activities. In *Proc. 9^{th} International Software Metrics Symposium*, pages 226–238, Sept. 2003.

[Car04] Pär Carlshamre. Release Planning in Market-Driven Software Product Development: Provoking an Understanding. *Requirements Engineering*, 7(3), 2004.

[CMM06] CMMI Product Team. CMMI for Development, Version 1.2. Technical Report CMU/SEI-2006-TR-008, Carnegie Mellon — Software Engineering Institute, 2006.

[Con01] Larry Constantine. Methodological Agility. *Software Development*, pages 67–69, 2001.

[Dav03] Alan M. Davis. The art of requirements triage. *IEEE Computer*, 36(3), 2003.

[DeG00] Gary DeGregorio. Technology Management Via a Set of Dynamically Linked Roadmaps. In *Proc. of the 2000 IEEE Engineering Management Society*, 2000.

[DN02] Liliana Dobrica and Eila Niemelä. A Survey on Software Architecture Analysis Methods. *IEEE Transactions on Software Engineering*, 28(7), 2002.

[DTB05] Trung T. Dinh-Trong and James M. Bieman. The FreeBSD Project: A Replication Case Study of Open Source Development. *IEEE Transactions on Software Engineering*, 31(6):481–494, June 2005.

[FBB+99] Martin Fowler, Kent Beck, John Brant, William Opdyke, and Don Roberts. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional, 1999.

[Fis98] M. Fischer. Software Architecture Awareness and Training for Software Practitioners. U.S. Army CECOM Course, June 1998.

[Foc08] Telelogic Focal Point. Web-site: http://www.telelogic.com/focalpoint, August 2008.

[Grü00] Paul Grünbacher. Collaborative Requirements Negotiation with EasyWinWin. In *Proc.* 11^{th} *International Workshop on Database and Expert Systems Applications*. IEEE, 2000.

[HR07] Martin Höst and Per Runeson. Checklists for Software Engineering Case Study Research. In *Proc.* 1^{st} *International Symposium on Empirical Software Engineering and Measurement*, pages 479–481, Sept. 2007.

[IEE98] IEEE. IEEE Std 830:1998, IEEE Recommended Practice for Software Requirements Specifications, 1998.

[ISO01] ISO/IEC. ISO/IEC Std 9126:2001, Software engineering — Product quality, 2001.

[JBR99] Ivar Jacobson, Grady Booch, and James Rumbaugh. *The Unified Software Development Process*. Addison-Wesley Professional, 1999.

[JS06] Sami Jantunen and Kari Smolander. Challenges of Knowledge and Collaboration in Roadmapping. In *Proc. International Workshop on Software Product Management (IWSPM'06)*, 2006.

[Jun98] Ho-Won Jung. Optimizing Value and Cost in Requirements Analysis. *IEEE Software*, 15(4):74–78, 1998.

[KAK02] Rick Kazman, Jay Asundi, and Mark Klein. Making Architecture Design Decisions: An Economic Approach. Technical Report CMU/SEI-2002-TR-035, Carnigie Mellon — Software Engineering Institute, 2002.

[Kar96] Joachim Karlsson. Software Requirements Prioritizing. In *Proceedings of* 2^{nd} *International Conference on Requirements Engineering*. IEEE Computer Society Press, 1996.

[Kar06] Lena Karlsson. *Requirements Prioritisation and Retrospective Analysis for Release Planning Process Improvement*. PhD thesis, Lund University, September 2006.

[KKC00] Rick Kazman, Mark Klein, and Paul Clements. ATAM: Method for Architecture Evaluation. Technical Report CMU/SEI-2000-TR-004, Carnigie Mellon — Software Engineering Institute, 2000.

[KOR97] Joachim Karlsson, Stefan Olsson, and Kevin Ryan. Improved Practical Support for Large-scale Requirements Prioritising. *Requirements Engineering*, 2(1), 1997.

[KR97] Joachim Karlsson and Kevin Ryan. A Cost-Value Approach for Prioritizing Requirements. *IEEE Software*, 14(5), 1997.

[Kru00] Philippe Kruchten. *The Rational Unified Process: An Introduction (2^{nd} Edition)*. Addison-Wesley Professional, 2000.

[KWR98] Joachim Karlsson, Claes Wohlin, and Björn Regnell. An evaluation of methods for prioritizing software requirements. *Information and Software Technology*, 39(14–15), 1998.

[Lan02] Rikard Land. Software Deterioration and Maintainability — A Model Proposal. In *Second Conference on Software Engineering Research and Practice in Sweden (SERPS)*, Karlskrona, Sweden, 2002. Blekinge Institute of Technology.

[Leh80] M.M. Lehman. Programs, Life Cycles, and Laws of Software Evolution. *Proceedings of the IEEE*, 68(9), Sept. 1980.

[Lin00] Markus Lindgren. Measurement and Simulation Based Techniques for Real-Time Systems Analysis. Uppsala University, December 2000. IT Licentitate thesis 2000-010, MRTC Report 00/25.

[Lin07] Markus Lindgren. Release Planning in Industry: Interview Data. Technical Report MDH-MRTC-219/2007-1-SE, Mälardalen Real-Time Research Centre, 2007.

[LKK05] Laura Lehtola, Marjo Kauppinen, and Sari Kujala. Linking the Business View to Requirements Engineering: Long-Term Product Planning by Roadmapping. In *Proc.* 13^{th} *IEEE International Conference on Requirements Engineering*, 2005.

[LLNW08a] Markus Lindgren, Rikard Land, Christer Norström, and Anders Wall. Key Aspects of Software Release Planning in Industry. In *Proc.* 19^{th} *Australian Software Engineering Conference*. IEEE Computer Society, 2008.

[LLNW08b] Markus Lindgren, Rikard Land, Christer Norström, and Anders Wall. Towards a Capability Model for the Software Release Planning Process — Based on a Multiple Industrial Case Study. In *Proc.* 9^{th} *International Conference on Product Focused Software Process Improvement*. Springer (LNCS), 2008.

[LLNW09] Markus Lindgren, Rikard Land, Christer Norström, and Anders Wall. An Initial Capability Model for the Software Release Planning Process — Based on a Multiple Industrial Case Study. *Submitted to the Journal of Software Process Improvement and Practice*, 2009.

[LMW05] Lars Lundberg, Michael Mattsson, and Claes Wohlin, editors. *Software quality attributes and trade-offs*. Blekinge Institute of Technology, 2005.

[LNWL08] Markus Lindgren, Christer Norström, Anders Wall, and Rikard Land. Importance of Software Architecture during Release Planning. In *Proc. Working IEEE/IFIP Conference on Software Architecture (WICSA) 2008*. IEEE Computer Society, 2008.

[LWLN08] Markus Lindgren, Anders Wall, Rikard Land, and Christer Norström. A Method for Balancing Short- and Long-Term Investments: Quality vs. Features. In *Proc.* 34^{th} *Euromicro Conference on Software Engineering and Advanced Applications*. IEEE Computer Society, 2008.

[McC76] Thomas J. McCabe. A Complexity Measure. *IEEE Transactions on Software Engineering*, SE-2(4), 1976.

[Mea06] Nancy R. Mead. Requirements Prioritization Introduction. Technical report, Carnigie Mellon University, 2006.

[MT90] Silvano Martello and Paolo Toth. *Knapsack Problems — Algorithms and Computer Implementations*. John Wiley & Sons Ltd., 1990.

[MWN+04] Goran Mustapic, Anders Wall, Christer Norström, Ivica Crnkovic, Kristian Sandström, Joakim Fröberg, and Johan Andersson. Real World Influences on Software Architecture - Interviews with Industrial Experts. In *Proc. 4^{th} Working IEEE/IFIP Conferance on Software Architecture (WICSA)*. IEEE Computer Society, 2004.

[oE04] The Royal Academy of Engineering. The Challenges of Complex IT Projects. Technical report, British Computer Society, 2004.

[Par94] David Lorge Parnas. Software Aging. In *16^{th} International Conference on Software Engineering*. IEEE Computer Society, 1994.

[PSRB07] Anna Perini, Angelo Susi, Filippo Ricca, and Cinzia Bazzanella. An Empirical Study to Compare the Accuracy of AHP and CBRanking Techniques for Requirements Prioritization. In *Proc. 5^{th} International Workshop on Comparative Evaluation in Requirements Engineering*. IEEE Computer Society, 2007.

[PW92] Dewayne E. Perry and Alexander L. Wolf. Foundations for the Study of Software Architecture. *ACM SIGSOFT Software Engineering Notes*, 17(4), 1992.

[RKH03] Björn Regnell, Lena Karlsson, and Martin Höst. An Analytical Model for Requirements Selection Quality Evaluation in Product Software Devlopment. In *Proc. 11^{th} IEEE International Requirements Engineering Conference*. IEEE Computer Society, 2003.

[Rob02] Colin Robson. *Real World Research 2^{nd} edition*. Blackwell Publishers, 2002.

[RS05] Günther Ruhe and Omolade Saliu. Art and Science of Software Release Planning. *IEEE Software*, 22(6):47–53, 2005.

[RSO08] Björn Regnell, Richard Berntsson Svensson, and Thomas Olsson. Supporting Roadmapping of Quality Requirements. *IEEE Software*, 25(2), 2008.

[SA05] Outi Salo and Pekka Abrahamsson. Integrating Agile Software Development and Software Process Improvement: a Longitudinal Case Study. In *Proc. International Symposium on Empirical Software Engineering*, 2005.

[Saa04] Thomas L. Saaty. Decision making — the Analytic Hierarchy and Network Processes (AHP/ANP). *Journal of Systems Science and Systems Engineering*, 13, 2004.

[SC98] M. Strauss and J. M. Corbin. *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory* 2^{nd} *edition.* Sage Publications, 1998.

[SEI05] How Do You Define Software Architecture? Web-site: http://www.sei.cmu.edu/architecture/definitions.html, November 2005.

[Sha02] Mary Shaw. What Makes Good Research in Software Engineering. *International Journal of Software Tools for Technology Transfer*, 4(1), 2002.

[Som00] Ian Sommerville. *Software Engineering* 6^{th} *edition.* Addison Wesley, 2000.

[Spi07] Diomidis Spinellis. Silver Bullets and Other Mysteries. *IEEE Software*, 24, 2007.

[SR05a] Moshood Omolade Saliu and Günther Ruhe. Supporting Software Release Planning Decisions for Evolving Systems. In 29^{th} *Annual NASA Software Engineering Workshop.* IEEE Computer Society, 2005.

[SR05b] Omolade Saliu and Günther Ruhe. Software release planning for evolving systems. *Innovations in Systems and Software Engineering*, 1(2), 2005.

[Sta95] Standish Group. CHAOS. Technical report, The Standish Group, 1995.

[vKR05] Brian R. von Konsky and Michael Robey. A Case Study: GQM and TSP in a Software Engineering Capstone Project. In *Proc.* 18^{th} *Conference on Software Engineering Education and Training*, pages 215–222, April 2005.

[Yin03] Robert K. Yin. *Case Study Research: Design and Methods (Applied Social Research Methods),* 3^{rd} *Edition.* Sage Publications Inc., 2003.