Visualizing, Analyzing and Managing the Scope of Software Releases in Large-Scale Requirements Engineering

Krzysztof Wnuk

Doctoral Thesis, 2012
Department of Computer Science
Faculty of Engineering
Lund University

ISSN 1404-1219
ISBN 978-91-976939-7-4
Dissertation 39, 2012
LU-CS-DISS:2012-2

Department of Computer Science
Faculty of Engineering
Lund University
Box 118
SE-221 00 Lund
Sweden
Typeset using LaTeX
Printed in Sweden by Tryckeriet i E-huset, Lund, 2012

Email: `krzysztof.wnuk@cs.lth.se`

Abstract

Large market-driven software companies face new challenges in requirements engineering and management that emerged due to their recent extensive growth. At the same time, the pressure generated by competitors' and users' expectations demands being more competitive, creative and flexible to more quickly respond to a rapidly changing market situation. In the pursuit of staying competitive in this context, new ideas on how to improve the current requirements engineering practice are requested to help maintaining the engineering efficiency while coping with growing size and complexity of requirements engineering processes and their products.

This thesis focuses on visualizing, analyzing and managing the scope of software releases in large-scale requirements management for developing software products to open markets. In particular, this thesis focuses on the following requirements management activities in the mentioned context, namely: scope management, decision making, obsolete requirements and requirements consolidation. The goals of the research effort in this thesis are to provide effective methods in supporting mentioned requirements management activities in a situation when the size of them and their complexity require large time and skills efforts.

Based on empirical research, where both quantitative and qualitative approaches were utilized, this thesis reports on improved understanding of requirements scoping in very-large projects by investigating factors affecting decision making, causes and effects of overscoping and presents visualization techniques to assist scope management for large-scale software product development contexts. The technical solutions reported in this thesis were empirically evaluated in case studies in a large-scale context and designed in close collaboration with our industry partners. Additionally, the benefits of using linguistic methods for requirements consolidation are investigated in a replicated experimental study based on a relevant industry scenario. Finally, the phenomenon of obsolete software requirements and their impact on industry practice is explored.

Acknowledgements

The work presented in this thesis was funded by the Swedish Governmental Agency for Innovation Systems under the grants UPITER and UPITER2, Efficient Requirements Architecture in Platform-Based Requirements Management for Mobile Terminals.

First and foremost, I am particularly grateful to my supervisor, Professor Björn Regnell, for his invaluable expertise and advice, inspiring and challenging discussions, and endless patience that supported me throughout this work. I would also like to thank my assistant supervisor, Professor Martin Höst, for his enthusiastic guidance and comments on my work.

The research presented in this thesis was conducted in close cooperation between academia and industry. Therefore, I am particularly grateful to Brian Berenbach from Siemens Corporate Research and Dr. Lena Karlsson, Lic. Eng. Thomas Olsson, Dr. Even-André Karlsson from Sony Mobile for their commitment. I am grateful to all participants of the studies presented in this thesis.

Recognition must also be given here to the co–authors of my papers and others who have helped writing and reviewing them. In particular, I thank Professor Tony Gorschek for excellent collaboration and advise, Lic. Eng. Markus Borg for checking the final print, Lic. Eng. Lars Nilsson, Dr. David Callele and Leora Callele for their perfection in language reviews of my articles and this thesis, which significantly improved their legibility.

I would like to thank my colleagues in the Software Engineering Research Group for an inspiring and supporting collaboration atmosphere. I want to thank all other colleagues at the Department of Computer Science for for providing an excellent environment to work in.

Last but not least, I would like to thank my wife Agata, the rock of our family, the light and love of my life, for her unwavering love and support. Also, to my family and friends: thank you for constantly reminding me what is the most important in life.

Krzysztof Wnuk
In the year of grace 2012

Contents

Introduction

Software becomes more and more pervasive and gains more and more importance in our lives. As a result, our dependence on software intensive system in everyday life increases. At the same time, the intangible and abstract nature of software artifacts demands revisiting and often redefining engineering approaches for constructing complex systems, established originally in other than software domains. The constant need to esteem new ways of achieving repeatability and quality control over the software production process gets particularly important for large software systems. In these systems, the adhered diversity and complexity may severely impede the management of software development processes. Similarly, as the size and complexity of software systems continues to increase, they result in increasingly large and complex sets of requirements. Currently, many companies are facing the problem of dealing with enormous complexity of requirements engineering related artifacts, where current requirements engineering technology can provide useful but only partial solutions.

This thesis focuses on visualizing, analyzing and managing the scope of projects in large-scale requirements engineering contexts. The research presented in this thesis aims for improving the understanding of large-scale requirements engineering contexts, in particular the nature of requirements management related activities, as well as providing methods for supporting some of these activities. The results from this thesis concern scoping, managing obsolete requirements and requirements consolidation tasks. The presented visualization techniques are applied to very-large software projects, increasing the understanding and assisting in improving the scoping process at the case company.

This thesis includes a collection of six papers. This introduction provides a background for the papers and relationships between the studies. Section 1 gives the introduction to the context of the thesis. Section 2 defines the focus of this thesis. Section 3 provides related work in the related subareas of requirements engineering. Section 4 describes the methodology used in this thesis. Throughout this introduction, the papers included in this thesis will be refereed to their roman number (see the list below). Other references can be found at the end of this introduction section.

Included papers

The following six papers are included in the thesis:

I **What Happened to Our Features? Visualization and Understanding of Scope Change Dynamics in a Large–Scale Industrial Setting**
Krzysztof Wnuk, Björn Regnell and Lena Karlsson
In Proceedings of the 17th IEEE International Requirements Engineering Conference (RE' 09), Atlanta, Georgia, USA, August 31 - September 04, IEEE, 2009, pp. 89–98

II **Replication of an Experiment on Linguistic Tool Support for Consolidation of Requirements from Multiple Sources**
Krzysztof Wnuk, Björn Regnell and Martin Höst
Empirical Software Engineering Journal (ESEJ), Springer, vol. 17, Jan. 2012, pp. 305–344

III **Obsolete Software Requirements**
Krzysztof Wnuk, Tony Gorschek and Shuiab Zahda
Under revision for Information and Software Technology journal, Elsevier, 2012

IV **Are You Biting Off More Than You Can Chew? A Case Study on Causes and Effects of Overscoping in Large–Scale Software Engineering**
Elizabeth Bjarnason, Krzysztof Wnuk and Björn Regnell
Information and Software Technology (IST) journal, vol. 54, issue 10, October 2012, Elsevier, pp. 1107–1124
This article is an extended version of the related article XVI presented at the 4th International Workshop on Software Product Management (IWSPM' 2010) which received the best paper award

V **What Decision Characteristics Influence Decision Making in Large–Scale Market–Driven Requirements Engineering?**
Krzysztof Wnuk, Jaap Kabbedijk, Sjaak Brinkkemper and Björn Regnell
under revision for the Software Quality Journal, Springer, 2012
This article is an extended version of the related article XIV presented at the First International Workshop on Product Line Requirements Engineering and Quality (PLREQ' 10)

VI **Scope Tracking and Visualization for Very Large-Scale Requirements Engineering**
Krzysztof Wnuk, Tony Gorschek, Björn Regnell and Even-André Karlsson
submitted to the IEEE Transactions on Software Engineering Journal, IEEE, 2012

Contribution Statement

Krzysztof Wnuk is the main author for five included papers. This means responsibility for running the research process, dividing the work between co–authors, and conducting most of the writing. The research in Papers I,II,III, and VI was mainly performed by Krzysztof Wnuk, who designed and conducted most of the work, as well as reported on the studies. Krzysztof Wnuk wrote most of Papers I, II, III, V and VI, with the assistance from co-authors. The execution of the study reported in Paper III was performed by Shuiab Zahda and the execution of the study reported in Paper V was performed by Jaap Kabbedijk.

For Paper IV, Krzysztof Wnuk actively participated in the study design and wrote significant parts of the text. Most of the design was performed together with the co–authors, while most of the execution and analysis was performed by Elizabeth Bjarnason.

Related Publications

The following papers are related but not included in the thesis:

VII **Can We Beat the Complexity of Very Large–Scale Requirements Engineering?**
Björn Regnell, Richard Berntsson Svensson, and Krzysztof Wnuk
In Proceedings of the 14th International Working conference on Requirements Engineering: Foundation for Software Quality (REFSQ'08), Montpellier, France, June 16-17, 2008, LNCS 5025, Springer-Verlag Berlin, Heidelberg, pp. 123-128
This paper presents challenges faced in very large–scale requirements engineering, which is the context of this thesis. This paper is partly included in Section 3.1 of the Introduction.

VIII **Scaling up requirements engineering – exploring the challenges of increasing size and complexity in market-driven software development**
Krzysztof Wnuk, Björn Regnell and Brian Berenbach
In Proceedings of the 17th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'11), Essen, Germany, March 28-30, 2011 , LNCS 6606, Springer-Verlag, Berlin, Heidelberg, pp. 54-59
This paper presents challenges faced when scaling up requirements engineering to very large projects, which is the context of this thesis. This paper is partly included in Section 3.1 of the Introduction.

IX **Visualization of Feature Survival in Platform–Based Embedded Systems Development for Improved Understanding of Scope Dynamics**

Krzysztof Wnuk, Björn Regnell and Lena Karlsson
In Proceedings of the 2008 International Requirements Engineering Visualization workshop (REV'08), Barcelona, Spain, 8 September 2008, IEEE, pp. 41-50

X **Investigating Upstream versus Downstream Decision–Making in Software Product Management**
Krzysztof Wnuk, Richard Berntsson Svensson and Björn Regnell
Third International Workshop on Software Product Management (IWSPM 2009), Atlanta, Georgia, USA, September 01, 2009, IEEE, pp. 23-26

XI **Feature Transition Charts for Visualization of Cross–Project Scope Evolution in Large–Scale Requirements Engineering for Product Lines**
Krzysztof Wnuk, Björn Regnell and Lena Karlsson
In Proceedings of the Fourth International Workshop on Requirements Engineering Visualization (REV'09), Atlanta, Georgia, USA, August 30, 2009, IEEE, pp. 11-20

XII **An Industrial Case Study on Large–Scale Variability Management for Product Configuration in the Mobile Handset Domain**
Krzysztof Wnuk, Björn Regnell, Jonas Andersson, and Samuel Nygren
In Proceedings of the Third International Workshop on Variability Modelling of Software-Intensive Systems (VAMOS'09), Seville, Spain, January 28-30, Research Report 29 Universität Duisburg-Essen, 2009, pp.155-164

XIII **Architecting and Coordinating Thousands of Requirements – An Industrial Case Study**
Krzysztof Wnuk, Björn Regnell and Claes Schrewelius
In Proceedings of the 15th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ '09), Amsterdam, The Netherlands, June 8-9 2009, LNCS 5512, Springer-Verlag, Berlin, Heidelberg, pp. 118-123

XIV **What decision characteristics influence decision making in market–driven large–scale software product line development?**
Jaap Kabbedijk, Krzysztof Wnuk, Björn Regnell and Sjaak Brinkkemper
In Proceedings of the International Workshop on Product Line Requirements Engineering and Quality workshop (PLREQ'10), Essen, Germany, July 3, 2010 Hildesheimer Informatik-Berichte, Volume: 2010, pp. 42-53

XV **Guiding Requirements Scoping Using ROI: Towards Agility, Openness and Waste Reduction**
Krzysztof Wnuk, David Callele and Björn Regnell

In Proceedings of the 18th IEEE International Requirements Engineering Conference (RE'10), Sydney, Australia, 27 September - 1 October 2010, IEEE, pp. 409-410

XVI **Overscoping: Reasons and consequences: A case study on decision making in software product management**
Elizabeth Bjarnason, Krzysztof Wnuk and Björn Regnell
In Proceedings of the Fourth International Workshop on Software Product Management (IWSPM 2010), Sydney, Australia, September 26, 2010, IEEE, pp. 30-39

XVII **More than requirements: Applying requirements engineering techniques to the challenge of setting corporate intellectual policy, an experience report**
David Callele and Krzysztof Wnuk
In Proceedings of the Fourth International Workshop on Requirements Engineering and Law (RELAW 2011), Trento, Italy, August 30, 2011, IEEE, pp. 35-42

XVIII **A case study on benefits and side–effects of agile practices in large-scale requirements engineering**
Elizabeth Bjarnason, Krzysztof Wnuk and Björn Regnell
In proceedings of the 1st Workshop on Agile Requirements Engineering (AREW 2011), Lancaster, UK, July 26 2011, ACM, New York, NY, USA, pp. 1-5

XIX **An empirical study on the importance of quality requirements in industry**
Jose Luiz de la Vara, Krzysztof Wnuk, Richard Berntsson Svensson, Juan Sanchez and Björn Regnell
In the Proceedings of the 23rd International Conference Software Engineering and Knowledge Engineering (SEKE 2011), Miami Beach, USA, 7 - 9 July, 2011, Knowledge Systems Institute Graduate School, pp. 438-443

XX **Requirements are slipping through the gaps: A case study on causes and effects of communication gaps in large-scale software development**
Elizabeth Bjarnason, Krzysztof Wnuk and Björn Regnell
In the Proceedings of the 19th IEEE International Requirements Engineering Conference (RE'11), Trento, Italy, August 29 - September 02, 2011, IEEE, pp. 37-46

XXI **Requirements scoping visualization for project management**
Krzysztof Wnuk and David Callele

In Proceedings of the 2nd International Conference on Software Business (ICSOB 2011), Brussels, Belgium, June 8-10, 2011, Lecture Notes in Business Information Processing, vol. 80, Springer-Verlag Berlin Heidelberg, pp. 168-180

XXII **Challenges in supporting the creation of data minable regulatory codes: a literature review**
Krzysztof Wnuk and Brian Berenbach
In the Proceedings of the 21st Annual International Symposium of the International Council on Systems Engineering (INCOSE 2011), INCOSE San Diego, CA, June 20 - June 23, 2011, INCOSE-International Council on Systems Engineering, pp. 1097-1114

XXIII **How can Open Source Software Development help Requirements Management gain the potential of Open Innovation: An Exploratory Study**
Krzysztof Wnuk, Dietmar Pfahl, David Callele and Even-André Karlsson
In print for the ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), 19-20 September, Lund, Sweden, 2012

XXIV **Controlling Lost Opportunity Costs in Agile Development – the Basic Lost Opportunity Estimation Model for Requirements Scoping**
Krzysztof Wnuk, David Callele, Even-André Karlsson and Björn Regnell
In print for the Third International Conference on Software Business, ICSOB 2012, MIT Sloan School of Management, 18-20 June, 2012

XXV **Industrial comparability of student artifacts in traceability recovery research – An exploratory survey**
Markus Borg, Krzysztof Wnuk and Dietmar Pfahl
In the Proceedings of the 16th European Conference on Software Maintenance and Reengineering, Szeged, Hungary, March 27, 2012, IEEE, pp. 181-189

XXVI **Towards scalable information modeling of requirements architectures**
Krzysztof Wnuk, Markus Borg and Saïd Assar
In print for the First International Workshop on Modeling for Data-Intensive Computing, October 15-18, 2012, Florence, Italy

XVII **The Effect of Stakeholder Inertia on Product Management**
Krzysztof Wnuk, Richard Berntsson Svensson and David Callele
In print for the Second International Workshop on Requirements Engineering for Systems and Systems-of-Systems (RESS'12), 25 September 2012, Chicago, IL, USA

1 Setting the Context

1.1 Software and requirements engineering

In 1960s, when the term Software Engineering (SE) was introduced, software was just a small, marginal part of an expensive computing machine. Changes over the last forty years have been remarkable, resulting in a situation where the software, not the hardware, is the main cost of a computing machine. Over the last decades, software engineering gained importance and is currently an engineering discipline that influences all aspects of software production, from the initial idea recognition and its specification to the post development software maintenance activities. The fact that software engineering is an engineering discipline implies that its products are things that work. As a result, software products are produced by applying theories and tools where these are appropriate (Sommerville, 2007). The main focus of software engineering is the practical problems of producing software. To build software, software engineers need to use their knowledge of computers and computing to solve problems. Therefore, the essential part in the definition of software engineering is understanding the nature of the problem in order to be able to apply the right "computing machinery" to solve it (Pfleeger, 2001). Software engineering is also considered as a part of system engineering discipline (Sommerville, 2007).

Today's software solutions are often very large and complex. The size and complexity explosion was partly the reason for establishing the software engineering field at the NATO Software Engineering Conference in 1968 (Naur and Randell, 1968). Since then, software engineering continued as a profession and field of study dedicated to creating better quality software that is more affordable, easier to maintain and quicker to build. Thanks to that effort, it is possible today to produce software systems with millions lines of code that can be robust, effective and secure. In a similar manner, it can be stated that the Requirements Engineering (RE) field was established partly due to extensively growing size of requirements specifications which created needs to provide engineering means to activities related with discovering system functionalities and constraints (Jacobs et al, 1994).

Producing large and complex software systems requires finding software engineering methods and techniques that can demonstrate coping with the scale and complexity of these systems. The *scalability* can be defined as a possibility of using a certain method or technique on a much bigger set of artifacts without an exponential, or other very significant, increase of the cost of using this technique. The cost is defined here both in terms of required effort and skills to tackle a given problem. Unsurprisingly, computer science and programming scientific articles seem to report scalable methods or paradigms more often than software engineering research literature. The main difficulty of successfully researching scalable software engineering mechanisms lies in the human-intense nature of software engineering tasks that limit automatic analysis and transformation possibilities.

Producing a software system includes a set of activities and associated results which are called the *software process*. The high-level activities of software specification, development, validation and evolution are parts of software processes. Software process models are ways of abstracting, defining and connecting these activities. Figure 1 depicts the first published software development model, called the waterfall model (Royce, 1970). This model is still used in 40% of the companies, according to the survey from 2003 (Neill and Laplante, 2003). The simplicity of this model is unquestionably one of its strong points (Pfleeger, 2001). The five principal stages of the model contains the following activities (Sommerville, 2007; Royce, 1970):

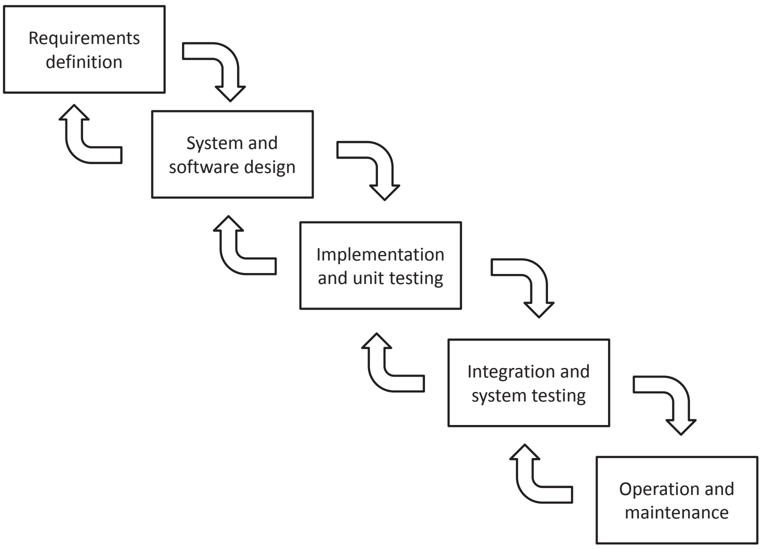

Requirements definition
System and software design
Implementation and unit testing
Integration and system testing
Operation and maintenance

Figure 1: The software life cycle waterfall model (Sommerville, 2007; Royce, 1970)

- *Requirements analysis and definition* - in this phase, high level goals, constrains and functionality are discussed are agreed on with system users. The resulting agreement on the content of the software system is often documented in a document called a system specification

- *System and software design* - the initial set of high-level requirements is mapped onto an overall system architecture comprising both hardware and software elements. At this stage, the fundamental software system abstractions and their relationships are also defined
- *Implementation and unit testing* - in this phase, software developers realize the software design into working software units. The accordance of the functionality of each working unit with the requirements is verified in unit testing
- *Integration and system testing* - the previously produced software units are integrated, and the integration correctness is tested as a complete system to ensure that the software requirements have been met
- *Operation and maintenance* - in the final phase, errors that were not discovered in earlier stages of the life cycle are addressed, improving the implementation of system units and enhancing the system's services when requirements are discovered

The waterfall received many critics since being introduced. McCraken and Jackson (1981) pointed out that the model imposes a project management structure on system development. Furthermore, Curtis et al (1987) noted that the waterfall model's major shortcoming lies in its inability to treat software as a problem-solving process and to consider software development process as a manufacturing process rather than a creation process. As a result, other software development process models were proposed. One of the proposed models is the spiral model proposed by Boehm (1988), where the software development process is represented as a spiral. A phase in this model is represented as a loop in a spiral with four sectors: (1) objective setting, (2) risk assessment and reduction, (3) development and validation and (4) planning. The proposed model explicitly recognizes and analyzes risks stressing the importance of continuous risk management. Another model, called the prototyping model, allows all or part of the system to be constructed quickly to understand or clarify issues, it has the same objective as an engineering prototype. Blazer's transformational model (1981) tries to reduce the opportunity for error by eliminating several development steps. Finally, new software development models that have recently emerged, such as Agile (Cunningham, 2012) and Software Product Lines (Pohl et al, 2005b) paradigms.

According to the Agile manifesto, (1) individuals and interactions should be put over processes and tools, and (2) the contract negotiation should be sustained by a close customer collaboration (Cunningham, 2012). Moreover, the Agile manifesto favors responding to change rather than following a plan, and the value of delivering early prototypes to the customer rather than a comprehensive documentation. On the other hand, the Software Product Lines (SPL) paradigm provides a strategic reuse of assets within an organization. SPLs help to cope with complexity of today's software-intensive systems by using platform and mass customization during their development (Pohl et al, 2005b).

The requirements analysis and definition phase has changed from being initially recognized as a simple planning phase where the requirements are written down to a separated research field within software engineering. Many definitions of requirements engineering were proposed since the field was established. The classical definition of requirements engineering given by Sommerville (2007) defines it as "the process of understanding and defining what services are required from the system and identifying the constraints of the system's operation and development". The use of the term "engineering" implies that the techniques used to ensure that system requirements are complete, consistent and relevant should be applied in a systematic way, and that the whole process should be repeatable. Kotonya (1998) compares requirements engineering to "system analysis" which is mainly concerned about analyzing and specifying business systems. However, system analysis is mainly focusing on business aspects, while requirements engineering is often concerned with both business and system concerns of a system to be developed. The importance of requirements engineering is stressed by Aurum and Wohlin (2005) as one of the most crucial stages in software design and development when the critical problem of designing the right software for the customer is tackled. Aurum and Wohlin extend the definition of requirements engineering with the identification of goals for a proposed system, the operation and conversion of these goals into services and constraints, as well as the assignment of responsibilities for the resulting requirements to agents as humans, devices and software (Aurum and Wohlin, 2005; Sommerville, 2007). The initially proposed ways of grasping the requirements engineering discipline need further extensions, for example in the Market-Driven Requirements Engineering (MDRE) context described later in this chapter.

There are many different views on what to include into the requirements engineering process (Kotonya and Sommerville, 1998; Sommerville, 2007; Neill and Laplante, 2003; Berenbach et al, 2009). The view depicted in Figure 2, illustrates the requirements engineering process in four high-level sub-processes, namely: (1) feasibility study, (2) requirements elicitation and analysis, (3) requirements specification and (4) requirements validation (Sommerville, 2007). Another view on the requirements engineering process, provided by Kotonya (1998), consists a set of three structured activities, namely: (1) requirements elicitation, (2) requirements analysis and negotiation and (3) requirements validation. Their overall goal is to produce, validate and maintain a system requirements document.

The four main sub-processes of the requirements engineering process model provided by Sommerville (2007) are complemented by the "feasibility study" sub-process concerned with assessing whether the system is useful to the business. Figure 2 illustrates also the relationships between these activities and documents produced at each stage in the requirements engineering process. The model presented in Figure 2 is one among many defined in the requirements engineering discipline. Another example is the spiral model that accommodates approaches to a development in which the requirements are developed to different levels of detail. The number of iterations around the spiral can vary, so the spiral can be exited after some or all of the user requirements have been elicited (Boehm, 1988).

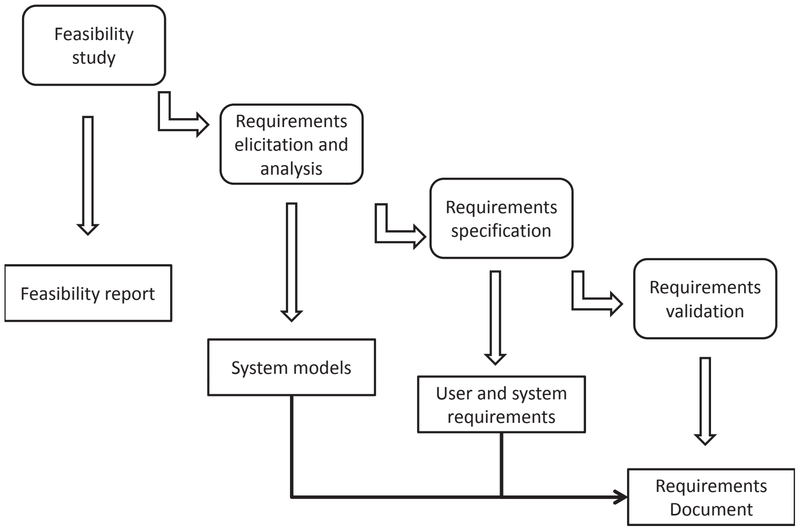


Figure 2: A requirements engineering process example (Sommerville, 2007).

1.2 Requirements management in a large-scale market driven context

Changes to existing requirements and new requirements arriving to the project are inevitable situations at all stages of the system development process (Kotonya and Sommerville, 1998). Furthermore, a common case is that a significant part of an initial system's requirements will be modified before it is put into service (1998). This fact may often cause serious problems for the development of the system. To cope with changes to requirements, Requirements Management (RM) activities are necessary to document and control these changes. Requirements management can also be considered as a process of managing large amount of information and ensuring that it is delivered to the right people at the right time.

The principal requirements management activities, according to Kotonya and Sommerville (1998), are: (1) change control and (2) change impact assessment. The change impact assessment comprises warrants that proposed changes have a known impact on the requirements and software system. The change control ensures that, if a change is accepted, its impact on design and implementation artifacts will be addressed (Kotonya and Sommerville, 1998). Also, managing requirements relationships and managing dependencies between the requirements can be considered as a part of RM. Therefore, RM activities are performed in parallel to the requirements engineering activities and they play a supportive role for them. According to Chrissis (2004), the purpose of requirements management is to manage all post-elicitation results in a project or product, and to identify inconsistencies between those requirements and a project's plans or outcomes. Berenbach et al (2009) go a step further and outline requirements management activities that take place in most, if not all, projects. They list tasks such as identifying volatile requirements, establishing policies for requirements processes, prioritizing requirements, establishing and updating the requirements baselines, documenting decisions and allocating requirements to releases (Berenbach et al, 2009). Other concerns of requirements management involve management of relationships between requirements, and management of dependencies between requirements documents and other documents produced during the software engineering process. This thesis focuses on requirements management for Market-Driven Requirements Engineering (MDRE) contexts.

Software can in general be released in two modes. The first one is called a customer specific mode (also called *bespoke* or contract-driven) when a software product is built to fulfill the contract agreement. The other one is called a *market-driven* mode (or packaged software or commercial off-the shelf) when a software product is addressed to a certain market or group of users. While the main objective in the bespoke mode is often to fulfill a contract and to comply with a requirements specification, the market-driven mode focus mainly to deliver the right product at the right time to the targeted market (Regnell and Brinkkemper, 2005). Moreover, in the bespoke requirements engineering, the success of a software product can be measured by its compliance to a previously agreed requirements specification. In Market-Driven Requirements Engineering (MDRE), the situation is however much more complex. Here, the success of the software product is mainly dependent on the market response which can not be fully assumed 'a-priori'. Therefore, the release time is also important (Wohlin et al, 1995; Sawyer, 2000; Chen et al, 2005), or for some cases even more important than the functionality that the newly released product is providing. The previous fact puts hard time constraints on the requirements engineering and management activities, demanding them to be more flexible, scalable and less time consuming (McPhee and Eberlein, 2002). For example, while setting the scope in a bespoke software project involves time-consuming negotiations and conflict mitigations, in MDRE the scope of the project has to be set using prioritization techniques based on market predictions and effort estimates (Karlsson, 1998; Sawyer, 2000; Carlshamre, 2002b). There is no consensus made between the customer and the contractor of the system, which means that the responsibility for the selection process is only on the contractor who must venture its implications.

A company that is operating in a MDRE context should continuously monitor the market situation by checking competitors' latest achievements, researching market needs and collecting all possible feedback from the market in a chase for achieving or maintaining the cutting edge position within its operational business. This chase after an optimal market window, together with other reasons, creates a constant flow of new requirements and ideas throughout the entire software product lifetime (Karlsson et al, 2002). As a result, the volume of the requirements database continues to grow, putting requirements management techniques and documentation systems to test. Therefore, the requirements process for MDRE needs to be enriched with procedures to capture and analyze this constant flow of requirements (Higgins et al, 2003). Software products in MDRE are continuously evolving and are delivered in multiple releases. The release planning has to focus on time-to-market and return of investment factors. On the contrary, bespoke requirements engineering focuses on one major release which follows the maintenance period. Finally, the results of the effort put during the project can be seen much quicker in the bespoke requirements engineering case where validation is made continuously though the contract. In MDRE, the market is primarily verifying the final products (Regnell and Brinkkemper, 2005).

The complexity and size of software intense systems continues to grow, which in turn gives increasingly large and complex sets of requirements. At the same time, requirements engineering research literature provides industrial examples (Natt och Dag et al, 2004, 2005; Regnell et al, 2006) where current RE technology have a useful but partial effect. Managing large numbers of requirements is an inevitable part of managing requirements in MDRE. The flow of new requirements is almost always delivering more requirements for software products than the actual development resources can implement during each project cycle. As a result, the size and complexity of the requirements databases grow even faster than the size and complexity of actual software products. In this thesis, this situation is named Large-Scale Requirements Engineering (LSRE) or even Very Large-Scale Requirements Engineering (VLSRE). These contexts are characterized in Paper VII, while the definitions and descriptions also are repeated in Section 3.1. The size of the requirements databases in a VLSRE case may exceed tens of thousands of requirements, which puts new expectations on requirements management methods and tool support. Furthermore, as development projects grow in complexity and new products with many features are released to the market, the importance of good practices in requirements management grows (Berenbach et al, 2009). Improving the scalability of requirements engineering and management tools, processes and methods is crucial for succeeding in VLSRE contexts. Most of the research in this thesis, apart from the experiment study reported in Paper II, has been conducted in a VLSRE context.

Software Product Lines (SPL) have gained more prominence amongst large software companies. SPLs provide a strategic reuse of assets withing an organization (Pohl et al, 2005b). Reuse of common features between product is called variability management. Variability is defined as the possibility of a software artifact having more than one behavior in its lifecycle (Svahnberg, 2003). Variability management is the result of software products changing from originally rather static systems to highly extensible and dynamically changing contemporary software systems (van Gurp et al, 2001).

An SPL concept called *scoping* is the process of selecting requirements to implement in the forthcoming project (Wohlin and Aurum, 2005; Greer and Ruhe, 2004). Scoping is a key activity for achieving economic benefits in product line development (Schmid, 2002). It is important to mention that SPLs apply to more than just the source code and extends to other artifacts used to construct software products. New products are created by reusing as much software artifacts as possible. Some of the benefits of using the SPL approach are: (1) decreased time-to-market, (2) improved control over unpredicted growth, (3) improved quality of the product (Linden et al, 2007) or (4) achieved reuse goals (Clements and Northrop, 2002). When using SPLs, a system's complexity and cost have an invese relationship. SPLs increase the possibility of flexibility within the organization, since the knowledge in the organization is more widely deployed (Clements and Northrop, 2002).

2 Research Focus

Empirical studies were conducted to shape the research goals for this thesis. Paper VII reported on the exploration of and the experiences in a very large-scale requirements engineering context. Three challenging research areas in very large-scale requirements engineering are identified in this paper.

- **Challenge 1: Sustainable requirements architectures: fighting information overload.** The term requirements architecture is here understood as the underlying structure by which the requirements are organized. This includes the data model of the requirements with their pre-conceived attributes and relations. In very large-scale requirements engineering (see Section 3.1 for the precise defonition of the VLSRE context), the amount of information that must be managed is immense and impossible to grasp by a single person. Therefore, the need emerged for scalable requirements architectures that are sustainable and allow for controlled growth. Scalable requirements architectures help the requirements engineers in a large organization to track the myriad of issues that continuously emerge. Paper III deals with this research challenge. Paper XXVI presents a modeling framework for requirements artifacts dedicated to a large-scale market-driven requirements engineering context.

- **Challenge 2: Effective requirements abstractions: fighting combinatorial explosions.** Managing interdependencies, prioritization, resource estimation, and change impact analysis are critical in VLSRE. The inevitable combinatorial explosions significantly increase the complexity of the mentioned tasks. A major vehicle for fighting these combinatorial explosions may be the use of abstraction mechanisms and experience-based heuristics. Papers I, II, VI and XXVI deal with this research challenge. Paper XXII explores challenges in supporting the creation of data minable regulatory codes in very large-scale requirements projects.

- **Challenge 3: Emergent quality predictions: fighting over-scoping.** Given a competitive market and a large and demanding set of stakeholders, there appears to be an inevitable shortage of resources to meet quality expectations. Predicting the system level quality that emerges from a myriad of details is challenging, sometimes resulting in a sustained risk of defining a too large scope for platform development. Papers I, IV, V, VI, IX, X, XI, XII, XIV, XV, XVI, XXII, XXIII, XXIV, XXV and XXVII cover aspects related to this research area.

A two-stage empirical investigation was conducted into large-scale requirements engineering contexts, and was reported in Papers VIII and XIII. First, seven practitioners from Sony Ericsson were questioned regarding the tasks in very large-scale requirements engineering and the quality attributes of requirements architecture. The results are reported in Paper XIII.

Next, six participants from two additional companies were interviewed in order to obtain an improved understanding of the challenges in scaling up requirements engineering processes. We focused on understanding which challenges are exacerbated in large or very large-scale contexts. These challenges helped in defining the research goals and provided input to the papers reported in this thesis. We identified five additional challenges.

- **Challenge 4: Scoping.** This includes the challenge of getting to see the "big picture" of numerous scoping decisions. This challenge calls for further research into visualizing the scope of a project. Scope visualizations could help minimize effort wasted on features that are later removed from the scope of a project. The scoping challenge also calls for further research into developing lightweight methods for scope management that can scale up to VLSRE. Papers I, IV, V, VI, IX, X, XI, XII, XIV, XV, XVI, XXII, XXIII, XXIV, XXV and XXVII cover aspects related to this challenge. This challenge is related to **challenge 3**.

- **Challenge 5: Organization and responsibility.** This includes the challenge of managing growing organizations with overlapping roles and untrained newcomers. This requires better understanding of how to organize a growing software organization. Clearly defined roles can minimize the problem of work "falling between the chairs".

- **Challenge 6: Communication gaps.** This includes the differences in understanding the requirements by different roles and direct communication with customers. This challenge calls for further research into supporting the multiple-viewpoint understanding and the interpretation of complex sets of requirements. It also calls for discovering methods which improve conveying the voice of the customers by customer representatives. Paper XX investigates this challenge.

- **Challenge 7: Process misalignments.** These occur when interacting sub-processes are not coordinated and synchronized. This challenge calls for further research into improved synchronization of processes, faster communication of process changes, and the development of methods for assessing the scalability of requirements management processes. Paper XVIII investigates this challenge.

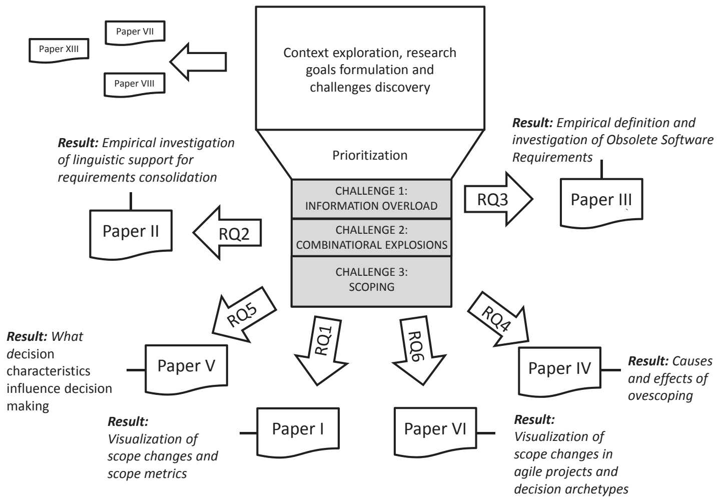
- **Challenge 8: The Structure of RE artifacts.** This includes how to group, reuse, understand and control changes in large and complex sets of requirements. This challenge calls for investigations into requirements artifact structures that are effective and efficient in the large. Papers VIII and XXVI investigate this challenge. This is also related to **challenge 1**.

The research focus of this thesis centers on an amalgamation of the identified research challenges 1, 2, 3 and 4: providing visualization of scoping dynamics for very-large projects (Papers I and VI); speeding up information analysis (Paper II); reducing the information overload (Paper III); understanding the reasons and consequences of setting a too large scope of a project (Paper IV); and factors affecting scoping decisions (Paper V). The remaining related challenges 5, 6, 7 and 8 are not investigated in this thesis[1].

The main research questions investigated in this thesis are as follows:

[1]The investigation of the remaining related challenges 5, 6, 7 and 8 is a part of future work, e.g. see Papers XX and XXVI.

- **RQ1**: How can the scope dynamics in a large-scale software development context be visualized?
- **RQ2**: Does the linguistic method of finding similar requirements outperform the searching and filtering method of the requirements consolidation?
- **RQ3**: What are obsolete software requirements? What is their impact on industry practice? How are they identified and managed?
- **RQ4**: What are the causes and effects of overscoping?
- **RQ5**: What decision characteristics influence decision making in large-scale market-driven requirements engineering?
- **RQ6**: How can the scope change dynamics in a large-scale agile-inspired requirements engineering context be visualized?



Paper XIII
Paper VII
Paper VIII
Context exploration, research goals formulation and challenges discovery
Prioritization
CHALLENGE 1: INFORMATION OVERLOAD
CHALLENGE 2: COMBINATIORAL EXPLOSIONS
CHALLENGE 3: SCOPING
Result: Empirical investigation of linguistic support for requirements consolidation
RQ2
Paper II
RQ3
Paper III
Result: Empirical definition and investigation of Obsolete Software Requirements
RQ5
Result: What decision characteristics influence decision making
Paper V
RQ1
Paper I
Result: Visualization of scope changes and scope metrics
RQ6
Paper VI
Result: Visualization of scope changes in agile projects and decision archetypes
RQ4
Paper IV
Result: Causes and effects of ovescoping

Figure 3: The relationships between challenges, research questions and papers presented in this thesis.

Figure 3 depicts relationships between research questions and research challenges. Paper II addresses research challenge 2. In Paper II (addressing research question RQ2), a replicated experiment designed to further investigate the linguistic tool support for the requirements consolidation task is reported. In this experiment, new requirements are analyzed against those already present in a requirements database, and similarities are discovered and recorded. Forty five subjects were assigned to use either linguistic similarity or searching and filtering for the requirements consolidation

Paper III addresses research challenge 1 and RQ3. This paper empirically investigates how outdated information is managed in requirements repositories. In the absence of mechanisms to clean up outdated information, the amount of information grows continuously. Paper III surveyed 219 respondents from 45 countries, exploring the phenomenon of OSRs by: (1) eliciting a definition of OSRs as seen by practitioners in industry, (2) exploring ways to identify and manage OSRs in requirements documents and databases, (3) investigating the potential impact of OSRs, (4) exploring the effects of project context factors on OSRs, and (5) defining what types of requirements are most likely to become obsolete.

Papers I, IV, V and VI explored the third challenge, scoping. In Paper I, addressing research question RQ1, the Feature Survival Charts scope visualization technique was implemented and evaluated in three projects. The evaluation demonstrated that the charts can effectively focus investigations of reasons behind scoping decisions, and that they can be valuable for future process improvements. A set of scoping measurements is also proposed, analyzed theoretically and evaluated empirically with data from the cases.

Paper IV, addressing research question RQ4, reports from a case study aimed at understanding overscoping in a large-scale, market-driven requirements engineering context. The results provide a detailed picture of overscoping as a phenomenon including a number of causes, root causes and effects. Paper IV also indicates that overscoping is mainly caused by operating in a fast-moving, market-driven context. Weak awareness of overall goals, in combination with low development involvement in early phases, may contribute to ’biting off’ more than a project can ’chew’. Furthermore, overscoping may lead to a number of potentially serious and expensive consequences, including quality issues, delays, and failure to meet customer expectations.

Paper V, addressing research question RQ5, investigates what factors influence the decision lead times and the decision outcomes. A large decision log in a retrospective case study at a large software company was statistically analyzed and the results were validated in a survey among industry participants.

Paper VI, addressing research question RQ6, extends the Feature Survival Charts technique to a very large agile-inspired context. Paper VI also presents decision patterns and archetypes derived from this context.

The list of challenges, albeit derived from several companies in an empirical investigation, should not be considered as the only list of issues in large-scale requirements engineering. Several authors investigated and reported challenges in requirements engineering (Curtis et al, 1988; Hall et al, 2002; Karlsson et al, 2002; Maccari, 1999; Lubars et al, 1993) Among other challenges not investigated in this thesis are, for example: managing requirements interdependencies (Carlshamre et al, 2001), managing quality requirements (Berntsson Svensson, 2009), allocating resources (Trautmann and Philipp, 2009), integrating requirements engineering with the development process (Maccari, 1999), estimating cost (Magazinovic and Pernstål, 2008) or defining scalable requirements management processes (Berenbach et al, 2009).

3 Related Work

The research in this thesis relates to various aspects of requirements engineering and management. The concepts that the research is mostly related to are: large-scale software engineering, market-driven requirements engineering, requirements management, requirements prioritization, release planning and roadmapping, and finally requirements visualization. These aspects, together with examples of scientific contributions, are presented in the sub-chapters that follow.

3.1 Engineering and researching large-scale software systems

One of the interesting characteristics of the requirements engineering is the ability to abstract large parts of the source code and pack them under a concise name of the feature. Depending on the abstraction level, 50 000 lines of the source code solution may be represented as a single market feature, or as a set of 200 system level requirements related with 200 quality aspects. This ability of compression may lead to the situation when requirements engineering research reported in a large-scale context actually operates on a small amount of high-level information, simplifying the problem of scalability. As a result, reported methods do not have to be fully scalable, unless they only operate on this high abstraction level.

While browsing requirements engineering research literature, it is tempting to make the statement that most research reported follows the mentioned abstraction level simplification. A precise definition of the context where the result of an inquiry applies, or have been performed, is undoubtedly a proper, but rare, behavior. When the simplification of the placement of the reported results on the abstraction ladder is made, addressing the scalability of achieved results becomes difficult. As a step towards clarifying the mentioned issue, related Paper VII proposes a classification of the orders of magnitude in requirements engineering is introduced, also repeated here. Table 1, also available in Paper VII, defines four orders of magnitude in RE, based on the size of the set of requirements that are managed by an organization that develops software-intensive systems. The levels are inspired by the characterization of orders of magnitude in the digital circuits integration field.

The number of requirements was chosen as a proxy for complexity, as it is believed in Paper VII that increased numbers of customers, end users, developers, subcontractors, product features, external system interfaces, etc. come along with increased number of requirements generated in the RE process. This in turn increase the complexity of requirements engineering. Furthermore, Paper VII suggests that the complexity of a set of requirements is heavily related to the nature of interdependencies among requirements, see e.g. Carlshamre et al (2001) for an empirical investigation of interdependencies. With a realistic degree of interdependencies among n-tuples of requirements, it can be hypothesized that the number of interdependencies to elicit, document and validate increases significantly with the increased number of requirements. When shifting from MSRE to LSRE, a typical heuristic for dealing with the complexity of interdependency management is to bundle requirements into partitions and thereby creating a higher level of abstraction, where interdependencies among bundles can be managed with reasonable effort. When shifting from LSRE to VLSRE, the conjecture is that even the number of bundles gets too high and the size of bundles becomes too large to allow for interdependency management with desired effectiveness. If the number requirements bundles becomes too large, the interdependency links loose practical usefulness as they relate to coarse grained abstractions.

SSRE and MSRE are a common scale in research papers that seek to validate a proposed method or tool. For example, in Feather et al (2000), a specific tool is validated only with a set of 67 requirements and Hayes et al (2004) used a dataset with 19 high level and 49 low-level requirements. In this situation, it is possible to enumerate and manage complex relations among requirements, even with dense relation patterns. However, it is believed in Paper VII that few industrial situations in current system development can avoid stretching beyond SSRE and even MSRE. Only a few examples in RE literature that discusses LSRE (such as Park et al (1998)) and VLSRE (such as Natt och Dag et al (2004)) can be found whether the author believes that LSRE is common industrial practice, confirmed also by Brinkkemper (2004).

The belief presented in Paper VII saying that a significant number of companies that currently face LSRE will grow into the situation of VLSRE is confirmed by Berenbach et al (2009), who report on large project with thousands of requirements Siemens is working with. Berenbach et al (2009) also mentioned that one of the current misconceptions about requirements engineering is the statement that processes that work for a small number of requirements will scale whether, according to him, requirements engineering processes do not scale well unless crafted carefully.

Another problem mentioned by Berenbach et al (2009) is the requirements explosion during a large project when the processes put in place at the beginning of a project do not take into consideration the number of re-

Table 1: Orders of magnitude in requirements engineering , based on Paper VII.

Abrev.	*Level*	*Order of magnitude*	*Sample empirical evidence*	*Interdependency management conjectures with current RE technology*
SSRE	Small-Scale Requirements Engineering	10 requirements		Managing a complete set of interdependencies requires small effort.
MSRE	Medium-Scale Requirements Engineering	100 requirements	(Feather et al, 2000)	Managing a complete set of interdependencies is feasible but requires large effort.
LSRE	Large-Scale Requirements Engineering	1000 requirements	(Park and Nang, 1998)	Managing a complete set of interdependencies is practically unfeasible, but feasible among small bundles of requirements.
VLSRE	Very Large-Scale Requirements Engineering	10000 requirements	(Regnell et al, 2006)	Managing a complete set of interdependencies among small bundles of requirements is unfeasible in practice.

quirements that may need to be managed. As an example (also repeated in Figure 4), Berenbach et al (2009) gave a project with 50 features to start that may not appear to be a large project. After a not unreasonable explosion of each feature to 100 or more high-level requirements the project can grow up to over 5000 high-level requirements. Adding an additional explosion layer of detail needed to implement the product in both its functional and quality aspects and create test cases can wind up with a total of 50000 requirements and at least the same number of traces. Such a number of requirements to manage and trace in unreasonable for today's large projects.

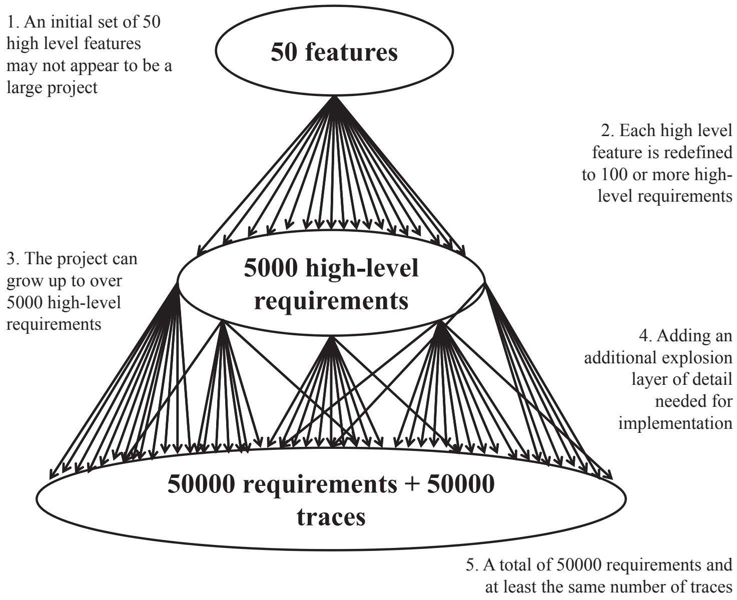

1. An initial set of 50 high level features may not appear to be a large project
50 features
2. Each high level feature is redefined to 100 or more high-level requirements
3. The project can grow up to over 5000 high-level requirements
5000 high-level requirements
4. Adding an additional explosion layer of detail needed for implementation
50000 requirements + 50000 traces
5. A total of 50000 requirements and at least the same number of traces

Figure 4: An example of an explosion of the number of requirements during a project (Berenbach et al, 2009).

To illustrate the complexity in VLSRE, an industrial example outlined in Paper VII is summarized here. This example provides a case description of embedded system engineering in the mobile phones domain, based on experience at Sony Ericsson which has faced a transition from LSRE to VLSRE in the last years, while remaining competitive on the market with a growing number of around 5 000 employees. Mobile phones include a wide range of features related to e.g. communication, business applications and entertainment. The technological content is complex and includes advanced system engineering areas such as radio technology, memory technology, communication protocols, security, audio and video, digital rights management, gaming, positioning etc.

The complexity of requirements engineering is driven by a large and diverse set of stakeholders, both external and internal. Table 2 gives examples of stakeholders that generate requirements. Some stakeholders are counted in billions, such as consumers of different segments, while some are counted in hundreds, such as operators. In the case of Sony Ericsson, the requirements that are generated from internal and external stakeholders amount to several tens of thousands. Figure 5 provides a simplified picture of the different types of requirements and their relations. Similar to the case provided by Feather et al (2000) and Gorschek and Wohlin (2006), requirements originating from external stakeholders, called market requirements, are separated from, but linked to system requirements that are input to platform scoping in a Software Product Line (SPL) setting. Market requirements are mainly generated by operators submitting specifications with thousands of requirements that require statements of compliance. The total number of market requirements as well as platform system requirements at Sony Ericsson each exceeds 10 000. In order to make scoping feasible, platform system requirements are bundled into hundreds of features that represent the smallest units that can be scoped in or out. In order to support product development, the platform capabilities are organized into configuration packages that improve over time, as more and more features are implemented for each new version of a platform. Products are configured through assemblies of configuration packages according to the rules of how they can be combined based on their interdependencies.

The reported publications that focus on large-scale software or requirements engineering can be classified into: (1) empirical reports from large-scale contexts that often provide a number of challenges, (2) papers that literally evaluate or present a technique that is suitable for large contexts and (3) vision papers, often written by senior members of the software engineering community that provocatively bring the scalability issue to the future research agenda. In the sub-chapters that follow, more detailed results from the literature investigation are presented in the mentioned categories.

3.1.1 Technical solutions or methods.

Among the papers that report a technique or solution suitable for large-scale software engineering contexts, Carman et al (1995) present a framework for engineering software reliability that was proposed at a large company called Bellcore. The framework, together with different reliability modeling tools, have been tested in pilot tests. Natt och Dag et al (2004) proposed a method to speed up requirements management that was applied to a context with over 10 000 requirements. Kaushik et al (2011) used information retrieval techniques to find traces between 13380 tests cases and a query of nine random software bugs.

Communication and coordination are considered by Garg (1989) as play-

Table 2: Examples of stakeholders that generate requirements (see Paper VII for more details).

External Stakeholders	Internal Stakeholders
Competitors	Accessories
Consumers of different segments	Customer Services
Content providers	Market research
Legislation authorities	Marketing and customer relations
Operators	Platform development (SW+HW)
Retailers	Product, application and content planning
Service providers	Product development (SW+HW)
Shareholders	Product management
Standardization bodies	Product management
Subcontractors and component providers	Sourcing, supply and manufacturing
	Technology and research development
	Usability engineering

ing a central role in any large-scale cooperative effort. The article presented a design method based on "Intelligent Software Hypertext System" which: (1) can support a flexible and general purpose model for information management (hypertext) and (2) can support various software life cycle models. Linger et al (2007) discussed "function extraction technology" as an emerging software engineering research area. This technology aims to extend the possibilities of automatic computation of software components and their compositions into systems.

Travassos et al (2008) presented and discussed an experimentation environment that could support large-scale experimentation and knowledge management in software engineering. The presented "Software Engineering Environment Framework" provides a set of facilities to allow geographically distributed software engineers and researchers to accomplish and manage experimentation processes and to manage the scientific knowledge during different studies. The methodological viewpoint on large-scale software engineering was also taken by Kitchenham et al (2007) who recommend adopting a more systematic approach to accumulate and report large quantities of empirical evidence. The proposed solution is to use quasi-experimental designs to improve the methodology of large-scale software engineering empirical studies.

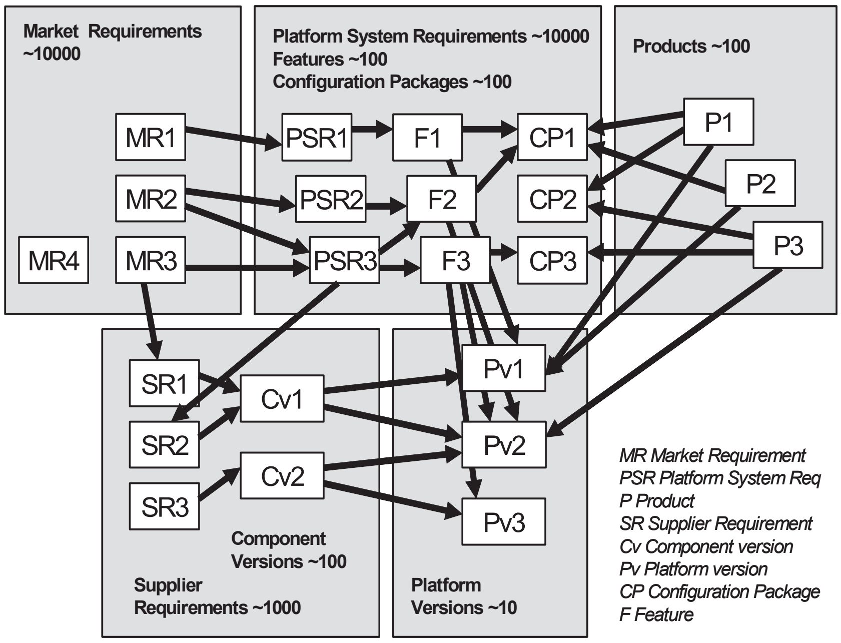


Figure 5: Orders of magnitude in different artifacts of a specific VLSRE case (see Paper VII for more details).

3.1.2 Empirical reports from large-scale contexts.

A significant fraction of empirical reports from large-scale contexts is written by practitioners from large companies. Among them, Conrad and Gall (2008) reported on lessons learned and challenges faced in the development of large-scale systems. The list of challenges comprises:

- Large number of customer requirements
- Formal interface to customer
- Management of customer expectations
- Changing technology
- Traceability
- Scope change and creep
- Resource fluctuation

The challenges were reviewed by requirements engineering experts at Siemens Corporate Research, who agreed that similar challenges could exist across projects with similar characteristics at numerous companies. One of the lessons learned was to establish a traceability model. Duan et al (2009a) used clustering techniques to assist in the prioritization process. The technique were evaluated on an example of 202 requirements. Cleland-Huang et al (2005) explored strategies for incorporating supporting information into a traceability recovery algorithm. The strategies were evaluated on a dataset consisting 180 functional requirements. Both mentioned examples can be classified as MSRE according to Paper VII.

Berenbach et al (2009) stresses the importance of requirements engineering for large projects is stressed. Berenbach et al also presented a list of the common misconceptions about requirements engineering. One of the misconceptions is that processes which work for a small number of requirements will scale. According to Berenbach et al, requirements engineering processes do not scale well unless crafted carefully, and scalability of requirements engineering processes is challenging.

Another view on LSRE is presented by Bergman et al (2002) who investigated the political nature of requirements for large systems and argued that requirements engineering theory and practice must become more engaged with these facets. Requirements for large systems are, according to Bergman et al (2002), constructed through a political decision process. In this process, requirements emerge as mappings between solution space and problem space. These solution spaces are "complex socio-technical ensembles" that often exhibit non-linear behavior in expansion due to domain complexity and political ambiguity.

Ebert (2004) presented techniques for pragmatically dealing with non-functional requirements in large telecommunication systems. Cleland-Huang et al (2008) proposed an elicitation and prioritization process that utilizes data-mining and recommender technologies to facilitate the active involvement of many thousands of stakeholders. The presented solution was claimed as scalable and capable to support elicitation and prioritization of requirements for very large systems.

3.1.3 Challenges and visions

Another significant part of articles about large or very large-scale software and requirements engineering represents the future vision type of articles. These articles are often written in a provocative way by senior researchers within the field to stimulate the community to tackle some of the research challenges and opportunities. Among them, Boehm (2006) identified eight relatively surprise-free trends: (1) the increasing interaction of software engineering and systems engineering, (2) increased emphasis on users and end value, (3) increased emphasis on systems and software dependability (4) increasingly rapid change, (5) increasing global connectivity and need for systems to interoperate, (6) increasingly complex "systems of systems", (7) increasing needs for COTS, reuse and legacy systems and software integration and (8) computational plenty. Boehm (2006) also stated that, traditionally, system and software development processes were recipes for "standalone stovepipe systems" with a high risk of inadequate interoperability with other stovepipe systems. The lack of a common denominator for those stovepipe systems may cause unacceptable delays, uncoordinated and conflicting plans, ineffective or dangerous decisions and inability to cope with rapid changes.

Herbsleb (2007) reflected on an increasing global connectivity and described a desired future for global development together with the problems that stand in the way of achieving that vision. Herbsleb outlined research challenges in four critical areas: (1)software architecture, (2) eliciting and communicating requirements, (3) environments and tools as well as (4) orchestrating global development. He named large-scale software development, when teams are geographically distributed, Global Software Development (GSD), and focused on technical coordination in GSD. The key phenomenon of GSD is the coordination over distance in a sense of managing dependencies between the tasks. The geographical and temporal distance can be considered as an additional, more process oriented, dimension of large-scale systems together with size and complexity.

Damian (2007) considered globalization to be one of the major research challenges in requirements engineering. A global context makes it more difficult to seek out and to integrate the necessary knowledge. Process mismatches, differing technical and domain vocabularies, incompatible environments, and conflicting assumptions can be particularly problematic in a GSD context (Bhat et al, 2006). The challenge of supporting the ongoing negotiation processes prevalent throughout the project life-cycle is also important for multi-site companies (Curtis et al, 1988; Damian and Zowhgi, 2003). Further challenge in GSD is to better understand what media are suitable for the different kinds of communication among all the business stakeholders, analysts and developers. Damian (2007) listed the following challenges in stakeholders' global interaction:

- Knowledge-acquisition and knowledge sharing processes that enable the exploration of stakeholders' needs
- Iterative processes that allow the reshaping of this understanding throughout the entire project
- Effective communication and coordination processes that support the first two types of processes listed

In 2006, the Software Engineering Institute (SEI), together with the U.S. Department of Defense (DoD) established a team of researchers and practitioners in order to investigate very complex systems characterized by thousands of platforms, sensors, decision nodes, weapons and war fighters connected through heterogeneous networks. The systems under consideration were characterized as pushing far beyond the size of current systems by every measure: number of lines of code, number of people employing the system for different purposes, amount of data stored, accessed, manipulated and refined, number of connections and interdependencies among software component and number of hardware elements.

The systems were named as Ultra-Large-Scale (ULS) systems (Northrop et al, 2006). A ULS system could be compared with a biological system, with its community of interdependent and competing organisms and changing environment. The relevant concepts include complexity, decentralized control, hard-to-predict effects of certain kinds of disruptions, difficulty of monitoring and assessment, and the risks in monocultures (Northrop et al, 2006). The ULS systems were characterized by (Northrop et al, 2006):

- **Decentralization.** The scale of ULS systems imposes their decentralization in a variety of ways
- **Inherently conflicting, unknowable, and diverse requirements.** The amount of stakeholders and the diversity of their needs in ULS systems will be immense. This will make impossible to solve all the conflicts between stakeholders needs
- **Continuous evolution and deployment.** ULS systems will continuously evolve, causing constant challenges in integration, interoperability and deployment
- **Heterogeneous, inconsistent, and changing elements.** A ULS system will be constructed from un-matching and evolving parts, causing many misfits and incompatibilities especially in the extension phase

As it won't be impossible to solve all conflicts in ULS systems, mechanisms will have to be proposed, or will automatically emerge, to resolve conflicts locally among those who have an immediate interest in them. ULS systems will also experience "wicked problems" when requirements are neither knowable in advance (because the final agreement about the system's functionality can not be reached due to its changeable nature) nor stable (because each developed solution changes the view on the problem and the problem itself). As a result, no solution is considered to have "solved" the problem (Northrop et al, 2006).

3.2 Requirements prioritization, product management, release planning and roadmapping

Decision processes are the driving forces to organize corporations' success (DeGregorio, 1999). At the same time, important decisions are often subjective (Strigini, 1996; Andriole, 1998). Software managers and decision makers need a better decision support in order to cope with the fast paste of changes in software technology (Aurum and Wohlin, 2003). Researchers have contributed in creating a better support for decision making based on their best knowledge and experience, computational and human intelligence, as well as methods and techniques (Ruhe, 2003). The decision-making process is considered as the dominant activity after the requirements are captured, analyzed and specified on the way towards their implementation. Aurum and Wohlin (2002; 2003) investigated decision making in requirements engineering by using classical decision making models, and they also illustrated illustrate how to integrate these models with requirements engineering process models.

An integral part of market-driven requirements engineering contexts is a constant flow of new requirements arriving from multiple sources (Regnell and Brinkkemper, 2005). Making decisions about which of these incoming requirements implement and which not is a vital part of developing software systems that meet stakeholders' needs and expectations (Karlsson and Ryan, 1997; Regnell et al, 1998). At the same time, it is almost impossible to involve all stakeholders to prioritize requirements, and there are usually more requirements than the company can implement within a given time and resources constraints (Berander, 2004). Thus, it is necessary to select a subset of requirements to implement in the forthcoming project, and hence postpone the implementation of other requirements to a later point in time (Wohlin and Aurum, 2005; Greer and Ruhe, 2004). This selection process is often called *scoping*, and is considered as a key activity for achieving economic benefits in product line development (Schmid, 2002). The requirements selection and release planning process is supported by a requirements prioritization, which can be defined as the activity during which the most important requirements for a system are identified (Sommerville, 2007). The criteria that determine the priority of a requirement include: (1) importance to users and customers, (2) implementation cost, (3) logical implementation order and (4) financial benefit (Lethola and Kauppinen, 2004). As a result, prioritization techniques help to make the outcome of sometimes difficult choices of which requirements to implement less surprising (Karlsson and Ryan, 1997).

There are several prioritization techniques introduced in the literature. An important contribution in this topic has been made by Karlsson et al (1997) who provided a method based on pair-wise comparisons. The method, utilizing the Analytical Hierarchical Process (AHP) (Saaty, 1980), provides a valuable assistance in the prioritization task. Others, such as Beck (2000), introduced a planning game method for prioritization. The planning game method uses ordinal scale for grouping and ranking requirements. The grouping is usually based on cost, value and risk criteria. On the other hand, Karlsson et al (1996; 1997) introduced a numeral assignment technique that uses grouping requirements in for example three or five groups, usually based on customer value. The result is presented on the ordinal scale. Leffingwell and Widrig (2003) proposed a method called the 100$ test or cumulative voting. It has been proposed suitable in distributed environments, and is based on assigning fictional money to requirements and the results are presented on a rational scale. Koziolek (2012) proposed prioritizing quality requirements based on software architecture evaluation feedback. Perini et al (2007) compared the accuracy of AHP and CBRanking prioritization techniques. Duan et al (2009b) suggested automating a significant part of the prioritization process among a large number of stakeholders while Veerappa and Letier (2011) proposed using clustering techniques to identify relevant group of stakeholders that should be considered in the prioritization process. Avesani et al (2005) suggested case-based ranking to address the scalability issues of the AHP prioritization technique. Wiegers (2003) presents a method that combines the customer value, penalty if the requirements is not implemented, implementation cost and risk. Dot voting, binary search tree and the Kano Model are also used to prioritize requirement in agile projects (Bakalova et al, 2011).

Although a significant progress has been made and reported in the prioritization techniques research, there are several issues related to this task. Firstly, there may be conflicts among customers' prioritization lists (Berander, 2004). In this situation, it is important to handle different stakeholders in a structured way. Regnell et al (2001) suggest that the most suitable strategy in the current market segment should be used to adjust each stakeholder's influence in the prioritization process. Secondly, it is often the case that requirements arriving to the company are specified at different levels of abstraction, impeding the requirements prioritization process (Gorschek and Wohlin, 2006). Thirdly, requirements dependencies can also influence the prioritization outcome (Herrmann and Daneva, 2008). Their significant impact on the prioritization process makes it even more complex. One of these dependencies is the inevitable relation between functional and non-functional requirements which are often neglected during the prioritization task.

Fourthly, the number of requirements to prioritize also impedes the prioritization process. In small-scale or even medium-scale requirements engineering, it is feasible to perform the prioritization task on a low level of abstraction where, in large- or very large-scale requirements engineering contexts the prioritization of low level requirements may be very time consuming or even impossible. Several studies proposed different ways of improving the scalability of requirements prioritization techniques: by using machine learning (Duan et al, 2009b), clustering (Veerappa and Letier, 2011), case-based ranking (Avesani et al, 2005). Finally, it is challenging to use business value as a main prioritization criterion (Racheva et al, 2010), proving quantitative method to sizing quality requirements (Herrmann and Daneva, 2008)

Releasing software to an open market is often done in several releases and then software is managed and developed as a product (Albourae and Ruhe, 2006; van de Weerd et al, 2006b). As a result, a product manager role emerged, bringing new types of tasks in MDRE context to cope with the shift from primarily developing customized software to developing software as a product (van de Weerd et al, 2006a,b). The special nature of software creates specific challenges in product management for software solutions, listed by van de Weerd (2006b):

- No cost for manufacturing and distributing extra copies
- The change to software product can be made rather easy by patches and release updates
- The complexity of organizing requirements and tracing changes tasks is high
- Software products are much more frequently released, partly due to their changeable characteristics
- The software product manager's responsibilities regarding the product functionality do not go along with authority over the development team.

The benefits of software products, as it can be seen from the list above, come along with challenges and more complex requirements organization. Two integral parts of software product management are product roadmapping and release planning. Roadmapping includes planning how to use available technological resources and scientific knowledge, and their relationships over a period of time (Vähäniitty et al, 2002). Roadmapping is a form of forecasting a product or product family evolution overtime, including their relationships (R. E. Albright, 2003). Regnell and Brinkkemper (2005) define a roadmap document as a document including product releases plans over a time frame of three to five years. The literature provides many types of roadmap documents, (Schalken et al, 2001) where the one suitable for MDRE contexts release planning is the Product-Technology Roadmap. Roadmapping is a complex task, and it brings challenges in co-operation in different layers of product development, continuous communication (R. E. Albright, 2003), dependencies handling between related products, and coping with rapid technology changes (Carmel, 1999).

Release planning, also called release management is another important activity in software product management. Software release management is the process of making software available to or obtained by its users (van der Hoek et al, 1997). Core functions in this process are requirements prioritization, release planning, constructing and validating a release requirements document and scope management. Various techniques have been proposed or explored in order to support release planning, namely integer linear programming (Abramovici and Sieg, 2002), the analytical hierarchy process (Saaty, 1980), stakeholders' opinions on requirements importance (Ruhe and Saliu, 2005), constraints programming (Regnell and Kuchcinski, 2011) and linear programming techniques using requirements interdependencies (Carlshamre, 2002a). Release planning is challenging due to: uncertainty of the information (Ruhe, 2009), many factors that may affect requirements priorities, requirements dependencies (Carlshamre et al, 2001), changing viewpoints and quality requirements (Jantunen et al, 2011; Berntsson Svensson, 2011).

The criteria identified as important by Wohlin and Aurun (2005) in selecting which requirements to include to the next project comprise: competitors, delivery date, development cost and stakeholder priority of requirement. The last two criteria are similar to the cost-value approach proposed by Karlsson and Ryan (1997) and to the QUPER model (Regnell et al, 2008). Paper I analyzes the reasons for deciding what to remove from the scope of a project.

Researchers have been investigating various aspects of MDRE, starting with Potts, who claimed that "during requirements analysis one party does not always elicit requirements from another, nor does it payback requirements so the other can accept, reject or refine them". Thus, requirements in MDRE context are actually proposed or invented, rather than elicited (Potts, 1995). This fact adds additional dimensions to requirements engineering and requirements management process definitions, and is often responsible for an immense increase of complexity of RE and RM related activities. Regnell et al (1998) presented a specific industrial requirements engineering process for packaged software, which helped the studied company to achieve a measurable improvement in the delivery precision and product quality. The same author has researched the requirements selection task for MDRE (Regnell et al, 2004) and explored bottlenecks in MDRE processes (Höst et al, 2001). Berntsson Svensson focused on roadmapping of quality requirements in MDRE (Berntsson Svensson, 2011). Others, such as Booth (2001) or Karlsson (2002), have focused on reporting challenges in MDRE based on empirical investigations, while Carlshamre et al (2000) focused on comparing two market driven requirements management models and emphasizing the crucial task of managing requirements dependencies.

3.3 Visualization in software and requirements engineering

Customers of software products are often non-technical people. Therefore, visualization in software engineering is a way of a more effective communication with these customers (Avison and Fitzgerald, 1998). For example, diagrams have an advantage of more concisely conveying the information (DeMarco, 1978) than the sentential representation of information (Larkin and Simon, 1987). Diagrams are scalable and can support a large number of perceptual inferences, which can easily be analyzed by humans (Larkin and Simon, 1987). As a result, many visual notations have been proposed since Goldstine and von Neumann's (1948) first flowcharts notation in 1948, and are currently used not only for supporting implementation and testing (Ball and Erick, 1981; Knight and Munro, 2000; Jones et al, 2000), but also other facets of software development (Ogawa et al, 2007; Sellier and Mannion, 2006; Tory and Moller, 2004; Hornecker and Buur, 2006; Vasile et al, 2006; MacDonell, 2005; Biffl et al, 2005; Koschke, 2003; Gotel et al, 2008). The strengths of visual notations are also used to develop software using the Model Driven Development (MDD) paradigm, where software is automatically generated from models (Beydeda et al, 2005).

The visual syntax of notations proposed in software engineering literature is treated with less attention than their semantics (Moody, 2009; Moody et al, 2010). The decisions about semantics (content) seam to be treated with great care. At the same time, visual representation (form) is considered a matter of aesthetics rather than effectiveness (Hitchman, 2002) despite that the positive influence of the visual forms of notations on their understandability has already been confirmed in a number of empirical studies (Purchase et al, 2002; Nordbotten and Crosby, 2001; Masri et al, 2008). Moody (2009) provides a set of design principles in order to achieve a visually efficient notation and establish a scientific foundation for designing visual notations in software engineering. Moody's analysis (2009) focuses on achieving cognitive effectiveness in terms of speed, ease and accuracy with which a representation can be processed by the human mind (Larkin and Simon, 1987).

As an early phase of software development, requirements engineering is communication intensive. Therefore, it requires intensive and efficient communication among multiple stakeholders in order to agree upon the needs for a new software system or its extensions. Effective visualization techniques may significantly improve this communication by envisioning the real value of requirements. Visualization have, according to Gotel (2007), been used to support three aspects of requirements engineering:

- Structure and relationships - visualizing the hierarchical structure of requirements documents, requirements patterns, or more complex graphs. Also, requirements traceability matrices are regularly created to convey linkage between artifacts and support change impact analysis (Duan and Cleland-Huang, 2006; Ozakaya, 2006; Sellier and Mannion, 2006; Osawa and Ohnishi, 2007; Supakkul and Chung, 2010)

- Elicitation support - visual prototypes, story board, mock-ups used to help stakeholders to explore requirements. If these initial drawings are made using a software tool, then the role of this type of visualization is more transient as they can be reused in later refinement activities (Pichler and Humetshofer, 2006; Feather et al, 2006; Zenebe and Norcio, 2007; Sutcliffe et al, 2011)

- Modeling - providing a visualization of requirements specified in a formal language in order to facilitate validation activities. I*, goal modeling, behavioral views, modeling stakeholder concerns and UML frameworks fall into this category and therefore it can be concerned as the dominant focus of research efforts in requirements engineering visualization (Teyseyre, 2002; UML, 2010; Konrad et al, 2006; Evermann, 2008; Sen and Jain, 2007; Isazadeh et al, 1999; Moody et al, 2009; Ugai et al, 2010; Gonzales-Baixauli et al, 2004)

Some researchers proposed using visualizations for supporting requirements engineering decision making. Finkelstein et al (2008) visualized the trade-offs of fairness between multiple customers. Schneider et al (2008) visualized informal communication, Feather et al (2006) visualized requirements risks while Gandhi and Lee (2007) proposed using requirements visualization to support understanding risk assessment during certification. Karlsson and Ryan visualized prioritization results (1997) while Ruhe and Saliu (2005) visualized release plans.

When requirements and their attributes are represented in a textual form, the resulting structure is a table or spreadsheet representation. In this case, the access to multi-placed and spread information can be challenging and overwhelming. Since visual notations offer more dimensions to represent information than text (Tufte, 1990), they are more efficient in representing the mentioned complex information structures and highlight the most relevant interactions and aspects (Dulac et al, 2002). As an example, Gotel et al (2007) proposed taking a set of requirements represented in this traditional textual form, supplemented by the structured UML diagrams, and rendering them in a way that proposes shared comprehension of the full set of a number of requirements-related questions, e.g. revealing unknown patterns. Some situations where this approach may be useful are for example: ensuring that the requirements are grounded in authoritative and representative source or providing a quick glance of the risks, cost and effort needed to implement requirements. In this thesis, visualizations are used to provide an overview of scoping processes in a very-large requirements engineering context.

Recent research in requirements visualization provide a variety of new visualization techniques that aim for "getting to see" requirements in new ways. Among proposed techniques, Lee at al. (2003) proposed an iconic technique that provides an excellent example of using both the shape and the color as additional dimensions to achieve greater cognitive dissonance. Another technique to visually represent requirements using a metaphorical approach is the Volcanic World Visualization technique, proposed by Gotel et al(2007) .

3.4 Natural Language Processing techniques in requirements management

Natural Language Processing (NLP) is a set of techniques for analyzing natural language on various levels and for various tasks and purposes (Liddy, 2003). Information Retrieval (IR), defined as finding unstructured material within larger material based on a given information need (Manning et al, 2008), is overlapping with NLP. IR puts emphasis on the process of finding information while NLP puts emphasis on techniques used in the searching process. In this thesis, we use the term NLP, as suggested by Falessi et al (2010).

Although using NLP techniques for requirements management tasks need to be supervised by practitioners (Ryan, 1993), they provide new possibilities of improving requirements management. These possibilities have been explored in a number of publications. Among those publications that include some kind of empirical evaluations, the majority of the proposed NLP techniques and tools are used to examine the quality of requirements specifications. The quality of requirements specifications is analyzed, for example in terms of the number of ambiguities (Fantechi et al, 2003; Macias and Pulman, 1995). Kamsties et al (2001) proposed detecting ambiguities using inspections while Rupp et al (2000) produced logical forms associated with parsed sentences to detect ambiguities.

Among other quality attributes of requirements artifacts analyzed using NLP techniques, Fabbrini et al (2001) proposed a tool that could improve understandability, consistency, testability and correctness of requirements documents. Edwards et al (1995) presented a tool that uses rule-based parsing to translate requirements from natural languages and thus help requirements analysis and maintenance throughout the system lifecycle. Gervasi et al (2000) presented natural language processing techniques to perform a lightweight validation of natural language requirements which was argued to have low computational and human costs.

Apart from assisting with assessing the quality of requirements, NLP techniques were also used for tasks such as extracting abstractions from text documents (Aguilera and Berry, 1991; Goldin and Berry, 1997), synthesizing crucial requirements from a range of documents that includes standards, interview transcripts, and legal documents (Sawyer et al, 2002), or identifying domain abstractions (Rayson et al, 2000). On the other hand, Sawyer et al (2005) proposed using statistical NLP techniques in supporting early phase requirements engineering while Gervasi (1999) used lexical features of the requirements to cluster them according to specific criteria, thus obtaining several versions of a requirements document. The sectional structure of these documents, and the ordering of requirements in each section, are optimized to facilitate understanding for specific purposes.

Enabling and maintaining requirements dependencies through traceability enables their efficient tracking (Cleland-Huang et al, 2012). By establishing traceability links between requirements and other software project artifacts, lifecycle tasks such as change impact analysis (Cleland-Huang et al, 2003), coverage analysis (Antoniol et al, 2002), linking of rationale and source to requirements (Ramesh and Jarke, 2001), requirements consolidation (Natt och Dag et al, 2006), finding reusable elements (Winkler and Pilgrim, 2010) and finally requirements validation and verification, become more manageable or efficient.

Vector Space Model (VSM) or Latent Semantic Indexing (LSI) information retrieval models (Baeza-Yates and Ribeiro-Neto, 1999) are among the most frequently used models to retrospectively (Asuncion et al, 2010) identify traces. Natt och Dag et al (2004; 2005; 2006) used VSM to measure linguistic similarities to link market requirements to product requirements. Cleland-Huang et al (Lin et al, 2006; Duan et al, 2009b; Cleland-Huang et al, 2010; Laurent et al, 2007) used VSM and the Poirot tool in several experiments while Antoniol (Antoniol et al, 2002) used VSM to recover links between source code and documentation. Hayes et al developed the RETRO tool to link requirements to design documents using vector-space models (Hayes Huffman et al, 2003), as well as LSI (Hayes Huffman et al, 2006). Finally, DeLucia used LSI to recover traceability links in software management systems (De Lucia et al, 2007) and Sultanov et al (2010) applied swarm techniques for requirements tracing.

3.5 Obsolescence in software and requirements engineering, Obsolete Software Requirements (OSRs)

Changes in software business are both rapid and significant and involve not only changing programming fundamentals and languages (Odersky et al, 2008), hardware and technologies Fons et al (2012), pricing and ownership (Jansen et al, 2012), but also business models, software product and services (Cusumano, 2008). For example, a recent shift of focus in software business from product to services (Cusumano, 2008) have ignited serious changes into software pricing models, development and management models, requirements management and software release plans. As a result, software companies need to shift focus and re-prioritize the requirements for their future products focusing on the functional and quality requirements more suitable for service oriented architecture solutions.

Another example of rapidly changing environment is MDRE where requirements are continuously issued by customers and other stakeholders (Regnell and Brinkkemper, 2005). The paste of changes may sometimes be very high resulting in requirements and features becoming quickly obsolete. Frequent and unplanned requirements changes can also cause scope creep, requirements creep and requirements leakage (also referred as uncontrolled requirements creep) (Robertson and Robertson, 1999; Iacovou and Dexter, 2004). Scope creep can then lead to significant scope reductions as overcommitment challenges are addressed. This, in turn, postpones the implementation of the planned functionality and can cause requirements to become obsolete and in the end even project failures (Iacovou and Dexter, 2004).

Software artifact obsolescence has been mentioned in the context of obsolete hardware and electronics in, for example, military, avionics or other industries. Herald et al proposed an obsolescence management framework for system components that is concerned with system design and evolution phases (Herald et al, 2009). Although this framework contains a technology roadmapping component, it does not explicitly mention obsolete requirements. Merola (2006) focused on the software obsolescence problem on the COTS components level for defense systems of systems. He stressed that even though the software obsolescence issue has been recognized as being of equal gravity to the hardware obsolescence, it has not reached the same visibility. Some of the proposed solutions for managing software obsolescence outlined by Merola include: (1) negotiating with the vendor to downgrade the software license, (2) using wrappers and software application programming interfaces, or (3) performing market analysis and surveys of software vendors.

Despite a rather clear relation to managing requirements for software product, the phenomenon of Obsolete Software Requirements seems to be underrepresented in literature. To the best of our knowledge, only a handful of articles and books mention the terms obsolete requirements or/and obsolete features. Among the existing evidence, Loesch and Ploederoeder (2007) claimed that the variability explosion in a software product line context is partially the result of not removing obsolete variable features. Murphy and Rooney (2006) stressed that requirements have 'a shelf life' and suggested that the longer it takes from defining requirements to implementation, the higher the risk of change. This inflexibility is also mentioned by Ruel et al (2010) who stated that change makes requirements obsolete, which in turn can dramatically extend project timelines and increase the total cost of a project. Similarly, Stephen et al (2011) listed obsolete requirements as one of the symptoms of failure of IT project for the UK government. While the report does not define OSRs *per se*, the symptom of failure is ascribed to OSRs and them blocking the timely adoption of the potential of new technologies.

The phenomenon of OSRs has not yet been recognized by neither the IEEE 830 standard (IEEE, 1997) nor CMMI version 1.3 (Software Engineering Institute, 2011). On the other hand, some researchers mentioned obsolete requirements. Savolainen et al (Savolainen et al, 2005) classified product line requirements into four categories: non-reusable, mandatory, variable and obsolete. Moreover they proposed a short definition of obsolete requirements and the process of managing these requirements for software product lines "by marking them obsolete and hence not available for selection into subsequent systems". Mannion et al (Mannion et al, 2000) proposed a category of variable requirements called obsolete and suggest dealing with them in a similar way as described by Savolainen et al (Savolainen et al, 2005).

Obsolescence of requirements can also be defined as their low volatility, thus OSRs and requirements volatility are highly related. SWEBOOK classifies requirements into a number of dimensions and one of the them is volatility and stability. SWEBOK mentions that some volatile requirements may become obsolete (IEEE Computer Society, 2004). Kulk and Verhoef (Kulk and Verhoef, 2008) reported that the maximum requirements volatility rates depend on size and duration of a project. They proposed a model that calculates the "maximum healthy volatility ratios" for projects. On the other hand, Zowghi and Nurmuliani (Zowghi and Nurmuliani, 2002) proposed a taxonomy of requirement changes where one of the reasons for requirements changes is obsolete functionality, defined as "functionality that is no longer required for the current release or has no value for the potential users".

3.6 Scoping in software projects and in requirements engineering

Defining the right scope of a project that fits into the project schedule has been recognized as a known risk in project management (Boehm, 1989). In 1996, Withey explored scoping in software product lines arguing that the economies of scope are the saving obtained when building more product with less input. Thus, the scope is defined by Withey as the common part of a software product line (Withey, 1996). Scoping and scope management were also considered as integral parts of release planning and software product management (van der Hoek et al, 1997). Other early work on software product lines scoping focused on identification of the scope of a software product line (DeBaud and Schmid, 1998, 1999).

Software product lines scoping is considered as a key activity for achieving economic benefits in product line development (Schmid, 2002). Despite that, the existing work in software product line focus mainly on the identification aspect of scoping, e.g. (Kishi et al, 2002; Savolainen et al, 2007; Fritsch and Hahn, 2004; John and Eisenbarth, 2009; Helferich et al, 2005; Kang et al, 2002; Pohl et al, 2005a; Riebisch et al, 2001; Taborda, 2004; Park and Kim, 2005). In a survey of scoping, John and Eisenbarth summarized 16 scoping approaches (John and Eisenbarth, 2009). Seven of the listed methods take features or product features as input.

Fritsch and Hahn (2004) presented a method of checking whether or not a set of products should constitute a product line for a given target market. Helferich et al (2005) focused on creating customer-oriented product portfolios while Kang et al (2002) focused on using marketing and product plan during the product line asset development. Furthermore, Pohl et al (2005a) discussed scoping in relation to product management and product portfolio planning while Park and Kim (2005) proposed a process for domain and economic analysis of core product line assets based on variability analysis. Riebisch et al (2001) proposed a scoping method based on four priority levels and a decision table while Taborda (2004) suggested using release matrices for scoping and release planning.

In MDRE (Regnell and Brinkkemper, 2005), continuously arriving requirements contribute to creating a state of congestion in the requirements handling process when more requirements enter the process than can be handled with the available resources (Karlsson et al, 2002). As a result, the selection process becomes a compromise where (due to the project being overloaded with requirements) the development of already committed features may need to be sacrificed at the expense of wasted effort. Several reasons for removing features form the scope were identified in Paper I.

Related work on scoping in project and product management often focus on a phenomenon of *scope creep*, defined as uncontrolled expansion of initially decided scope (Carter et al, 2001; Kulk and Verhoef, 2008; DeMarco and Lister, 2003; Hall et al, 2002; Iacovou and Dexter, 2004). Scope creep could be caused by: (1) sales stuff agreeing to deliver unrealistically large features without checking the scheduling implications first (Hall et al, 2002) or (2) stakeholders unable to concur on project goals (DeMarco and Lister, 2003). As a result, scope creep can have serious negative consequences on project, including project failures (Iacovou and Dexter, 2004). One of the possible ways to mitigate negative effects of scope creep is combining evolutionary prototyping and risk-mitigation strategies (Carter et al, 2001). Patton (2003) suggested that combining interaction design and extreme programming could help to get the right and yet flexible scope of a project.

Another related phenomenon is *requirements scrap* which is defined as a situation when the scope of a project both increases but also decreases (Kulk and Verhoef, 2008). Whether increase of the project scope is simply scope creep, decrease of the scope could be a result of shrinking budgets or running out of schedule (Kulk and Verhoef, 2008). Paper I visualized a requirements scrap situation and Paper IV investigated the causes and effects of overscoping which include: many changes after the project scope is set, quality issues, wasted effort and customer expectations not always met, see Paper IV for more details. Finally, *requirements churn* phenomenon occurs when changes to requirements do not affect the size of requirements during the project. Kulk et al (2008) presented an example of requirements churn when colors in an interface need to changed and buttons reordered in an interfaces.

4 Research Methodology

The research effort in software engineering is aiming for answering questions regarding developing, maintaining and evaluating software. Conducting research in software engineering requires studying software engineers as they work (Singer et al, 2007). Therefore, the research methods (Glass, 1994; Wohlin et al, 2000) used in this thesis were applied based on their empirical potential. At the same time, the research presented in this thesis could be characterized as conducted using a pragmatic approach (Easterbrook et al, 2007) because a greater interest was put on obtaining practical knowledge as a more valuable source of information.

Various research designs, strategies and data collection methods were used to deal with the research challenges and to develop answers to the research questions outlined in Section 2.

4.1 Research design

According to Robson (2002), there are two main approaches to research: the *fixed* and the *flexible* research design. Wohlin et al (2000) call these approaches research paradigms. The fixed research design, also called the *quantitative* research design (Wohlin et al, 2000), can be characterized by the fact that the design is finished before the data collection starts. Because of that, this approach can also be called a "theory-driven" research design (Robson, 2002). The *fixed* research design is often used to find which one of two or more proposed solutions can exhibit a different behavior. The results of studies conducted using the fixed research design are reported in terms of groups rather than individuals (Robson, 2002). Therefore, the weakness of the *fixed* research design is an inability to capture the individual characteristics of individual human behavior (Robson, 2002).

In contrast, the *flexible* research design, also called the *qualitative* paradigm (Wohlin et al, 2000), evolves during the research process when the data collection and analysis are intertwined. Qualitative data is typically non-numerical, often focused on words, but may also include numbers. Fixed and flexible research designs can further be classified into research strategies, described in the section that follows. The research presented in this thesis uses both types of research designs (see Table 3).

4.2 Research strategies used

The choice of research strategy is an important step in research methodology and it is limited by the prerequisites for the investigation to be performed (Wohlin et al, 2000). Research strategies could also be called research methods (Easterbrook et al, 2007). In this thesis, the name research strategies is used to avoid confusion with data collection methods. Robson provides a list of research strategies in social science (Robson, 2002). In this thesis, the following research strategies have been used:

Case study. This strategy can be categorized as a traditional flexible research strategy. The reason why it is considered as a flexible design is the fact that the details of the design typically "emerge" during data collection and analysis. The case study can be both quantitative and qualitative (Wohlin et al, 2000). Case studies are recognized as appropriate methods to understand complex social phenomena (Yin, 2003). Software engineering is, in general, a complex social phenomenon which allows investigators using the case study approach to preserve its holistic and meaningful characteristics. Therefore, Runeson et al(Runeson and Höst, 2009; Runeson et al, 2012) pointed out that the case study methodology is “well suited for many kinds of software engineering research”. The ability to understand the complexity of the analyzed problem rather than abstracting from it is a principal advantage of performing qualitative case studies (Seaman, 1999). Moreover, Wieringa and Heerkens (2007) classified case study as a well suited for requirements engineering research, even though the results of a case study are more difficult to interpret and generalize than the results of an experiment (Wohlin et al, 2000).

Action research. The important part of this strategy is to influence or change some aspects of whatever is in the focus of the inquiry (Robson, 2002; Davison et al, 2004; Easterbrook et al, 2007). Close collaboration between researchers and those who are the focus of the research is central to action research. Further, action research focuses on "authentic" problems and authentic "knowledge outcomes"(Easterbrook et al, 2007). The result of using this strategy comprises: (1) an improvement of a practice of some kind, (2) the improvement of the understanding of a practice by its practitioners, and (3) the improvement of the situation in which the practice takes place. Easterbrook et al (2007) argues that a large part of the software engineering research is actually using this strategy. It is a common scenario in software engineering research that ideas are originally developed by trying them out on real development projects, and reporting the experiences (Easterbrook et al, 2007). Moreover, Wieringa and Heerkens(2007) put action research on the list of the methods that can be used in requirements engineering research. Action research is closely related to case study(Runeson and Höst, 2009).

Experimental strategy. This strategy can be categorized as a traditional fixed design research strategy (Robson, 2002). In experimental strategy, the researcher introduces a controlled change into the context of the experiment in order to see the result of this change on the object of the experiment. The measured effect of manipulation is then statistically analyzed to confirm the significance of the effect (Wohlin et al, 2000). The details of the design are fully pre-specified before the main data collection begins (there is typically a “pilot” phase before this when the feasibility of the design is checked and changes made if needed). The need to conduct experiments in software engineering was for the first time emphasized in the middle of the 1980’s by Basili and Rombach (1988) and stressed by many others later on (Potts, 1993; Basili, 1996; Fenton et al, 1994; Glass, 1994; Kitchenham et al, 1995). Experiments are mainly quantitative since they focus on measuring different variables, changing them, and measuring them again

(Wohlin et al, 2000). Replicated experiments play an important role in software engineering by allowing us to build knowledge about which results or observations hold under which conditions (Shull et al, 2008).

Survey. This strategy can be used to characterize a broad population of individuals (Easterbrook et al, 2007; Rea and Parker, 2005). Defining clear research questions and a representative sample from a population are important preconditions for conducting survey research (Easterbrook et al, 2007; Robson, 2002). This strategy, also called "‘non-experimental strategy"’, is similar to the experimental strategy but researchers do not attempt to change the studied context or behavior of participants (Robson, 2002). Interviews or questionnaires can be used to collect data (Wohlin et al, 2000) and variables can be measured and controlled. Hypothesis testing is also possible in survey research (Rea and Parker, 2005). Easterbrook et al (2007) listed survey research as one of the research methods in software engineering.

4.3 Data collection methods used

Selecting data collection methods is a necessary and important part of the research methodology (Easterbrook et al, 2007; Lethbridge et al, 2005; Robson, 2002). Since requirements engineering involves real people working in real environments, researching requirements engineering is essential to study practitioners as they solve real problems (Lethbridge et al, 2005). Wrongly selected data collection methods may not reveal all characteristics of the data under analysis and may harm the analysis phase and even the results of the study.

There are many data collection methods (Lethbridge et al, 2005; Robson, 2002). Therefore, selecting appropriate data collection methods requires careful consideration of the research design as well as the pragmatics of the research setting (Easterbrook et al, 2007). Many aspects affect this selection process, where one of the most common is the degree of involvement of software engineers (Lethbridge et al, 2005). During the data collection, it is important to quantify the advantages and disadvantages of the different techniques from the perspectives of the experimenter, the participants, reliability and the generalizability of the results (Easterbrook et al, 2007; Lethbridge et al, 2005). Therefore, multiple techniques can be used to overcome the limitations of single techniques (Easterbrook et al, 2007; Lethbridge et al, 2005), for example while gathering data from multiple perspectives. We have used multiple techniques for data collection in Papers I, V and VI.

The data collection methods used in this thesis are presented with respect to the taxonomy of techniques based on the degree of involvement of software engineers presented by Lethbridge et al (2005) and by the taxonomy presented by Robson (2002). Among the first degree techniques, where software engineers were directly involved in the study, interviews and questionnaires were used. Among the second degree techniques, the instrumenting systems technique was used in this thesis. Among the third degree of involvement techniques, the analysis of electronic databases of work performed was used. Additionally, content analysis technique described by Robson (2002) was also used in this thesis.

Interviews. Interviews are the most straightforward instrument for data collection (Lethbridge et al, 2005). Interviews can be used in studies where the goal is to explore challenges (Karlsson et al, 2002) or gain opinions, about the process or product (Lethbridge et al, 2005). Interviews are flexible and inquisitive (Lethbridge et al, 2005) and recommended for software engineering research (Runeson and Höst, 2009). Despite their time consuming nature, interviews brings the possibility to follow up answers given by participants of the study, interpret the tone of their voice, expressions and intonations. Depending on the resources available, interviews can be used to collect small or large volumes of data (Runeson et al, 2012). According to Robson (2002), interviews can be classified into three types: fully structured, semi-structured and unstructured. In this thesis, semi-structured interviews were used. Semi-structured interviews use a set of predetermined questions but also allow improvisation and exploration (Robson, 2002; Runeson and Höst, 2009). Questions order and wording can be changed. Furthermore, particular questions which seem inappropriate with a particular interviewee can be omitted, or additional ones included. In this thesis, semi-structures interviews were performed in Papers IV and VI.

Questionnaires. Questionnaires are one of the possible data collection techniques for survey research strategies (Easterbrook et al, 2007; Robson, 2002; Singer et al, 2007). Questionnaires allow collection of large amount of data in a cost and time effective way (Lethbridge et al, 2005; Rea and Parker, 2005). Furthermore, web-based questionnaires allow data collection from geographically diverse locations. Moreover, the time to deliver and send back the questionnaire is significantly reduced and the task of filling the questionnaire is simplified (Punter et al, 2003). Questionnaires are suitable for software engineering research (Pfleeger and Kitchenham, 2001; Lethbridge et al, 2005; Singer et al, 2007). On the other hand, poor questions wording and low response rates are the two most common disadvantages of questionnaires. Lethbridge et al reported a 5% response rate from web based software engineering surveys (Lethbridge et al, 2005). The response rate in Paper III was about 8%.

Content analysis. Content analysis technique deals with artifacts produced during software development process. It is classified as an "unobtrusive measure", which means that data collection does not affect collected documents (Robson, 2002). The gathered information, which in this case can be a variety of written information, is analyzed and conclusions based on the content are reported. The indirect involvement of software engineers in the data collection task makes this technique suitable for large volumes of data, which is the case for the studies in Papers I, II, V and VI. It is also a useful technique to be utilized when the goal of the study is to gather or propose a set of metrics (Lethbridge et al, 2005), as in Paper I. The content analysis technique can also be used as a secondary method (Robson, 2002), see Paper VI.

Instrumenting systems. The prerequisite of using this techniques is to have access to software engineers' environment when they are working, but does not require direct contact between the participants and the researchers. This indirect nature of this techniques make is suitable for collecting large volumes of data (Lethbridge et al, 2005). In the instrumenting systems technique, the researcher builds "instrumentation" into the software tools used by software engineers (Singer et al, 2007). In this thesis, the instrumentation technique is used in Papers I, and VI to visualize information recorded in the requirements management tool. The visualization techniques provide process monitoring facilities not available by the requirements management tools. Since people tend to be poor judges of factors such as relative frequency and duration of the various activities they perform, this technique can be used to provide such information accurately (Singer et al, 2007). On the other hand, it is difficult to analyze data from an instrumented system meaningfully, in this case the scope changes history recorded. This disadvantage of this method may require using another data collection method. This technique was also indirectly used in Paper II.

Analysis of electronic databases of work performed. The work performed by developers and software engineers is often stored and managed in various types of electronic databases (Singer et al, 2007). The information and the records how the information was created and managed are a rich source of information for software engineering researchers (Singer et al, 2007; Lethbridge et al, 2005). This data analysis technique is suitable for large amounts of data. The collected data is not influenced by the presence of a researcher. The disadvantage of this technique comprises low or sometimes lack of control over the quality and quantity of the information gathered (Lethbridge et al, 2005). This technique has been used in Paper V where decision logs were studied, in Paper II where the results of requirements consolidation were analyzed and in papers I and VI where decision making history was analyzed and visualized.

4.4 Research classification

Table 3 provides the mapping between the presented papers, research strategies, designs and data collection methods used.

A flexible research design as well as a case study and action research strategies were used in Paper I. Due to a limited number of publications within the requirements engineering field that exclusively addresses issues related to scope management in very large projects, a literature re-

Table 3: Research Classification

Paper	*Research Design*	*Research Strategies*	*Data Collection Methods*
I	flexible	Case study, action research	Analysis of electronic databases of the work performed, content analysis and instrumenting systems
II	fixed	Experiment	Instrumenting systems and analysis of electronic databases of work performed
III	fixed	Survey	Questionnaires
IV	flexible	Case study	Interviews
V	flexible	Case study	Analysis of electronic databases of worked performed, content analysis and questionnaires
VI	flexible	Case study, action research	Instrumenting systems, analysis of electronic databases of the work performed, content analysis and interviews

view was not conducted. The need for investigating scope management in large project emerged from interviews reported in Papers VIII and XIII. We used analysis of electronic databases of the work performed, content analysis and instrumenting systems to collect the empirical data. We analyzed the database where the information about managing features and making decisions about them was stored. A set of scripts was implemented to automatically analyze the available data and provide the FSC visualizations.

A fixed research design was used in Paper II. The goal of Paper II was to experimentally assess whether a natural language processing functionality can provide a better assistance in a task of finding similar requirements than the searching and filtering functionalities. Since two treatments were compared in this study, the experimental research strategy was used. The effect of the manipulation was measured on students. The details of the design were fully pre-specified before the data collection began. Instrumenting systems and analysis of electronic databases of work performed were used to collect the empirical data.

A fixed research design was also used in Paper III. A survey research strategy was utilized to investigate the phenomenon of obsolete software requirements. The goals were to define the phenomenon of obsolete software requirements, investigate how they are handled in industry today and their potential impact. The literature review conducted before executing the survey, revealed that only a handful of papers discuss this topic. The questionnaire data collection method was used in Paper III.

A flexible research design was used in Paper IV. A case study is chosen as a research strategy. The main reason to use this strategy is the nature of the investigated research questions. The phenomenon under investigation, in this case overscoping, was studied in its natural environment (Yin, 2003; Runeson et al, 2012). In-depth analysis of a single case helps to understand the surrounding context of the investigated phenomenon. Interviews were used as a method for data collection.

A flexible research design was also used in Paper V. A case study was chosen as a research strategy and analysis of electronic databases of the work performed was first used to collect the empirical data about factors affecting decision lead-times and outcomes. Next, a questionnaire was used to validate the findings from the first phase of the study.

The aims for Paper VI were accordingly: (1) to develop a method for visualizing the scoping process in agile-inspired continuous development of embedded systems and (2) to identify decision patterns and archetypes based on a very-large record of decisions from the case company. A flexible research design was used in this paper. Case study and action research strategies were used. Researchers were involved in several steps towards implementing the visualization technique, applying the technique on an empirical set of data, and finally using findings from the previous steps to influence and improve the scoping practice at the case company.

The instrumentation tool into the requirements management tool was built. The exported data provided input for the implementation of the FSC+ visualization technique of scoping decisions over time. Next, the analysis of visual representation of the work performed in the scoping process was performed. Several thousands of features were visualized and analyzed. Furthermore, meetings with practitioners and informal interviews were held while developing the visualization technique in order to collect feedback and suggestions about it. Finally, interviews with practitioners were held in order to discuss the results from applying the technique as well as its usefulness. The evaluation of the solution and suggestions for further improvements were collected.

4.5 Validity

Even though selecting proper research strategies and methods is an important step of conducting meaningful research, it does not imply that the results should be trusted without any doubts or questions. Therefore, the results of any research effort should be interpreted in the light of the threats to the validity (Wohlin et al, 2000). A thorough and upstanding criticism of the results is the only way of enabling or rejecting possibilities of generalization or replication. Thanks to threats of validity, researchers can distinguish which results corroborate under which conditions, making them more useful for building up knowledge. These criteria are useful for evaluating various studies, including controlled experiments, most case studies and survey research (Wohlin et al, 2000; Yin, 2003; Easterbrook et al, 2007; Runeson et al, 2012). Several threats to validity classifications were used in this thesis (Robson, 2002; Yin, 2003; Wohlin et al, 2000). The below presented description and discussion of threats to validity is based on the classification suggested by Wohlin et al(2000).

Internal validity concerns the question about the confounding factors that may affect the causal relationship between the treatment and the outcome (Wohlin et al, 2000). Experiments are the type of studies where addressing internal validity is critical (Wohlin et al, 2000). If a researcher incorrectly concludes that the treatment affects the outcome without knowing that a third factor has caused or significantly influenced the outcome, then the study has a low degree of internal validity (Wohlin et al, 2000). Internal validity threats have been given the greatest attention in experimental and quasi-experimental research. In case studies, it should only be a concern for a causal type of studies where an investigator is trying to determine whether there is a casual relationship between events x and y without knowing that some third factor z may actually have caused y (Yin, 2003; Runeson et al, 2012). This logic is not applicable to descriptive or exploratory studies which are not concerned with making causal claims (Yin, 2003). In this thesis, multiple techniques were used to address internal validity threats, including: (1) continuous validation of emerging results and techniques for Papers I and VI, (2) sending transcripts back to interviewees to validate the correctness of derived causal relationships in Paper IV, (3) performing a replication on an experiment to show the range of conditions under which experimental results hold in Paper II or (4) reviewing the survey questions by a native English speaker in Paper III and (5) conducting a survey to compare the results from the content analysis in Paper V.

Conclusion validity arises from the ability to draw correct conclusions about the relation between the treatment and the outcome (Wohlin et al, 2000). Conclusion validity is related to the reliability of the analysis procedures used in a study (Runeson et al, 2012). In qualitative case studies, there is a risk of drawing different conclusions if another person will analyze the interview material. In experiments, conclusion validity could be addressed by using appropriate statistical tests (Wohlin et al, 2000), see Paper II. Conclusion validity could also be called reliability (Yin, 2003; Runeson et al, 2012). A typical criticism of a single case study is the question if a follow-up interview study would produce different results, even if the same research procedures are followed. If the same analysis procedures are used and the same results are obtained, then the study has a high degree of reliability (Yin, 2003). Several methods of addressing conclusion validity were used in this thesis, e.g. recording and transcribing interviews in Papers IV,using observer triangulation during the data analysis and storing all artifacts from the case studies in Papers I, V and VI.

the following techniques were used to address threats to the conclusion validity: performing multiple case studies on the topic of scope dynamics visualization in Papers I and VI, combining two data collection procedures in Paper V, conducting workshop sessions with additional participants in Paper IV, and replication of an experiment in Paper III. Finally, for all studies presented in this thesis, a documentation of the studies was created in enable replications and the questions used were published online.

Construct validity is concerned with the relation between theories behind the research and the observations (Wohlin et al, 2000). Construct validity concerns the quality of the construct (or operational measures (Yin, 2003)) developed when conducting the study or measuring certain effects. The use of multiple sources of evidence and a chain of evidences may increase the construct validity (Yin, 2003) in order to ensure that the result is an effect of the treatment. Case studies have often been criticized for using "‘subjective"’ judgments to collect the data (Yin, 2003). Interview studies need to ensure that the questions stated are interpreted in the same way by the researcher and the interviewed person (Runeson et al, 2012). Multiple sources of evidence were used in Papers I, IV, V and VI.

External validity is related to establishing the domain, task or people to which a study's findings can be generalized. Results obtained in the context of a unique environment, or with a specific group of subjects, may not be fully transferable to other contexts and environments. The ways of minimizing this type of validity treats are using theory in single-case studies or using replication logic in multiple-case studies (Yin, 2003). Moreover, if a case study is focusing on explaining or understanding a phenomenon in its natural setting, then the attempt to generalize from the study is outside its aims (Runeson et al, 2012). We addressed external validity threats by combining a case study and a survey in Paper V, conducting a survey in Paper III and relating the findings from each study to the state of the research published in other papers. However, external validity should be taken into consideration for Papers I, IV and VI.

5 Research Results

The main contributions of this thesis are presented in relation to each research question outlined in Section 2. For each addressed research question, the main threats to validity of the research results are summarized. More detailed contributions and threats to the validity of each paper in this thesis can be found in the respective paper.

5.1 Main contribution of RQ1

The main contributions of Paper, I addressing RQ1, is the Feature Survival Chart technique for showing project scope changes over time. The FSC is applied to three large projects. The results of this empirical evaluation demonstrate that the charts can effectively help in investigating reasons behind scoping decisions. Furthermore, Paper I provides a set of scoping measurements, theoretically analyzed and applied to the empirical data given by the case company.

5.1.1 Main validity issues of RQ1

The main validity issue in Paper I is concerned with external validity. The FSCs have been designed with the requirements management process in mind. However, the author believes that the visualization technique can provide means of describing complex scoping processes also for other software management and requirements management process models, see Paper VI. The second major threat to external validity is concerning the general application of the FSC. The concept has been tested on empirical data from the same company, making the results or evaluation, although positive, falling short on the attempt of generalization. However, since visualization techniques are generally recognized as useful in increasing the understanding of complex or changing datasets, the questions regarding this threat can be limited to details of presented visual techniques rather than their general usefulness. This type of discussion is outlined in Paper VI. The decisions on which projects the visualization techniques should be tested on were made together with practitioners, minimizing the construct validity threat. Finally, the solution was accepted to be implemented as a part of a requirements management measurement and assessment tool, extending its usefulness over the participants involved in the evaluation.

5.2 Main contribution of RQ2

The main contribution of Paper II, addressing RQ2, are the results of an experiment performed to assess if a linguistic method for finding similar requirements can over-perform searching and filtering. The results from the original experiment are confirmed in five out of six tested hypotheses. Moreover, the replication confirms that using NLP techniques help to miss fewer requirements links and make more correct links. The second contribution of this study is the result of the cross-experimental hypotheses testing. Finally, the paper discuss the reasons behind results discrepancies.

5.2.1 Main validity issues of RQ2

The first main threat, related to external validity of this study, is the number of analyzed requirements during the experiment. Since only a relatively small number of requirements is analyzed during the experiment, it is hard to generalize the results on a very large set of requirements, which often is the case in industrial settings. The second main threat is related to conclusion validity, since on the contrary to the original study, where subjects worked independently, in the replicated experiment they were asked to work in pairs. This threat was addressed by performing the analysis of a pre-study questionnaires filled in by all experiment participants. During this analysis, the differences in knowledge of English, industrial experience and experience from the courses have been compared to assess the degree of heterogeneity of pairs.

The third main threat is the difference in user interfaces of compared tools, which may result in a performance difference. This threat has been addressed by giving subjects that used the more complicated tool more time to get familiar with the user interface.

The last main threat is related to the awareness of subjects about their own errors. This may have influenced the number of correct and faulty links. Also, when subjects knew that the time was measured, it is possible that they were more aware of the time spent and therefore effecting the performance results. This threat was addressed by explicitly mentioning that the subjects can not gain anything from performing better or worse with the task, and also by mentioning that the correct answer in not known.

5.3 Main contribution of RQ3

The main contribution of Paper III (addressing research question RQ3) are the results from a survey conducted among 219 respondents from 45 countries exploring the phenomenon of OSRs by: (1) eliciting a definition of OSRs as seen by practitioners in industry, (2) exploring ways to identify and manage OSRs in requirements documents and databases, (3) investigating the potential impact of OSRs, (4) exploring effects of project context factors on OSRs, and (5) defining what types of requirements are most likely to become obsolete.

The results clearly indicate that OSRs are a significant challenge for companies developing software systems – OSRs were considered serious by 84.3% of the respondents. Moreover, a clear majority of the respondents indicated no use of methods or tools to support identification and handling of OSRs, and only 10% of survey respondents reported having automated support. This indicates that there is a need for developing automated methods or tools to support practitioners in the identification and management of OSRs. Further, our study revealed that manually managing OSRs is currently the dominant procedure, leaving the scalability of currently used methods in question. Finally, the study reports that larger project are more negatively impacted by OSRs.

5.3.1 Main validity issues of RQ3

The main validity of RQ3 is related to construct validity threats. The phrasing of questions is a threat to construct validity. The authors of this paper and an independent native English speaker and writer-reviewer revised the questionnaire to alleviate this threat. To minimize the risk of misunderstanding or misinterpreting the survey questions, a pilot study was conducted on master students in software engineering.

The low statistical power threat to conclusion validity was addressed by using as suitable statistical tests as was possible on the given type of data. Before running the tests, we tested if assumptions of the statistical tests were not violated. Further, the reviews of the questionnaire and the pilot study addressed the instrumentation threat to internal validity.

5.4 Main contribution of RQ4

The main contribution of Paper IV, addressing research question RQ4, is improved understanding of causes and effects of overscoping. The results provide a detailed picture of overscoping as a phenomenon including a number of causes, root causes and effects. The results indicate that overscoping is mainly caused by operating in a fast-moving, market-driven domain. Continous requirements inflow, weak awareness of overall goals, in combination with low development involvement in early phases may contribute to ’biting off’ more than a project can ’chew’. Furthermore, overscoping may lead to a number of potentially serious and expensive consequences, including quality issues, delays and failure to meet customer expectations. Finally, the study indicates that overscoping occurs also when applying agile RE practices, though the overload is more manageable and perceived to result in less wasted effort when applying a continuous scope prioritization, in combination with gradual requirements detailing and a close cooperation within cross-functional teams.

5.4.1 Main validity issues of RQ4

The main threat to description validity regarding to RQ4 is misinterpretation of the interviewees. This threat was addressed by recording the interviews and performing observer triangulation on the transcripts and sending the transcripts back to the interviewees. The main threat to interpretation validity is the risk of imposing the hypothesis onto the interviewees. To address this threat, open interview questions were always posed before asking specific questions based on the hypothesis. The main threat to theory validity for this study is the risk of missing additional or alternative factors. One source of this threat is the limited set of practitioners from which data has been gathered. Another potential source is the risk of observer biases related to the researcher’s pre-knowledge of the company. The main threat to theory validity is the risk of missing additional or alternative factors. This threat was mitigated by conducting workshops with additional participants who suggested additional factors that may affect overscoping. Finally, we used analytical generalization to draw conclusions without statistical analysis and, under certain conditions, relating them also to other cases.

5.5 Main contribution of RQ5

The main contribution of Paper V, addressing RQ5, are the results from a statistical analysis of a large requirements engineering decision log and from a survey among industry participants. The results show that the number of products affected by a decision has a positive impact on the decision leadtime. Furthermore, the results in Paper V confirms that change requests issued by important customers are more likely to be accepted. Finally, Paper V suggests that the leadtime to reject a change proposal is longer than to accept it.

5.5.1 Main validity issues of RQ5

The main threat to internal validity is the number of investigate factors that may impact decision making in requirements engineering. This threat was minimized by investigating as many possible factors that could influence the decision lead-time and decision outcome as possible. The identified relationships were confronted with the results from the survey in which these relationships were further tested. The main threat to construct validity is the quality of the dataset used. The decision log used in the study was an archival record, which could be considered as stable, exact and quantitative. Whenever decisions in the log were incomplete or ambiguous, the inconsistencies were discussed them with the responsible product manager to avoid making wrong interpretations. To alleviate the threats to construct validity of the survey part of the study, an independent senior researcher experienced in the area reviewed the questionnaire. To address the main threat to external validity in the study, a survey was designed and executed among industry practitioners. The generalizability of the results is strengthen as the majority of the survey respondents worked with companies with a typical project generating not more than 100 requests.

5.6 Main contribution of RQ6

The main contributions of Paper VI, addressing RQ6, are the FSC+ technique to visualize scope changes in large-scale agile inspired projects, definition and analysis of decision patterns and definition of feature archetypes. The FSC+ scope visualizations are designed to be flexible and capable of visualizing the scope according to the current needs of practitioners. The sorting, filtering, color scheme and the time span of the charts can be adjusted and changed depending on project characteristics. The identification and empirical analysis of the most common decision patterns help to better understand the degree of adherence to the process guidelines and the reasons for not following the process in some cases. The atomic decision visualizations also allow analyzing the degree of process adherence. The identified decision patterns and decision archetypes enable projecting of the feature decision patterns in future based on the pattern matching principle.

5.6.1 Main validity issues of RQ6

The main threat to validity of RQ6 is the single company threat to external validity. The results and techniques presented in Paper VI are based on a single company case study and thus their transferability can be questioned. On the other hand, the size of the analyzed dataset and its diversity significantly increases external validity of the findings. The second main threat is related to conclusion validity. The evaluation is based on subjective opinions collected during interviews.

6 Further Research

Table 4 outlines the topics for further research. The overall research plan is intended to continue with the same general focus as presented in this thesis, namely to increase the understanding of various aspects of large and very large-scale requirements engineering decision making and to support some aspects with new methods or tools. Further, it is also important to continue the empirical investigation of VLSRE contexts to more precisely understand their nature.

6.1 FR1: Scoping by Controlling Opportunity Losses

Visualized in papers I and VI projects experienced continuous and frequent changes of the project scope. The visualized contexts were changing rapidly as the scope of the projects was adjusted to competitive pressures and market changes. However, the prioritization techniques reported in the literature Karlsson and Ryan (1997); Beck (2000); Leffingwell and Widrig (2003) model the scoping decisions as discrete decisions in time which can lead to the following issues: (1) the form of the scope decision is typically binary (keep or cancel) and final, (2) other options, such as investigate further, reschedule, decompose or refine are not considered and may impede flexibility in highly dynamic markets. The resulting continuous re-prioritization of the project scope creates a risk of uncontrolled scope changes and overscoping, see Paper IV.

In the presence of inevitable uncertainties and changes to the project scope, we present a novel way of looking at the task of requirements scoping. We postulate to consider the decision-making criteria as temporal functions, not absolute values, which facilitates their use even in dynamic

Table 4: Further research plans and ideas.

Further Research	Description	Research Approach
FR1	Scoping by Controlling Opportunity Losses	Further empirical case studies
FR2	Providing scalable requirements architectures	Case studies with the iMORE modeling framework
FR3	Additional investigations of supporting the human analyst and using the NCD for requirements traceability	Instrumenting systems and case studies
FR 4	Methods for handling OSRs	Prototype development and industrial case studies
FR 5	Investigation of overscoping in other contexts and the impact of agile practices on overscoping	Case studies focusing on investigation of ovescoping and the impact of agile practices on overscoping
FR 6	Investigation of additional factors that may affect decision lead-times and decision outcomes	Case studies focusing on investigating additional factors that could affect decision lead-times and decision outcomes
FR7	Extending the proposed visualization techniques on the system requirements level visualization. Improving the user interaction. Additional empirical evaluations.	

situations with uncertain scoping decisions. In this way, different management techniques, such as introducing excess numbers of features into the process and allowing the requirements process to find the strongest, can be addressed.

The LOEM (Lost Opportunity Estimation Model) model is based on the assumptions that the following factors are temporal functions:

- *Market value* An innovative feature conveying a competitive advantage may have significantly greater value than, for example, a maintenance feature.

- *Development costs* Development costs accrue at different rates, at different phases of the project. Scoping decisions can include risk analyses if these costs are known.

6.1.1 The Value and Cost Functions

The value function $V(t)$ is the estimated market value, expressed as follows:

$$\int_a^b V(t)\,dt \tag{1}$$

where $t = a$ and $t = b$ are the times when the functionality starts to have market value and the functionality cease to have any value in the market.

The cost function $C(t)$ is the summation of the individual cost factors over time, expressed as follows:

$$\int_a^b C(t)\,dt \quad where \quad C(t) = \sum_{k=1}^{n} \int_a^b C_k(t)\,dt \tag{2}$$

where a is the beginning of the work on a certain feature and b is either the end of implementing the feature or the moment when the feature was removed from the scope. k is the number of cost factors included in the estimate, e.g. requirements management or development costs.

6.1.2 Return on Investment Threshold

We propose that features under investigation should be kept within project scope until an ROI threshold level is reached. If the feature ROI is less than the ROI threshold, the feature is canceled. The ROI threshold, δ is not simply value/cost – it generally includes many other business factors such as probability of market success, based on past performance, and analyses of projected market conditions at the time of product release.

$$\frac{\int_a^b V(t)\,dt}{\int_a^b C(t)\,dt} \leq \delta \tag{3}$$

The flexibility inherent in the development process can be manipulated by adjustments to the value of δ. We note that the value of δ may not be the same for different types of features – speculative features may need larger δ values than infrastructure or enabling features, see equation 3.

6.1.3 Feature Impact

The overall impact of a set of features within a development cycle can be calculated using the following equation:

$$\sum_{k=1}^{K} \left(\frac{\int_a^b V_k(t)\, dt}{\int_a^b C_k(t)\, dt} - \delta_k \right) \tag{4}$$

where K is the number of features analyzed, a is the point in the project when the feature was added and b when the feature was canceled. The values for a and b are not the same for different k. If the feature impact is negative, then the project manager can not expect that the product cycle will be profitable as a whole. A negative value for feature impact is generally acceptable only if there are enough infrastructure features (whose true value is deferred) to offset revenue generating features.

6.1.4 Initial LOEM Model Validation

The utility of this model was initially investigated using data from the same large industrial project as in the rest of this paper. The project has 223 features and 120 features were canceled before the end of the analysis period. The results from the initial validation shows that so much as 77% of the total effort spent on features that were descoped before the end of the requirements management process could have been saved if the project management will immediately decide to descope each feature that passes the critical level of 1,5 RIO threshold. Based on the promising results of the initial validation, we believe that the model can be applied to both process improvement and postmortem analysis of scoping decisions. For more details regarding the model, see Paper XV.

6.1.5 The Basic Lost Opportunity Estimation Model (BLOEM) for Requirements Scoping

The Basic Lost Opportunity Estimation Model is a model dedicated to companies that use agile-inspired software development methods. The model supports requirements scoping in the environments where the time-dependent business value estimates change as the requirements analysis process progresses. Companies utilizing agile-inspired development models attempt to increase requirements process flexibility and process responsiveness to unexpected changes in scope using continuous scope update and one-dimensional (relative) market-value estimates as a substitute for real values (Racheva et al, 2010). The BLOEM attempts to control wasted effort by facilitating earlier identification of feature cancellation candidates, promoting constructive use of resources. The model enhances existing processes by providing input to the keep/cancel scoping decision.

The BLOEM model operates on a single return on investment calculation represented by the *value function* which can be relative or absolute value for candidate features. The model is based on the market-driven requirements engineering premise that the value of a requirement is a temporal function that is sensitive to market forces and opportunities - often a feature will only have market value for a limited time. Even features that offer unique capabilities see a significant reduction in their market value when competitors catch up and offer the feature in their own products.

The total value $V(t)$ is represented is the same way as in the LOEM model, see equation 1. The value function will depend on the characteristics of the target market and must be estimated when applying the model.

The BLOEM model assumes that features under investigation should be kept within the project scope until a defined value threshold (δ), known as the Final Decision Point (FDP), is reached. The threshold value can be unique to each feature and should be estimated per feature. High-value features (e.g. priority in the top 25%) could have the FDP threshold set higher than less valuable features (e.g. priority in the bottom 25%). Final decisions as to whether to keep (and realize the investment) or cancel (and minimize losses) are then delayed for the most valuable features while the least valuable features are canceled relatively early. The FDP can be used as to enforce a budget-like approach to the scoping management process.

We consider all canceled features to be wasted effort. However, investments in features that are canceled before the threshold are considered *controlled waste*: there is waste but it is under management control and the risk of inter-feature dependencies is held to an acceptable level. Features that are canceled after the final decision point are *uncontrolled waste* - something unexpected has happened and time or resource constraints cannot be met for this release cycle.

The flexibility of the decision process can be adjusted by tuning the value of δ. The *overall impact* of a set of **K** of withdrawn or canceled features is calculated using formula 5:

$$\sum_{k=1}^{K} \frac{\int_a^b V_k(t)\,dt - \delta_k}{K} \tag{5}$$

6.1.6 Initial BLOEM Model Validation

BLOEM was initially validated using a set of 166 features analyzed by a very-large company that utilizes agile-inspired development methods. The feature status in the data set ranged from the definition phase, through implementation, to completion. During this period 87 features were canceled. Under the assumption of the constant value function, the average uncontrolled waste was 10.2% of the normalized value for the 25 features that were withdrawn after their final decision point. Under the assumption of the normal value function, the average uncontrolled waste was 20.3% of the normalized value for the 4 features that were withdrawn after their final decision point. The 25 features remained in the process for a cumulative 1021 days after the FDP while the 4 features remained in the process for a cumulative 75 days after the FDP. The resources expended upon these features during this period represent both direct costs and lost opportunity costs. For more details, please refer to Paper XXIV.

6.2 FR2: Providing Scalable Requirements Architectures

The amount of data in large-scale software engineering contexts continues to grow and it challenges the efficiency of software engineering efforts. At the same time, requirements and associated information are key elements of successful software projects. For large and very-large projects, designing scalable requirements architectures is one of the strategic challenges to tackle, see Section 2.

We suggest a framework for requirements information modeling. The iMORE framework (information Modeling in Requirements Engineering) is developed with collaboration with our industry partners in an action research mode (Robson, 2002). The framework is based on the distinction between the external information structures and internal information structures, outlined in Figure 6 by a dashed line. The inclusion of the external information structures was stressed as *very important* by the industrial practitioners during the development of the model. This is caused by a large and continuously changing number of sources of requirements and other information types that directly interact with the company, including competitors, suppliers, open source components and other partners. The need to access external information is valid of all abstraction levels of the model; from source code to product portfolio documents.

The information is divided into three main blocks: the upstream, the requirements and the downstream blocks. In the 'upstream block' all high-level information is stored, including the goals, strategies and business needs. In the 'requirements block' all requirements associated information is stored, including functional requirements, quality requirements, constraints, legal requirements and regulations. The source code, bug reports, code documentation and other related information is stored in the 'downstream' block.

The last main element in the iMORE framework is handling temporal aspect of the information structure, depicted by a vertical arrow in Figure 6. The temporal aspects include capturing the evolution of the data models in terms of the evolution of the artifacts and their associated structures. To

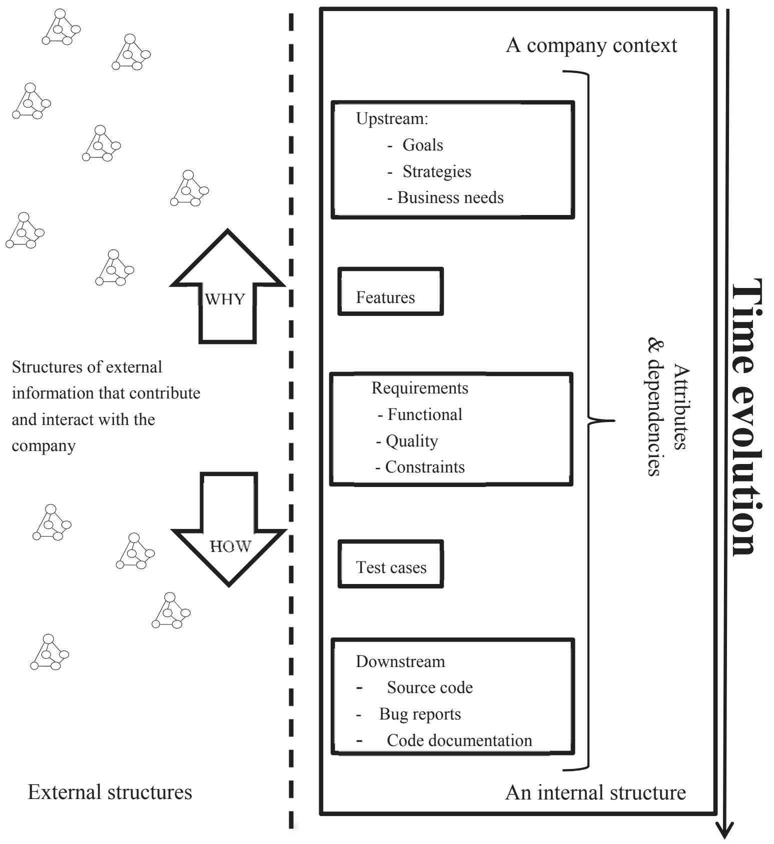


Figure 6: The iMORE modeling framework.

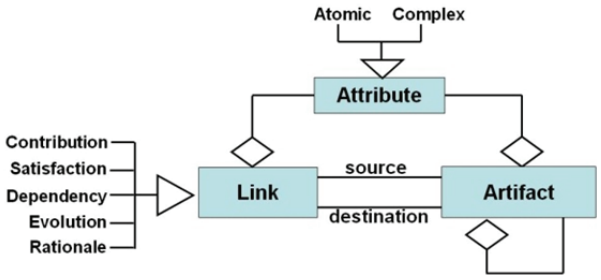


Figure 7: The iMORE meta-model.

manage this aspect, the underlying meta-model defines 'Evolution' type of links between two artifacts. Using this category of links, users can handle the evolution over time of artifacts and their structure.

A simple traceability meta-model derived from related works (Ramesh and Jarke, 2001) and previous research (El Ghazi and Assar, 2008) was used to design the information structure. The information structure is depicted in Figure 7. The structure of an element to be stored in the repository and to be traced in the software project is constructed using two generic concepts: artifact and attribute. An attribute can be an atomic element of information (e.g. owner, release date, version number, URL) or any complex structure (e.g. list of modification dates). The set of attributes is not only limited to a particular block of information but may also cover several blocks or even the entire information structure creating a set of 'global' attributes. Finally, any artifact in the repository can be linked to other artifacts. Five categories of links are predefined in the iMORE meta-model; they are briefly explained using the following examples:

- A requirement document A **contributes** to the specification of a design feature B

- A design feature A **satisfies** an external law based constraint B

- A design feature A **depends** on another design feature B

- A design specification A is the result of the **evolution** of a design specification B

- An external law based constraint A is the **rationale** for a requirement document B

The initial evaluation of the iMORE framework was conducted with industry experts at three companies. All five respondents confirmed the need for modeling requirements information in a more findable and understandable way. Further, all respondents agreed to the distinction and stressed that external information currently dominated their daily work. Respondents suggested what information and when should be integrated with the model, see Paper XXVI for more details.

One of five respondents suggested that managing the temporal aspect of the information structure isn't so important. Another respondent suggested creating an attribute called "'period of validity"' for managing the temporal aspect. Two respondents suggested using triggers based on attributes to manage the temporal aspect. Finally, one respondent suggested a method based on combining baselines and trigger-based updates. For more details regarding the iMORE model and the initial evaluation, please refer to Paper XXVI.

6.3 FR3: Additional investigations of possible usage of a linguistic tool support for requirements management related tasks.

Paper II presents the results from an evaluation of linguistic support for identification of similar requirements. The natural language processing field can provide a number of other techniques that can automatically analyze natural language documents. For example, Latent Semantic Indexing (De Lucia et al, 2007), probabilistic models (Rayson et al, 2000), or swarm techniques could be used to analyze requirements (Sultanov and Huffman Hayes, 2010). Future research is planned to focus on unsupervised natural language processing methods, for example clustering (Duan et al, 2009a) or searching methods, that may provide valuable help with impact analysis. Finally, it is also planned to test the above mentioned techniques for other relevant tasks in large-scale requirements management, e.g. creating data minable regulatory codes, see Paper XXII for more details.

6.3.1 Supporting the Human Analyst

Another possible alley for future research is to focus on the manual part of the process when the human analyst need to analyze candidate links and determine which of them are true link and which are false positives. Recent work by Huffman Hayes et al (2010) focuses on the role of human analyst in requirements traceability. What is missing in their work is the ability of the automated traceability method to show to the analyst the difference between the true positive and the false positive. This brings us to the challenging problem of relevance in information science (Borlund, 2003).

Several measures were proposed to better evaluate IR methods. Järvelin and Kekäläinen (Järvelin and Kekäläinen, 2002) proposed measures based on cumulated gain, combining the degree of relevance and rank. They give an indication about the quality of a method, but does not evaluate the support for decision making. Spink and Greisdorf (Spink and Greisdorf, 2001) suggested the median effect as a measure to evaluate the way the distribution of relevance judgments of retrieved items are generated by a method. However, this method is mainly focused on being an alternative to dichotomous measures. Kekäläinen and Järvelin (Kekäläinen and Järvelin, 2002) also identified the weakness of just evaluating binary relevance as is the case for recall and precision, and they proposed generalized recall and precision, which reward IR methods that extract more relevant items. These measures also do not evaluate how easy it is to make decisions. Other measures, e.g. expected search length (Cooper, 1968), normalized recall measure (Rocchio, 1966), sliding ratio measure (Pollack, 1968), satisfaction-frustration-total measure (Myaeng and Korfhage, 1990) can all be used to credit methods presenting relevant items high up the list, but again the evaluation of decisions making support is not targeted.

Our approach to the problem above is to introduce the Signal-to-Noise Ratio (SNR), a measure widely used in science and engineering to quantify how much a signal has been corrupted by noise, to information retrieval research and requirements engineering.

$$SNR = \frac{P_{signal}}{P_{noise}} \tag{6}$$

$$SNR = \frac{mu}{sigma} \tag{7}$$

In its classical definition, see equation 6, signal-to-noise is a ratio between the average power of the signal (P_{signal}) and the average power of the noise (P_{noise}) (Gonzalez and Woods, 2006). Since signals have usually a very wide dynamic range, the logarithmic decibel scale is used to express SNR. There exist alternative definitions of SNR, like for example the ratio of mean (mu) to standard deviation (sigma) of a signal or measurement, see equation 7. This definition is commonly used in image processing (Gonzalez and Woods, 2006; Stathaki, 2008; Raol, 2009; Russ, 1999) where the SNR of an image is usually calculated as the ratio of the mean pixel value to the standard deviation of the pixel values over a given neighborhood.

The higher the ratio, the less obtrusive the background noise is. In other words it is a ratio between the meaningful information and the background noise. In our case, the signal is represented by the correct link between two objects while the noise is represented by all other potential links.

Precision and recall are widely used statistical classifications. They can be seen as measures of exactness or fidelity (precision) or completeness (recall) (Baeza-Yates and Ribeiro-Neto, 1999). If every result retrieved from the task is relevant, we can say that the precision is equal to 1. However, the precision value 1 does not say if all existing relevant documents were retrieved. Therefore, if all relevant documents are retrieved by the task the recall will be 1. Similarly as for the precision, the recall value 1 does not say anything about how many irrelevant document were also retrieved with all relevant. These measures can be suitable for measuring the quality of the result of the automatic classification task, but do not assume that someone, for example a requirements analyst will use the results as a list of possible candidates in linking similar objects (Natt och Dag et al, 2006). In this case, the measures described above give only an indication of the entire result set, without assessing the quality of how distinctive the correct answers are from the incorrect ones. We assume that this support can help the human analyst to assign more correct links and minimize the number of false positive links (Natt och Dag et al, 2006).

6.3.2 Rationale for defining Signal-to-Noise ratio

To illustrate the rationale for defining yet another measure, we use the example of linguistic tool support for requirements consolidation from Paper II. In this case, a similarity measure was used to propose a list of requirements candidates and their similarity score to one actually analyzed. The requirements analyst uses the resulting list of candidates to assign links between two or more similar requirements. In the replicated experiment performed in Paper II, a set of 30 requirements has been analyzed against 160 other requirements. The key with correct answers, consisting of 20 links, was prepared before the experiment. Once the correct links were assigned they have been checked with the output of the tool to assess which position on the candidate list they will be classified.

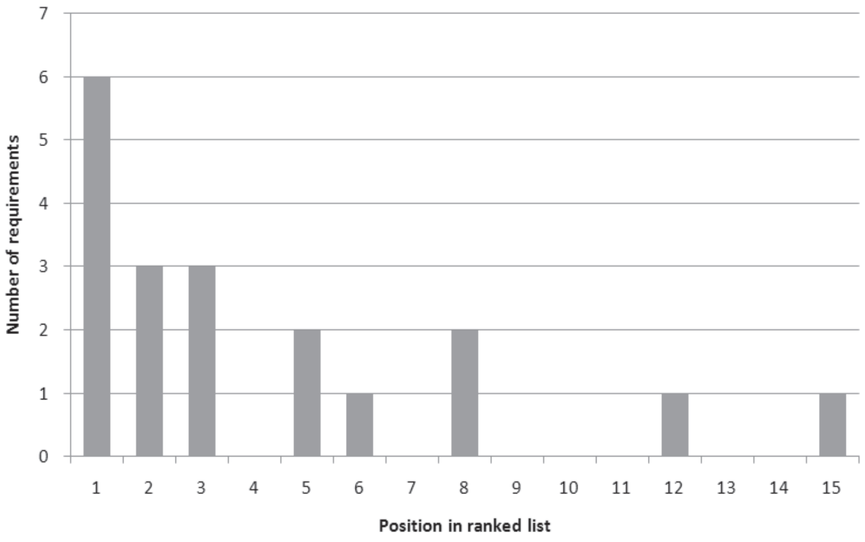


Figure 8: Histogram of the positions of similar requirements in the automatically produced ranked list of similar requirements (Natt och Dag et al, 2006)

Figure 8 depicts a histogram of the distribution of the positions at which the correct links appear on the ranked list of candidates produced by the tool Natt och Dag et al (2006). For the data set used in Natt och Dag et al (2006), almost all (17 out of 20) of the correct answers end up at position 8 or better in the ranked list, and could therefore be quickly spotted. The question that remains unanswered here is what are the similarity scores for all the incorrect links proposed on the list (false positives). For example, if a correct answer has position 8 on the list, what are the similarity values for the 7 other answers ranked as more similar than the correct answer. These false positives need to be analyzed in order to find the correct link. The decision which links to reject may be hard if the similarity scores of all incorrect but highly ranked requirements are very similar to the score of the correct answer. In other words, the analysts may get mislead by the fact that the *noise* is highly ranked than the *correct signal*.

We propose three versions of Signal-to-Noise that can be applied for evaluation of automatic methods for supporting decision-oriented tasks in software and requirements engineering. The first measure, called *Individual SNR* is the ratio between the correct answer and the closest incorrect answer on the list of candidates, see equation 8. In a case when the correct answer is not ranked as number one on the list of candidates, the *Individual SNR* is an average value of the two closest incorrect proposals, one higher than the correct one and one lower than the correct answer. The individual SNR describes the ability of spotting the correct answer among the closest proposed incorrect answer(s).

$$SNR_{individual} = \frac{score\_corr}{(score\_incorr\_down + scope\_incorr\_up)/2} \tag{8}$$

The second version of SNR we propose is the *Maximum Noise SNR* which is the ratio of the similarity degree of the correct answer of the candidates list to the maximum value of similarity of the incorrect answer presented in the list of candidates, see equation 9.

$$SNR_{Max} = \frac{score\_corr}{\sum\limits_{MAX} score\_incorrect} \tag{9}$$

The third proposed version of SNR is called *Average SNR*. It is defined as the ratio between the value of similarity of the correct answer to the average noise ratio value above a certain threshold, see equation 10. The threshold in Figure 8 has been set to 15, which means that candidate requirements that scored below number 15 on the candidate list were not considered. The average SNR definition considers a similarity value or any other measure used rather than a certain number on the ranked list as the threshold.

$$SNR_{Avg} = \frac{score\_corr}{\frac{\sum\limits_{i=1}^{n} score\_incorrect}{nbr\_of\_incorrect\_hits}} \forall i < threshold \tag{10}$$

6.3.3 Empirical Application of SNR

The initial evaluation of the SNR measures was performed on the same dataset as used in Paper II. The analysis was performed for two methods of measuring the similarity between the requirements: (1) the linguistic method based on Vector Space Model (Natt och Dag et al, 2005; Natt och Dag, 2010) and (2) the Normalized Compression Distance (NCD) method based on information theory (Cilibrasi et al, 2012). The set contains 30 requirements that are checked against 160 requirements. There exist 20 correct links between the requirement sets. Knowing which links are correct, we compared the values of the similarity scores for the correct and incorrect answers.

The results from measuring the three types of SNR are presented in Tables 5 and 6. For the average SNR calculations we use similarity value 0.5 as the threshold value. The very low similarity measures of the NCD forced us to lower the threshold to 0.4 for this method, accepting that direct comparisons no longer are possible. Requirement R19, the only quality requirement in the dataset, was in all calculations considered as an outlier and disregarded. Its format and structure leads to too high similarity values.

A quick comparison between the two techniques is presented in Table 7. It is rather clear that both methods provide poor support for the analyst who has to make decisions supported by the similarity value. The SNRs are in all cases close to 1 and the correct link is in the most cases not the first on the list. The linguistic VSM based approach (Natt och Dag et al, 2005; Natt och Dag, 2010) has in all cases higher SNR than the normalized compression distance method.

As it can be seen in Table 5 only for one requirements (R19) the SNR is more than 2. The is the above mentioned quality requirement that was consider as an outlier. In all other cases the values range from 0.69 to 1.2, depending on the type of the SNR. The *Na* values in Tables 5 and 6 for the AvgSNR corresponds in this case to the situation when the measurement could not be performed: for example in case of R9 and R16 both the correct answer and the highest noise were below the threshold 0.5. Moreover, for some data points, the values for MaxSNR and AvgSNR are identical (R3 and R17), which means in this case that there are only two data points above the threshold and the top candidate is the noise (R3 case) or a correct signal (R17). The equal values for both IndSNR and MaxSNR indicate that the correct answer is number one on the list of candidates, see R2, R5, R11, R17, R18 and R19 in Table 5. The average value for IndSNR is slightly higher (1.035) than for MaxSNR (0.93) and AvgSNR (0.974) 7. Finally, the biggest median value represents results for AvgSNR (1.06), comparing to IndSNR(1.009) and MaxSNR(0.94)

The results for measuring all three types of SNR for the Normalized Compression Distance method of assessing the similarity between requirements are more dispersed, see Table 6. The values range from 0.28 to 1.04. The positions where the correct answer ended up on the list of candidates are much lower than for the linguistic similarity measure method, only for 4 cases it is within the top 10 candidates. The average and the median for

Table 5: The results from measuring all three types of SNR for the ReqSimile tool that uses the VSM model.

Requirement	IndSNR	MaxSNR	AvgSNR	Pos. on the list
R1 (SC13)	1.04	0.98	1.16	2
R2 (41104)	1.26	1.26	1.34	1
R3 (41112)	1.12	0.97	0.97	2
R4 (41114)	1	0.68	0.68	3
R5 (41123)	1.04	1.04	1.14	1
R6 (41301)	1.005	0.89	0.95	5
R7 (41307)	1.009	0.57	0.62	64
R8 (41309)	0.97	0.79	0.83	15
R9 (41414)	1.02	0.91	Na	3
R10 (41601)	0.99	0.95	1.19	3
R11 (41606)	1.008	1.008	1.07	1
R12 (41608)	0.99	0.94	1.06	5
R13 (41710)	1.02	0.89	0.92	8
R14 (41804)	1.009	0.94	1.20	6
	1.001	0.91	1.20	8
R15 (41811)	1.001	0.89	1.06	12
R16 (4205)	1.01	0.89	Na	2
R17 (43302)	1.12	1.12	1.12	1
R18 (43303)	1.05	1.05	1.05	1
R19 (43402)	2.7	2.7	Na	1

Table 6: The results from measuring all three types of SNR for the tool using the NCD model.

Requirement	IndSNR	MaxSNR	AvgSNR	Pos. on the list
R1 (SC13)	1.03	0.93	0.97	4
R2 (41104)	1	0.49	0.49	111
R3 (41112)	1	0.75	Na	20
R4 (41114)	1.06	0.84	Na	5
R5 (41123)	1.01	1.01	1.04	1
R6 (41301)	0.99	0.42	0.48	155
R7 (41307)	1.003	0.66	0.72	97
R8 (41309)	0.998	0.84	0.86	57
R9 (41414)	1	0.597	0.62	134
R10 (41601)	1.007	0.66	0.79	80
R11 (41606)	1.01	0.8	Na	52
R12 (41608)	0.98	0.85	0.92	13
R13 (41710)	1	0.89	1	9
R14 (41804)	0.993	0.91	0.99	14
	1	0.77	0.84	73
R15 (41811)	1	0.73	NaN	95
R16 (4205)	0.93	0.507	0.57	92
R17 (43302)	0.99	0.564	0.564	41
R18 (43303)	1	0.537	0.537	29
R19 (43402)	0.996	0.28	0.389	136

Table 7: Comparison of the three SNR measures and position of the correct link for VSM and NCD

	VSM	NCD
Average IndSNR	1.035	1
Median IndSNR	1.009	1
Average MaxSNR	0.930	0.724
Median MaxSNR	0.94	0.75
Average AvgSNR	0.974	0.759
Median AvgSNR	1.06	0.75
Average Position	7.526	56.947
Median Position	3	52

the IndSNR are the highest among all types of SNR.

The results presented above, although preliminary, give a clear indication that the studied method provide poor support for the human analyst. Confronted with very hardly distinguishable similarity values, the analyst has to review a large amount of candidate links in order to find the correct answers. This creates a risk for additional false positives, which is actually confirmed by Cuddeback et al (2010). Initial results clearly indicate that the analyst could have difficulties in finding the right answer when looking at the similarity values. Thus, future work is needed to better understand how IR methods can better support human analyst judgments.

6.4 FR4: Methods for handling OSRs.

Paper III reported that OSRs have a serious impact on companies developing software intensive systems. At the same time, 73.6% of our respondents reported that their requirements engineering process does not take OSRs into consideration. Therefore, future work should be directed towards developing requirements management processes that take OSRs into consideration.

Paper III also reports that identification and management of OSRs is predominantly a manual activity, and less than 10% of the respondents reported having any automated functionality for OSRs identification. Thus, it is planned to further research scalable methods and tools for automated identification and management of OSRs.

One avenue for future research is to integrate identification of obsolete elements into the iMORE modeling framework, see Section 6.2. Using the "'period of validity"' attribute could be one of the possible mechanisms of managing the temporal aspect of requirements. Furthermore, creating a trigger that indicates when a requirement is implemented or integrated in a product, or has been integrated for the last 2 releases or more may help to identify OSRs. Finally, identifying potentially obsolete requirements among requirements that were not changed, edited or modified could be one of the possible ways to manage them.

Another interesting avenue for future research is to investigate the feasibility of using garbage collection (McCarthy, 1960) methods to manage obsolete requirements. Several possible algorithms for garbage collection were proposed, e.g. references counting (Collins, 1960), "mark and sweep" algorithm (McCarthy, 1960) and the copying algorithm (Minsky, 1963). The references counting and "mark and sweep" algorithms need to analyze each allocated block of memory, which in requirements case would be all requirements in the requirements database. Another challenging problem with applying these methods for requirements garbage collection is the fact that they rely on pointers pointing at the memory blocks. Thus, it seems that accurate traceability is a prerequisite of requirements garbage identification using the above methods. Since both establishing (Hayes Huffman et al, 2003) and maintaining (Cleland-Huang et al, 2003) traceability links are considered challenging, this may negatively impact the accuracy of the garbage collection algorithms. The issue of accurate traceability links need to be resolved while applying the copying algorithm for requirements garbage collection, while copying the objects from the "‘active objects"’ space to "‘garbage"’ space.

Two additional and interesting issues should be investigated and solved when attempting to use classical garbage collection algorithms on requirements. First, how to know when to run the garbage collection? The "‘full memory"’ trigger to start garbage collection can’t be directly reused as the requirements databases get rarely full. Thus, it appears to be more logical to start requirements garbage collection when a number of requirements triggers are activated, based on certain attributes or other criteria. Second, an issue remains about how to categorize requirements that do not have traceability links. The dataset used in Paper II contained these types of requirements. Although not linked with other requirements or other documents, they could still be needed for the project purposes and thus should not be garbage collected.

6.5 FR5: Investigation of overscoping in other contexts and the impact of agile practices on overscoping.

Further work in relation to Paper IV and research question RQ4 focuses on more case studies that investigate overscoping at other companies. It is especially interesting to investigate overscoping at companies that use agile-inspired processes and methods. The questionnaire respondents in Paper IV mostly agreed that the three identified agile requirements engineering practices may positively impact the challenge of overscoping. Further, the respondents suggested that overscoping is still present in the new (agile) way of working at the case company, thus it is more manageable. Therefore, additional investigations are necessary to better understand how the iterative requirements engineering (Cao and Ramesh, 2008) helps to avoid overscoping.

The impact of other aspects such as organization set up, the involvement of development teams in early phases of requirements definition and communication on overscoping is also planned to be investigated in the future. In related paper XX, a case study aiming at a deeper understanding the causes and effects of communication gaps in large-scale industrial set up is presented. We discovered that communication gaps are caused by the following factors: *scale, common view, temporal aspect* and *decision structures*.

The size and complexity of the software development increases the challenge of requirements communication. Study XX discovered communication gaps between the requirements engineers and a number of stakeholders, result in missing requirements, e.g. for quality. Instead, these requirements surface in later phases, thus, incurring increased cost. *Common views* and mutual understanding are necessary for communication to be productive. Weak understanding of each other's roles and responsibilities causes gaps in communication. For example, the testers' competences are not utilized when defining and reviewing requirements, or the requirements engineers are not consulted when making implementation choices that affect the requirements.

Temporal aspects can came into play when there is a lack of continuity in requirements awareness through the project life cycle. This may cause gaps in the requirements communication. Hand-over points, e.g. defined by the process, where the responsibility is passed on to new roles constitute a risk of missing vital requirements knowledge and awareness. This may result in requirements being misunderstood and incorrectly implemented, or, making decisions that affect the requirements without considering all relevant aspects. For example, if there is no requirements awareness in the implementation phase, the developers tend to make their own requirement modifications without considering the impact on the customer or on other parts of the development organization, such as test.

Decision structures also contribute to communication gaps. Weak, or unclear, visions or goals for the software development (due to not being communicated or not being clear enough) contributes to weak communication, primarily, between those defining the requirements and the development unit, since there is no mutual understanding of the goal.

Paper XX also reports that communication gaps can have serious and expensive consequences in terms of wasted effort and quality issues, as well as, not meeting the customers' expectations and even communicating an incorrect picture of what requirements a product fulfills. Furthermore, communication gaps can contribute to overscoping and to even more communication gaps, i.e. the software development ends up in a "'vicious cycle"'.

Another interesting avenue for future research about overscoping is to investigate the impact of cost and schedule estimations on overscoping. Achieving correct cost and schedule estimation at the high-level (in agile projects) is considered challenge (Ramesh et al, 2010). Thus, transitioning into agile-inspired processes may not directly mitigate the negative impact of inaccurate cost estimations on overscoping. In a related study, Jørgensen et al (2004) discovered that too optimistic estimates may lead to effort overruns (in other words overscoping) for projects where complete specification and quality were prioritized. This relates to one of the causes of overscoping discovered in Paper IV, namely detailed requirements specification produced upfront, cause C5. In another study, Jørgensen et al (2012) studied the imperfect relation between the actual and the estimated cost, concluding that projects with the size measured based on the actual cost report an increase in cost overrun with increase project size. Thus may suggest that for very-large projects, the cost overrun will be quite high, which confirms findings from Paper I. Finally, as reported by Jørgensen, (2004), practitioners perceived factors outside their own control as a reason for effort estimates errors.

6.6 FR6: Investigation of additional factors that may affect decision lead-times and decision outcomes.

Two of the investigated in Paper V relationships turned out to be statistically significant and confirmed by the survey respondents: (1) the lead-time to make a decision increases when more product are affected by a decision and (2) change requests issued by important customer are more likely to be accepted. Therefore, future work should be directed towards investigating why the other investigated relationships turned out to be not statistically significant.

One interesting avenue for future research is to investigate the relationship between time to make a decision and the importance of the customer. 62.8% of survey respondents indicated that time to make the decision is shorter when the decision is filled by an important customer. In a related study, Taylor et al (Taylor et al, 2011) reported that large customers more frequently get their requests implemented. Paper V confirms this statement both in the statistical analysis and in the survey. Therefore, it is a subject of future work to further explore if this statement can be extended to the decision outcome as well as if this relationship holds for further other datasets with more balanced ratio of internal to external errands.

Another interesting avenue for future research is to further investigate if the release date or number may influence the decision lead-time and outcome. Paper V could not statistically confirm that this factor affects decision lead-time or outcome due to conflicting results between the statistical analysis and the survey. At the same time, the release planning literature Ruhe (2003); Ruhe and Greer (2003); Ruhe and Saliu (2005); Ngo-

The and Ruhe (2005b,a); Ruhe (2009); Karlsson and Ryan (1997); Bagnall et al (2001) implicitly suggest that the next release is the most important and thus decision lead-times should be shorter for errands regarding this release. Thus, we believe that future investigations are needed to resolve the role of the release number in decision making efficiency and outcomes.

Additional factors such as the number of stakeholders involved or the number of dependencies between software components could also affect decision lead-times and outcomes. Furthermore, future work is planned to formulate a function of estimated lead-time in relation to the decision outcome based on the identified impacting factors and gathered empirical data. Since the average lead-time needed to reject a decision is statistically significantly longer that the lead-time needed to accept a decision, see Paper V for more details, there should exist a "tilting point" in time after which most change requests are rejected. We believe that this tilting point is related to the average time that a product generated value, the characteristics of the customers and stakeholders, the development model and the release frequency used by a company. These assumptions should be investigated together with the final decision point concept presented in related Papers XV and XXIV.

6.7 FR7: Extending the proposed visualization techniques on the system requirements level visualization. Improving the user interaction. Additional empirical evaluations.

A carefully designed visualization could assist with for example a typical requirements comprehension problem of gaining a quick assessment on the ”health” of a set of requirements (Gotel et al, 2007). This task is usually impeded by the need to browse through disjoint textual requirements documentation and accompanying models. The visualizations presented in Papers I and VI bring a quick and clear assessment on the scoping process for large projects, but can also be a base for more in-depth analysis while envisioning details about the scoping process. Thus, additional studies on finding effective visual means of scope dynamics visualization are in the agenda of further research.

Especially, the research is planned to be focusing on providing useful visual means that can help project and product managers to quickly assess the efficiency of the scoping process in terms of resource situation and scope capacities. Furthermore, a more in-depth study that aims for defining additional scope tracking measurements and applying them into the case company context data, is planned. Finally, additional studies on finding optimal visual metaphors for complementary aspects of scoping are planned as further research topics.

Bibliography

Abramovici M, Sieg OJ (2002) Status development trends of product lifecycle management systems. In: Proceedings of International Conference Integrated Product and Process Development, pp 55–70

Aguilera C, Berry D (1991) The use of a repeated phrase finder in requirements extraction. Journal of Systems and Software 13:209–230

Albourae T, Ruhe G (2006) Lightweight replanning of software product releases. In: Proceedings of the 1st International Workshop on Software Product Management (IWPSM 2006), pp 27–34

Andriole S (1998) The politics of requirements management. IEEE Software 15(6):82 –84

Antoniol G, Canfora G, Casazza G, De Lucia A, Merlo E (2002) Recovering traceability links between code and documentation. IEEE Transactions on Software Engineering 28(10):970 – 983

Asuncion H, Arthur U, Taylor U, Richard N (2010) Software traceability with topic modeling. In: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1, ACM, New York, NY, USA, ICSE '10, pp 95–104

Aurum A, Wohlin C (2002) Applying decision-making models in requirements engineering. Information and Software Technology 45(14):2–13

Aurum A, Wohlin C (2003) The fundamental nature of requirements engineering activities as a decision-making process. Information and Software Technology 45(14):945–954

Aurum A, Wohlin C (2005) Engineering and Managing Software Requirements. Springer

Avesani P, Bazzanella C, Perini A, Susi A (2005) Facing scalability issues in requirements prioritization with machine learning techniques. Paris, France, pp 297 – 305

Avison D, Fitzgerald G (1998) Information systems development methodologies techniques, and tools. John Wiley

Baeza-Yates R, Ribeiro-Neto B (1999) Modern Information Retrieval. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA

Bagnall A, Rayward-Smith V, Whittley I (2001) The next release problem. Information and Software Technology 43(14):883 – 90

Bakalova Z, Daneva M, Herrmann A, Wieringa R (2011) Agile requirements prioritization: What happens in practice and what is described in literature. In: Requirements Engineering: Foundation for Software Quality. Proceedings of the 17th International Working Conference, REFSQ 2011, Berlin, Germany, pp 181 – 95

Ball T, Erick S (1981) Software visualization in the large. IEEE Computer 29(4):3–14

Balzer R (1981) Transformational implementation: An example. IEEE Transactions on Software Engineering 7(1):3–14

Basili V (1996) The role of experimentation in software engineering: Past, current and future. In: Proceedings of the 18th International Conference on Software Engineering (ICSE 96), pp 442–449

Basili V, Rombach D (1988) The tame project: Towards improvement-oriented software environments. IEEE Transactions on Software Engineering 14(6):758–773

Beck K (2000) Extreme Programming Explained: Embrace Change. Addison-Wesley

Berander P (2004) Using students as subjects in requirements prioritization. In: Proceedings of the 2004 International Symposium on Empirical Software Engineering, pp 167–176

Berenbach B, Paulish D, Kazmeier J, Rudorfer A (2009) Software & Systems Requirements Engineering: In Practice. Pearson Education Inc.

Bergman M, King J, Lyytinen K (2002) Large-scale requirements analysis revisited: The need for understanding the political ecology of requirements engineering. Requirements Engineering Journal 7(3):152–171

Berntsson Svensson R (2009) Managing quality requirements in software product development. Licentiate Thesis

Berntsson Svensson R (2011) Supporting release planning of quality requirements: The quality performance model. PhD thesis, Lund University, Sweden

Beydeda S, Book M, Gruhn V (2005) Model-Driven Software Development. Springer-Verlag

Bhat J, Gupta M, Murthy S (2006) Overcoming requirements engineering challenges: Lessons from offshore outsourcing. IEEE Software 23(5):38–44

Biffl S, Thurnher B, Goluch G, Winkler D, Aigner W, Miksch S (2005) An empirical investigation on the visualization of temporal uncertainties in software engineering project planning. In: Proceeding of the International Symposium on Empirical Software Engineering (ISESE 2005), pp 437–446

Boehm B (1988) A spiral model of software development and enchancement. Computer 22(5):61–72

Boehm B (1989) Tutorial: Software risk management. Tech. Rep. ISBN-0-81 86-8906-4, IEEE Computer Society

Boehm B (2006) Some future trends and implications for systems and software engineering processes. Systems Engineering 9(1):1–19

Booth R, Regnell B, Aurum A, Jeffrey R, Natt och Dag J (2001) Market-driven requirements engineering challenges: An industrial case study of a process performance declination. In: Proceedings of the 6th Australian Workshop on Requirements Engineering (AWRE 2001), pp 41–47

Borlund P (2003) The concept of relevance in ir. J Am Soc Inf Sci Technol 54(10):913–925

Brinkkemper S (2004) Requirements engineering research the industry is and is not waiting for. In: Proceedings of 10th Anniversary International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ 2004), pp 41–54

Cao L, Ramesh B (2008) Agile requirements engineering practices: An empirical study. IEEE Software 25:60–67

Carlshamre P (2002a) Release planning in market-driven product development: Provoking an understanding. Requirements Engineering Journal 7(3):139–151

Carlshamre P (2002b) A usability perspective on requirements engineering – from methodology to product development. PhD thesis, Linköping University, Sweden

Carlshamre P, Regnell B (2000) Requirements lifecycle management and release planning in market-driven requirements engineering processes. In: Proceedings of the 11th International Workshop on Database and Expert Systems Applications, pp 961–965

Carlshamre P, Sandahl K, Lindvall M, Regnell B, Natt och Dag J (2001) An industrial survey of requirements interdependencies in software product release planning. In: Proceedings of the Fifth IEEE International Symposium on Requirements Engineering (RE 2001), pp 84–91

Carman D, Dolinsky A, Lyu M, Yu J (1995) Software reliability engineering study of a large-scale telelcommunications software system. In: Proceedings of the International Symposium on Software Reliability Engineering, pp 350–359

Carmel E (1999) Global software teams: collaborating across borders and time zones. McGraw-Hill, New york

Carter R, Anton A, Dagnino A, Williams L (2001) Evolving beyond requirements creep: a risk-based evolutionary prototyping model. In: Proceedings of the Fifth IEEE International Symposium onRequirements Engineering, pp 94 –101

Chen J, Reilly R, Lynn G (2005) The impacts of speed-to-market on new product success: the moderating effects of uncertainty. IEEE Transactions on Engineering Management 52(2):199–212

Chrissis M, Konrad M, Shrum S (2004) CMMI: Guidelines for Process Integration and Product Improvement. Pearson Education Inc.

Cilibrasi R, Cruz A, Rooij S, Keijzer M (2012) The complearn suite website. `http://www.complearn.org/index.html`

Cleland-Huang J, Mobasher B (2008) Using data mining and recommender systems to scale up the requirements process. In: Proceedings of the 2nd international workshop on Ultra-large-scale software-intensive systems, pp 3–6

Cleland-Huang J, Chang C, Christensen M (2003) Event-based traceability for managing evolutionary change. Software Engineering, IEEE Transactions on 29(9):796 – 810

Cleland-Huang J, Settimi R, Duan C, Zou X (2005) Utilizing supporting evidence to improve dynamic requirements traceability. In: Proceedings of the 13th IEEE International Conference on Requirements Engineer (RE 2005), pp 135–144

Cleland-Huang J, Czauderna A, Gibiec M, Emenecker J (2010) A machine learning approach for tracing regulatory codes to product specific requirements. In: 2010 ACM/IEEE 32nd International Conference on Software Engineering, vol 1, pp 155 –164

Cleland-Huang J, Gotel O, Zisman A (2012) Software and Systems Traceability. Springer

Clements P, Northrop L (2002) Software Product Lines: Practices and Patterns. Addison-Wesley

Collins G (1960) A method for overlapping and erasure of lists. Commun ACM 3(12):655–657

Cooper W (1968) Expected search length: A single measure of retrieval effectiveness based on the weak ordering action of retrieval systems. J Am Soc Inf Sci 19(1):30–41

Cuddeback D, Dekhtyar A, Hayes J (2010) Automated requirements traceability: The study of human analysts. In: 18th IEEE International Requirements Engineering Conference (RE'10), pp 231 –240

Cunningham W (2012) Manifesto for agile software development. `http://agilemanifesto.org/`

Curtis B, Krasner H, Iscoe N (1988) A field study of the software design process for large systems. Communications of the ACM 31(11):1268–1287

Curtis W, Krasner H, Shen V, Iscoe N (1987) On building software process models under the lamppost. In: Proceedings of the 9th international conference on Software Engineering (ICSE 1987), pp 96–103

Cusumano M (2008) The changing software business: moving from products to services. Computer 41(1):20 – 7

Damian D (2007) Stakeholders in global requirements engineering: Lessons learned from practice. IEEE Software 24(2):21–27

Damian D, Zowhgi D (2003) Requirements engineering challenges in multi-site software development organizations. Requirements Engineering Journal 8(3):149–160

Davison R, Martinsons M, Ned K (2004) Principles of canonical action research. Information Systems Journal 14(1):65–86

De Lucia A, Fasano F, Fausto F, Oliveto R, Tortora G (2007) Recovering traceability links in software artifact management systems using information retrieval methods. ACM Trans Softw Eng Methodol 16(4)

DeBaud J, Schmid K (1998) Identifying and evolving the scope of software product lines. In: In Proceedings of the European Reuse Workshop 1998 (ERWŠ98), pp 69–72

DeBaud J, Schmid K (1999) A systematic approach to derive the scope of software product lines. In: Proceedings of the 21st International Conference on Software Engineering (ICSE 1999), pp 34–43

DeGregorio G (1999) Enterprise-wide requirements and decision management. In: Proceedings of the 9th International Symposium of the International Council on System Engineering, pp 775–582

DeMarco T (1978) Structured Analysis and System Specification. Yourdon Press

DeMarco T, Lister T (2003) Risk management during requirements. Software, IEEE 20(5):99 – 101

Duan C, Cleland-Huang J (2006) Visualization and analysis in automated trace retrieval. In: Proceedings of the First International Workshop on Requirements Engineering Visualization (REV 2006), pp 54–65

Duan C, Laurent P, Cleland-Huang J, Kwiatkowski C (2009a) Towards automated requirements prioritization and triage. Requirements Engineering Journal 14(2):73–89

Duan C, Laurent P, Cleland-Huang J, Kwiatkowski C (2009b) Towards automated requirements prioritization and triage. Requir Eng 14(2):73–89

Dulac N, Viguier T, Leveson N, Storey M (2002) On the use of visualization in formal requirements specification. Los Alamitos, CA, USA, pp 71 – 80

Easterbrook SM, Singer J, Storey M, Damian D (2007) Guide to Advanced Empirical Software Engineering, Springer, chap Selecting Empirical Methods for Software Engineering Research, pp 285–311

Ebert C (2004) Dealing with nonfunctional requirements in large software systems. Annals of Software Engineering 3(1):367–395

Edwards M, Flanzer M, Terry M, Landa J (1995) Recap: a requirements elicitation, capture and analysis process prototype tool for large complex systems. In: Proceedings of the First IEEE International Conference on Engineering of Complex Computer Systems, 1995. Held jointly with 5th CSESAW, 3rd IEEE RTAW and 20th IFAC/IFIP WRTP, pp 278–281

El Ghazi H, Assar S (2008) A multi view based traceability management method. In: Research Challenges in Information Science, 2008. RCIS 2008. Second International Conference on, pp 393 –400

Evermann J (2008) A cognitive semantics for the association construct. Requirements Engineering Journal 13(3):167–186

Fabbrini F, Fusani M, Gnesi S, Lami G (2001) An automatic quality evaluation for natural language requirements. In: Proceedings of the 7th International Workshop on Requirements Engineering Foundation for Software Quality (REFSQ 2001), pp 4–5

Falessi D, Cantone G, Canfora G (2010) A comprehensive characterization of nlp techniques for identifying equivalent requirements. In: Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ACM, New York, NY, USA, ESEM ’10, pp 1–10

Fantechi A, Gnessi S, Lami G, Maccari A (2003) Applications of linguistic techniques for use case analysis. Requirements Engineering Journal 8(3):161–170

Feather M, Cornford S, Gibbel M (2000) Scalable mechanisms for requirements interaction management. In: Proceedings of the Fourteen International Conference on Requirements Engineering (RE 2000), pp 119–129

Feather M, Cornford S, Kiper J, Menzies T (2006) Experiences using visualization techniques to present requirements, risks to them, and options for risk mitigation. In: Proceedings of the First International Workshop on Requirements Engineering Visualization (REV 2006), pp 80–89

Fenton N, Pfleeger S, Glass R (1994) Science and subscience: A challenge to software engineers. IEEE Software 11(4):86–96

Finkelstein A, Harman M, Mansouri SA, Ren J, Zhang Y (2008) "fairness analysis" in requirements assignments. Barcelona, Catalunya, Spain, pp 115 – 124

Fons F, Fons M, Canto E, Lopez M (2012) Deployment of run-time reconfigurable hardware coprocessors into compute-intensive embedded applications. Journal of Signal Processing Systems 66(2):191 – 221

Fritsch C, Hahn R (2004) Product line potential analysis. In: SPLC, pp 228–237

Gandhi R, Lee SW (2007) Visual analytics for requirements-driven risk assessment. 445 Hoes Lane - P.O.Box 1331, Piscataway, NJ 08855-1331, United States

Garg P (1989) On supporting large-scale decentralized software engineering processes. In: Proceedings of the 28th IEEE Conference on Decision and Control, pp 1314–1317

Gervasi V (1999) Environment support for requirements writing and analysis. PhD thesis, University of Pisa

Gervasi V, Nuseibeh B (2000) Lightweight validation of natural language requirements: A case study. In: Proceedings of the 4th International Conference on Requirements Engineering, Society Press, pp 113–133

Glass R (1994) The software research crisis. IEEE Software 11(6):42–47

Goldin L, Berry D (1997) Abstfinder, a prototype natural language text abstraction finder for use in requirements elicitation. Automated Software Engineering pp 375–412

Goldstine H, von Neuman J (1948) Planning and coding of problems for an electronic computing instrument. Tech. rep., The Institute of Advanced Study Princeton, New Jersey

Gonzales-Baixauli B, Prado-Leite J, Mylopoulos J (2004) Visual variability analysis for goal models. Los Alamitos, CA, USA, pp 198 – 207

Gonzalez R, Woods R (2006) Digital Image Processing (3rd Edition). Prentice-Hall, Inc., Upper Saddle River, NJ, USA

Gorschek T, Wohlin C (2006) Requirements abstraction model. Requirements Engineering Journal 11:79–101

Gotel O, Marchese F, Morris S (2007) On requirements visualization. In: Proceedings of the Second International Workshop on Requirements Engineering Visualization (REV 2007), pp 80–89

Gotel O, Marchese F, Morris S (2008) The potential for synergy between information visualization and software engineering visualization. In: Proceedings of the 12th International Conference Information Visualisation, pp 547–552

Greer D, Ruhe G (2004) Software release planning: an evolutionary and iterative approach. Information and Software Technology 46(4):243–253

Hall T, Beecham S, Rainer A (2002) Requirements problems in twelve software companies: an empirical analysis. IEEE Software 149(5):153 – 160

Hayes Huffman J, Dekhtyar A, Osborne J (2003) Improving requirements tracing via information retrieval. In: Proceedings of the 11th IEEE International Conference on Requirements Engineering, IEEE Computer Society, Washington, DC, USA, RE '03, pp 138–147

Hayes Huffman J, Dekhtyar A, Sundaram S, Howard S (2004) Helping analysts trace requirements: an objective look. In: Proceedings of the 12th IEEE International Requirements Engineering Conference (RE'04), pp 249 – 259

Hayes Huffman J, Dekhtyar A, Sundaram S (2006) Advancing candidate link generation for requirements tracing: The study of methods. IEEE Trans Softw Eng 32(1):4–19

Helferich A, Herzwurm A, Schockert G (2005) Mass customization of enterprise applications: Creating customer oriented product portfolios instead of single systems. In: Proceedings of the 3rd Interdisciplinary World Congress on Mass Customization and Personalization

Herald T, Verma D, Lubert C, Cloutier R (2009) An obsolescence management framework for system baseline evolution perspectives through the system life cycle. Syst Eng 12:1–20

Herbsleb J (2007) Global software engineering: The future of socio-technical coordination. Future of Software Engineering 1(1):188–198

Herrmann A, Daneva M (2008) Requirements prioritization based on benefit and cost prediction: An agenda for future research. In: Proceedings of the 16th IEEE International Requirements Engineering Conference, RE'08, Barcelona, Catalunya, Spain, pp 125 – 134

Higgins S, Laat M, Gieles P, Geurts E (2003) Managing requirements for medical it products. IEEE Software 20(1):26-33

Hitchman S (2002) The details of conceptual modeling notations are important - a comparison of relationship normative language. Communication AIS 9(10):188–198

Hornecker E, Buur J (2006) Getting a grip on tangible interaction: A framework on physical space and social interaction. In: In Proceeding of the SIGCHI Conference on Human Factors in Computing Systems, pp 437–446

Höst M, Regnell B, Natt och Dag J, Nedstam J, Nyberg C (2001) Exploring bottlenecks in market-driven requirements management. Journal of Systems and Software 59(3):323–332

Iacovou C, Dexter A (2004) Turning around runaway information technology projects. Engineering Management Review, IEEE 32(4):97 –112

IEEE (1997) IEEE recommended practice for software requirements specifications, 830-1998. `http://standards.ieee.org/findstds/standard/830-1998.html`

IEEE Computer Society (2004) Software Engineering Body of Knowledge (SWEBOK). Angela Burgess, EUA, URL `http://www.swebok.org/`

Isazadeh A, Lamb D, Shepard T (1999) Behavioural views for software requirements engineering. Requirements Engineering 4(1):19 – 37

Jacobs S, Jarke M, Pohl K (1994) Report on the first international ieee symposium on requirements engineering. Automated Software Engineering 1(1):129–132

Jansen S, Brinkkemper S, Souer J, Luinenburg L (2012) Shades of gray: Opening up a software producing organization with the open software enterprise model. Journal of Systems and Software 85(7):1495 – 1510

Jantunen S, Lehtola L, Gause D, Dumdum U, Barnes R (2011) The challenge of release planning. In: Proceedings of the Fifth International Workshop on Software Product Management (IWSPM'2011), pp 36 –45

Järvelin K, Kekäläinen J (2002) Cumulated gain-based evaluation of ir techniques. ACM Trans Inf Syst 20(4):422–446

John I, Eisenbarth M (2009) A decade of scoping: a survey. In: Proceedings of the 13th International Software Product Line Conference, Carnegie Mellon University, Pittsburgh, PA, USA, SPLC '09, pp 31–40

Jones J, Harrold M, Stasko J (2000) Visualization of test information to assist fault localization. In: Proceedings of the 24th International Conference on Software Engineering (ICSE 2002), pp 198–205

Jørgensen M, Molkken-Ostvold K (2004) Reasons for software effort estimation error: Impact of respondent role, information collection approach, and data analysis method. IEEE Transactions on Software Engineering 30(12):993 – 1007

Jørgensen M, Halkjelsvik T, Kitchenham B (2012) How does project size affect cost estimation error? statistical artifacts and methodological challenges. International Journal of Project Management

Kamsties E, Berry D, Paech B (2001) Detecting ambiguities in requirements documents using inspections. In: Proceedings of the First Workshop on Inspection in Software Engineering (WISE 2001), pp 68–80

Kang K, Donohoe P, Koh E, Lee J, Lee K (2002) Using a marketing and product plan as a key driver for product line asset development. In: Proceedings of the Second International Conference on Software Product Lines, Springer-Verlag, London, UK, SPLC 2, pp 366–382

Karlsson J (1996) Software requirements prioritizing. In: Proceedings of the 2nd International Conference on Requirements Engineering (ICRE 96), p 110

Karlsson J (1998) A systematic approach for prioritizing software requirements. doctorial dissertation,. PhD thesis, Linköping University, Sweden

Karlsson J, Ryan K (1997) A cost-value approach for prioritizing requirements. IEEE Software 14(5):67–74

Karlsson J, Wohlin C, Regnell B (1997) An evaluation of methods for prioritizing software requirements. Information and Software Technology 39(14-15):939–947

Karlsson L, Åsa G Dahlstedt, Natt Och Dag J, Regnell B, Persson A (2002) Challenges in market-driven requirements engineering - an industrial interview study. In: Proceedings of the Eighth International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ 2002)

Kaushik N, Tahvildari L, Moore M (2011) Reconstructing traceability between bugs and test cases: An experimental study. In: Reverse Engineering (WCRE), 2011 18th Working Conference on, pp 411 –414

Kekäläinen J, Järvelin K (2002) Using graded relevance assessments in ir evaluation. Journal of American Society of Information Science Technology 53(13):1120–1129

Kishi T, Noda N, Katayama T (2002) A method for product line scoping based on decision-making framework. In: Proceeding Second International Software Product Lines Conference (SPLC 2002), pp 53–65

Kitchenham B, Pickard L, Pfleeger SL (1995) Case studies for method and tool evaluation. IEEE Software 12(4):52–62

Kitchenham B, Budgen D, Brereton P, Turner M, Charters S, Linkman S (2007) Large-scale software engineering questions - expert opinion or empirical evidence? IET Software 1(5):161–171

Knight C, Munro M (2000) Virtual but visible software. In: Proceedings of the IEEE International Conference on Information Visualization, pp 198–205

Konrad S, Gall M (2008) Requirements engineering in the development of large-scale systems. In: Proceedings of the 16th International Requirements Engineering Conference (RE 2008), pp 217–222

Konrad S, Goldsby H, Lopez K, Cheng B (2006) Visualizing requirements in uml models. In: Proceedings of the First International Workshop on Requirements Engineering Visualization (REV 2006), pp 1–10

Koschke R (2003) Software visualization in software maintenance, reverse engineering, and re-engineering: a research survey. Journal of Software Maintenance and Evolution Research and Practice 15(2):87–109

Kotonya G, Sommerville I (1998) Requirements Engineering. John Wiley and Sons

Koziolek A (2012) Research preview: Prioritizing quality requirements based on software architecture evaluation feedback. In: Proceedings 18th International Working Conference on Requirements Engineering: Foundation for Software Quality. REFSQ 2012, Berlin, Germany, pp 52 – 8

Kulk G, Verhoef C (2008) Quantifying requirements volatility effects. Sci Comput Program 72(3):136–175

Larkin J, Simon H (1987) Why a diagram is (sometimes) worth ten thousand words. Cognitive Science 11(1):65–100

Laurent P, Cleland-Huang J, D C (2007) Towards automated requirements triage. In: Proceedings of the 15th IEEE International Requirements Engineering Conference, RE ’07., pp 131 –140

Lee M, Reilly R, Butavicius M (2003) An empirical evaluation of chernoff faces, star glyphs and spatial visualization for binary data. In: Proceedings of the Australian Symposium on Information Visualization, pp 1–10

Leffingwell D, Widrig D (2003) Managing Software Requirements: A Unified Approach. Addison-Wesley

Lethbridge T, Sim S, Singer J (2005) Studying software engineers: Data collection techniques for software field studies. Empirical Software Engineering Journal 10(3):311–341

Lethola L, Kauppinen M (2004) Empirical evaluation of two requirements prioritization methods in product development projects. In: Proceedings of the 11th European Conference EuroSPI, pp 161–170

Liddy E (2003) Natural Language Processing. Encyclopedia of Library and Information Science, 2nd Ed., NY. Marcel Decker, Inc

Lin J, Lin CC, Cleland-Huang J, Settimi R, Amaya J, Bedford G, Berenbach B, Ben Khadra O, Duan C, Zou X (2006) Poirot: A distributed tool supporting enterprise-wide automated traceability. In: Proceedings of the 14th IEEE International Requirements Engineering Conference, IEEE Computer Society, Washington, DC, USA, RE ’06, pp 356–357

Linden F, van der Schmid K, Rommes E (2007) Software Product Lines in Action The Best Industrial Practice in Product Line Engineering. SpringerVerlag

Linger R, Pleszkoch M, Burns L, Hevner A, Walton G (2007) Next-generation software engineering: Function extraction for computation of software behavior. In: Proceedings of the 40th Annual Hawaii International Conference on System Sciences (HICSS 2007), pp 9–17

Loesch F, Ploederoeder E (2007) Restructuring variability in software product lines using concept analysis of product configurations. In: proceedings of the 11th European Conference on Software Maintenance and Reengineering, CSMR ’07, pp 159 –170

Lubars M, Potts C, Richter C (1993) A review of the state of the practice in requirements modeling. In: Requirements Engineering, 1993., Proceedings of IEEE International Symposium on, pp 2 –14

Maccari A (1999) The challenges of requirements engineering in mobile telephones industry. Database and Expert Systems Applications, International Workshop on 0:336–345

MacDonell S (2005) Visualization and analysis of software engineering data using self-organizing maps. In: Proceeding of the International Symposium on Empirical Software Engineering (ISESE 2005), pp 115–124

Macias B, Pulman S (1995) A method for controlling the production of specifications in natural language. The Computer Journal 48(4):310–318

Magazinovic A, Pernstål J (2008) Any other cost estimation inhibitors? In: Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, pp 233–242

Manning C, Raghavan P, Schtze H (2008) Introduction to Information Retrieval. Cambridge University Press, New York, NY, USA

Mannion M, Lewis O, Kaindl H, Montroni G, Wheadon J (2000) Representing requirements on generic software in an application family model. In: Proceedings of the 6th International Conerence on Software Reuse: Advances in Software Reusability, Springer-Verlag, London, UK, pp 153–169

Masri K, Parker D, Gemino A (2008) Using iconic graphics in entity-relationship diagrams: The impact on understanding. Journal of Database Management 19(3):22–41

McCarthy J (1960) Recursive functions of symbolic expressions and their computation by machine, part i. Commun ACM 3(4):184–195

McCracken D, Jackson M (1981) A minority dissenting opinion. In: W.W. Cotterman, et al. (Eds.). Systems Analysis and Design - A Foundation for the 1980s., pp 551–553

McPhee C, Eberlein A (2002) Requirements engineering for time-to-market projects. In: Proceedings Ninth Annual IEEE International Conference and Workshop on the Engineering of Computer-Based Systems, pp 17–24

Merola L (2006) The cots software obsolescence threat. In: Commercial-off-the-Shelf (COTS)-Based Software Systems, 2006. Fifth International Conference on, p 7 pp.

Minsky M (1963) A lisp garbage collector algorithm using serial secondary storage. Tech. rep., Cambridge, MA, USA

Moody D (2009) The “physics” of notations: Towards a scientific basis for constructing visual notations in software engineering. IEEE Transactions on Software Engineering 35(6):756–779

Moody D, Heymans P, Matulevicius R (2010) Visual syntax does matter: improving the cognitive effectiveness of the i* visual notation. Requirements Engineering 15:141–175

Moody DL, Heymans P, Matulevicius R (2009) Improving the effectiveness of visual representations in requirements engineering: An evaluation of i* visual syntax. In: Proceedings of the 17th IEEE International Conference on Requirements Engineering, Atlanta, GA, United states, pp 171 – 180

Murphy D, Rooney D (2006) Investing in agile: Aligning agile initiatives with enterprise goals. Cutter IT Journal 19(2):6 –13

Myaeng SH, Korfhage RR (1990) Integration of user profiles: models and experiments in information retrieval. Information Processing & Management 26(6):719 – 738

Natt och Dag J (2010) The reqsimile tool website. `http://reqsimile.sourceforge.net/`

Natt och Dag J, Gervasi V, Brinkkemper S, Regnell B (2004) Speeding up requirements management in a product software company: Linking customer wishes to product requirements through linguistic engineering. In: Proceedings of the 12th International Requirements Engineering Conference (RE 2004), pp 283–294

Natt och Dag J, Gervasi V, Brinkkemper S, Regnell B (2005) A linguistic engineering approach to large-scale requirements management. IEEE Software 22(1):32–39

Natt och Dag J, Thelin T, Regnell B (2006) An experiment on linguistic tool support for consolidation of requirements from multiple sources in market-driven product development. Empirical Software Engineering Journal 11(2):303–329

Naur P, Randell B (1968) Software engineering: Report of a conference sponsored by the nato science committee. Tech. rep., NATO Scientific Affairs Division

Neill C, Laplante P (2003) Requirements engineering: the state of the practice. IEEE Software 20(6):40-45

Ngo-The A, Ruhe G (2005a) Decision support in requirements engineering. In: Aurum A, Wohlin C (eds) Engineering and Managing Software Requirements, Springer Berlin Heidelberg, pp 267–286

Ngo-The A, Ruhe G (2005b) Engineering and Managing Software Requirements, Springer, chap Decision Support in Requirements Engineering, pp 267–286

Nordbotten J, Crosby M (2001) The effect of graphic style on data model interpretation. Information Systems Journal 9(2):139–155

Northrop L, Felier P, Habriel RP, Boodenough J, Linger R, Klein M, Schmidt D, Sullivan K, Wallnau K (2006) Ultra-Large-Scale Systems: The Software Challenge of the Future. Software Engineering Institute

Odersky M, Spoon L, Venners B (2008) Programming in Scala: A Comprehensive Step-by-step Guide, 1st edn. Artima Inc

Ogawa M, Bird K, Deyanbu C, Gourley A (2007) A visualization social interaction in open source software project. In: Proceedings of the 6th International Asia-Pacific Symposium on Visualization (APVIS 2007), pp 25–32

Osawa K, Ohnishi A (2007) Similarity map for visualizing classified scenarios. In: Proceedings of the Second International Workshop on Requirements Engineering Visualization (REV 2007), pp 80–89

Ozakaya O (2006) Representing requirements relationships. In: Proceedings of the First International Workshop on Requirements Engineering Visualization (REV 2006), pp 75–84

Park S, Kim SD (2005) A systematic method for scoping core assets in product line engineering. In: Proceedings of the 12th Asia-Pacific Software Engineering Conference, APSEC '05., p 8 pp.

Park S, Nang J (1998) Requirements management in large software system development. In: Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, pp 2680–2685

Patton J (2003) Unfixing the fixed scope project: Using agile methodologies to create flexibility in project scope. Agile Development Conference/Australasian Database Conference 0:146

Perini A, Ricca F, Susi A, Bazzanella C (2007) An empirical study to compare the accuracy of ahp and cbranking techniques for requirements prioritization. New Delhi, India, pp 23 – 34

Pfleeger S (2001) Software Engineering – Theory and practice. Prentice–Hall

Pfleeger S, Kitchenham B (2001) Principles of survey research: part 1: turning lemons into lemonade. SIGSOFT Softw Eng Notes 26(6):16–18

Pichler M, Humetshofer H (2006) Business process-based requirements modeling and management. In: Proceedings of the First International Workshop on Requirements Engineering Visualization (REV 2006), pp 20–29

Pohl K, Böckle G, Linden F, Niehaus E, Böckle G (2005a) Product management. In: Software Product Line Engineering, Springer Berlin Heidelberg, pp 163–192

Pohl K, Bockle G, van der Linden F (2005b) Software Product Line Engineering: Foundations, Principles and Techniques. SpringerVerlag

Pollack S (1968) Measures for the comparison of information retrieval systems. Am Doc 19(4):387–397

Potts C (1993) Software engineering research revisited. IEEE Software 10(5):18–28

Potts C (1995) Invented requirements and imagined customers: requirements engineering for off-the-shelf software. In: Proceedings of the Second IEEE International Symposium on Requirements Engineering (RE 95), pp 128–130

Punter T, Ciolkowski M, Freimut B, John I (2003) Conducting on-line surveys in software engineering. In: Empirical Software Engineering, 2003. ISESE 2003. Proceedings. 2003 International Symposium on, pp 80 – 88

Purchase H, Carrington D, Allder J (2002) Empirical evaluation of aesthetics-based graph layout. Empirical Software Engineering Journal 7(3):233–255

R E Albright TK (2003) Roadmapping in the corporation. Research-Technology Management 46(1):31–40

Racheva Z, Daneva M, Sikkel K, Wieringa R, Herrmann A (2010) Do we know enough about requirements prioritization in agile projects: Insights from a case study. In: Proceedings of the 18th International IEEE Requirements Engineering Conference, IEEE Computer Society Press, Los Alamitos, pp 147–156

Ramesh B, Jarke M (2001) Toward reference models for requirements traceability. Software Engineering, IEEE Transactions on 27(1):58 –93

Ramesh B, Cao L, Baskerville R (2010) Agile requirements engineering practices and challenges: an empirical study. Information Systems Journal 20(5):449–480

Raol JR (2009) Multi-Sensor Data Fusion with MATLAB: Theory and Practice. Taylor and Francis, Inc.

Rayson P, Emmet L, Garside R, Sawyer P (2000) The revere project: Experiments with the application of probabilistic nlp to systems engineering. In: Proceedings of the 5th International Conference on Applications of Natural Language to Information Systems, pp 288–300

Rea L, Parker R (2005) Designing and conducting survey research: a comprehensive guide. Jossey-Bass

Regnell B, Brinkkemper S (2005) Engineering and Managing Software Requirements, Springer, chap Market–Driven Requirements Engineering for Software Products, pp 287–308

Regnell B, Kuchcinski K (2011) Exploring software product management decision problems with constraint solving - opportunities for prioritization and release planning. In: Proceedings of the Fifth International Workshop on Software Product Management (IWSPM'2011), pp 47 –56

Regnell B, Beremark P, Eklundh O (1998) A market–driven requirements engineering process – results from an industrial process improvement programme. Requirements Engineering Journal 3(2):121–129

Regnell B, Höst M, Natt och Dag J, Hjelm A (2001) Case study on distributed prioritization in market-driven requirements engineering for packaged software. Requirements Engineering Journal 6(1):51–62

Regnell B, Ljungquist B, Thelin T, Karlsson L (2004) Investigation of requirements selection quality in market-driven software processes using an open source discrete event simulation framework. In: Proceedings of the 5th International Workshop on Software Process Simulation and Modeling, pp 89–93

Regnell B, Olsson HO, Mossberg S (2006) Assessing requirements compliance scenarios in system platform subcontracting. In: Proceedings of the 7th International Conference on Product Focused Software Process Improvement, pp 362–376

Regnell B, Berntsson Svensson R, Olsson T (2008) Supporting roadmapping of quality requirements. IEEE Software 25(2):42–47

Riebisch M, Streitferdt D, Philippow I (2001) Feature scoping for product lines. In: in Proceeding of PLEES'03, International Workshop on Product Line Engineering The Early Steps: Planning Modeling and Managing

Robertson S, Robertson J (1999) Mastering the requirements process. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA

Robson C (2002) Real World Research. Blackwell Publishing

Rocchio J (1966) Document retrieval systems: optimization and evaluation. PhD thesis, Harvard University, USA

Royce W (1970) Managing the development of large software systems: concepts and techniques. In: Proceedings of the IEEE WESTCON Conference, pp 328–338

Ruel H, Bondarouk T, Smink S (2010) The waterfall approach and requirement uncertainty: An in-depth case study of an enterprise systems implementation at a major airline company. Int J Technol Proj Manage (USA) 1(2):43 – 60

Ruhe G (2003) Software engineering decision support - a new paradigm for learning software. Lecture Notes in Computer Science 2640(1):104–113

Ruhe G (2009) Product Release Planning: Methods, Tools and Applications. Auerbach Publications

Ruhe G, Greer D (2003) Quantitative studies in software release planning under risk and resource constraints. In: Proceedings of the International Symposium on Empirical Software Engineering (ISESE 2003), pp 262–271

Ruhe G, Saliu M (2005) The art and science of software release planning. IEEE Software 22(6):47–53

Runeson P, Höst M (2009) Guidelines for conducting and reporting case study research in software engineering. Empirical Software Engineering Journal 14(2):131–164

Runeson P, Host M, Rainer A, Regnell B (2012) Case Study Research in Software Engineering: Guidelines and Examples. Wiley

Rupp C (2000) Linguistic methods of requirements engineering (nlp). In: Proceedings of the EuroSPI 2000, pp 68–80

Russ J (1999) The image processing handbook (3rd ed.). CRC Press, Inc., Boca Raton, FL, USA

Ryan K (1993) The role of natural language in requirements engineering. In: Proceedings of the IEEE International Symposium on Requirements Engineering, San Diego California, IEEE Computer Society Press, pp 240–242

Saaty T (1980) The Analytic Hierarchy Process, Planning, Piority Setting, Resource Allocation. McGraw-Hill, New york

Savolainen J, Oliver I, Mannion M, Z H (2005) Transitioning from product line requirements to product line architecture. In: proceedings of the 29th Annual International Conference on Computer Software and Applications, COMPSAC 2005., vol 1, pp 186 – 195 Vol. 2

Savolainen J, Kauppinen M, Mannisto T (2007) Identifying key requirements for a new product line. In: Proceedings of the 14th Asia-Pacific Software Engineering Conference (APSEC 2007), pp 478–485

Sawyer P (2000) Packaged software: Challenges for re. In: Proceedings of the Sixth International Workshop on Requirements Engineering: Foundations of Software Quality (REFSQ 2000), pp 137–142

Sawyer P, Rayson P, Garside R (2002) Revere: Support for requirements synthesis from documents. Information Systems Frontiers 4(3):343–353

Sawyer P, Rayson P, Cosh K (2005) Shallow knowledge as an aid to deep understanding in early phase requirements engineering. IEEE Transactions on Software Engineering 31(11):969–981

Schalken J, Brinkkemper S, Vliet H (2001) Assessing the effects of facilitated workshops in requirements engineering. In: Proceedings of the 8th Conference on Evaluation and Assessment in Software Engineering (EASE 2004), Press, pp 135–144

Schmid K (2002) A comprehensive product line scoping approach and its validation. In: Proceedings of the 24th International Conference on Software Engineering (ICSE 2002), pp 593–603

Schneider K, Stapel K, Knauss E (2008) Beyond documents: Visualizing informal communication. Barcelona, Spain

Seaman C (1999) Qualitative methods in empirical studies of software engineering. IEEE Transactions on Software Engineering 25(4):557–572

Sellier D, Mannion M (2006) Visualizing product line requirements selection decision inter-dependencies. In: Proceedings of the Second International Workshop on Requirements Engineering Visualization (REV 2007), pp 20–29

Sen A, Jain S (2007) A visualization technique for agent based goal refinement to elicit soft goals in goal oriented requirements engineering. 445 Hoes Lane - P.O.Box 1331, Piscataway, NJ 08855-1331, United States

Shull F, Carver J, Vegas S, Juristo N (2008) The role of replications in empirical software engineering. Empirical Software Engineering Journal 13(2):211–218

Singer J, Sim SE, Lethbridge TC (2007) Guide to Advanced Empirical Software Engineering, Springer, chap Software Engineering Data Collection for Field Studies, pp 9–34

Software Engineering Institute (2011) Capability maturity model integration (cmmi), version 1.3. http://www.sei.cmu.edu/cmmi/solutions/info-center.cfm

Sommerville I (2007) Software Engineering. Addison-Wesley

Spink A, Greisdorf H (2001) Regions and levels: measuring and mapping users' relevance judgments. J Am Soc Inf Sci Technol 52(2):161–173

Stathaki T (2008) Image Fusion: Algorithms and Applications. Academic Press

Stephen J, Page J, Myers J, Brown A, Watson D, Magee I (2011) System error fixing the flaws in government it. Tech. Rep. 6480524, Institute for Government, London

Strigini L (1996) limiting the dangers of intuitive decision making. Software, IEEE 13(1):101 –103

Sultanov H, Huffman Hayes J (2010) Application of swarm techniques to requirements engineering: Requirements tracing. In: Proceedings of the 2010 18th IEEE International Requirements Engineering Conference, IEEE Computer Society, Washington, DC, USA, RE '10, pp 211–220

Supakkul S, Chung L (2010) Visualizing non-functional requirements patterns. Sydney, NSW, Australia, pp 25 – 34

Sutcliffe A, Thew S, Jarvis P (2011) Experience with user-centred requirements engineering. Requirements Engineering 16(4):267 – 280

Svahnberg M (2003) Supporting software architecture evolution - architecture selection and variability. PhD thesis, Blekinge Institute of Technology

Taborda L (2004) Generalized release planning for product line architectures. In: Nord R (ed) Software Product Lines, Lecture Notes in Computer Science, vol 3154, Springer Berlin / Heidelberg, pp 153–155

Taylor C, Miransky A, Madhavji N (2011) Request-implementation ratio as an indicator for requirements prioritisation imbalance. In: Proceedings of the 5th International Workshop on Software Product Management (IWSPM 2011), Trento, Italy, pp 3 – 6

Teyseyre A (2002) A 3d visualization approach to validate requirements. In: Proceedings of the Congreso Argentino de Ciencias dela Computacion, pp 1–10

Tory M, Moller T (2004) Rethinking visualization: A high-level taxonomy. In: Proceedings of IEEE Symposium on Information Visualization, (INFOVIS 2004)., pp 151–158

Trautmann N, Philipp B (2009) Resource-allocation capabilities of commercial project management software: An experimental analysis. In: Proceedings of the 2009 International Conference on Computers and Industrial Engineering (CIE 2009), pp 1143–1148

Travassos G, dos Santos P, Neto P, Biolchini J (2008) An environment to support large scale experimentation in software engineering. In: Proceedings of the 13th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2008), pp 193–202

Tufte E (1990) Envisioning Information. Graphics Press LLC

Ugai T, Hayashi S, Saeki M (2010) Visualizing stakeholder concerns with anchored map. Sydney, NSW, Australia, pp 20 – 24

UML (2010) The unified modeling language webpage. `http://www.uml.org`

Vähäniitty J, Lassenius C, Rautiainen K (2002) An approach to product roadmapping in small software product business. In: Proceedings of the 7th European Conference Software Quality, pp 56–65

van de Weerd I, Brinkkemper S, Nieuwenhuis R, Versendaal J, Bijlsma L (2006a) On the creation of a reference framework for software product management. In: Proceedings of the First International Workshop on Software Product Management (IWSPM 2006), pp 3–12

van de Weerd I, Brinkkemper S, Nieuwenhuis R, Versendaal J, Bijlsma L (2006b) Towards a reference framework for software product management. In: Proceedings of the 14th IEEE International Requirements Engineering Conference (RE 2006), pp 319–322

van der Hoek A, Hall R, Heimbigner D, Wolf A (1997) Software release management. In: Proceedings of the Sixth European Software Engineering Conference (ESEC/FSE 97), pp 159–175

van Gurp J, Bosch J, Svahnberg M (2001) On the notion of variability in software product lines. In: Proceedings of the Working IEEE/IFIP Conference on Software Architecture, pp 45–55

Vasile S, Bourque P, Abran A (2006) Visualization - a key concept for multidimensional performance modeling in software engineering management. In: Proceeding of the IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR 2006), pp 334–339

Veerappa V, Letier E (2011) Clustering stakeholders for requirements decision making. Berlin, Germany, pp 202 – 8

Wiegers K (2003) Software Requirements: Practical Techniques for Gathering and Managing Requirements Throughout the Product Development Cycle. Addison-Wesley

Wieringa R, Heerkens H (2007) Designing requirements engineering research. In: Proceedings of the 5th International Workshop on Comparative Evaluation in Requirements Engineering, pp 36–48

Winkler S, Pilgrim J (2010) A survey of traceability in requirements engineering and model-driven development. Softw Syst Model 9(4):529–565

Withey J (1996) Investment Analysis of Software Assets for Product Lines. Tech. Rep. CMU/SEI-96-TR-010, Software Engineering Institute, Carnegie Mellon University

Wohlin C, Aurum A (2005) What is important when deciding to include a software requirements in a project or release? In: Proceedings of the International Symposium on Empirical Software Engineering (ISESE 2005), pp 246–255

Wohlin C, Min X, Magnus A (1995) Reducing time to market through optimization with respect to soft factor. In: Proceedings of the 1995 IEEE Annual International Engineering Management Conference, pp 116–121

Wohlin C, Runeson P, Höst M, Ohlsson M, Regnell B, Wesslen A (2000) Experimentation in Software Engineering An Introduction. Kluwer Academic Publishers

Yin R (2003) Case Study Research: Design and Methods. Sage Publications

Zenebe A, Norcio A (2007) Visualization of item features, customer preference and associated uncertainty using fuzzy sets. In: Proceedings of the Annual Meeting of the North American Fuzzy Information Processing Society, pp 7–12

Zowghi D, Nurmuliani N (2002) A study of the impact of requirements volatility on software project performance. In: Proceedings of the Asia-Pacific Software Engineering Conference (APSEC'2002), IEEE Computer Society, Los Alamitos, CA, USA, pp 3–11

Paper I

What Happened to Our Features? Visualization and Understanding of Scope Change Dynamics in a Large-Scale Industrial Setting

Krzysztof Wnuk[1], Björn Regnell[1], Lena Karlsson[2]
[1]Dept. of Computer Science, Lund University, Sweden
{krzysztof.wnuk,bjorn.regnell}@cs.lth.se
[2]Det Norske Veritas, Sweden
lena.karlsson@dnv.com

In Proceedings of the
17th International Requirements Engineering Conference (RE09),
September 2009, Atlanta, USA

Abstract

When developing software platforms for product lines, decisions on which features to implement are affected by factors such as changing markets and evolving technologies. Effective scoping thus requires continuous assessment of how changes in the domain impact scoping decisions. Decisions may have to be changed as circumstances change, resulting in a dynamic evolution of the scope of software asset investments. This paper presents an industrial case study in a large-scale setting where a technique called Feature Survival Charts for visualization of scoping change dynamics has been implemented and evaluated in three projects. The evaluation demonstrated that the charts can effectively focus investigations of reasons behind scoping decisions, valuable for future process improvements. A set of scoping measurements is also proposed, analyzed theoretically and evaluated empirically with data from the cases. The conclusions by the case company practitioners are positive, and the solution is integrated with their current requirements engineering measurement process.

1 Introduction

Deciding which requirements to include into the scope of an upcoming project is not a trivial task. Requirements for complex systems may be counted in thousands, and not all may be included in the next development project or next release. This means that it is necessary to select a subset of requirements to implement in the forthcoming project, and hence postpone the implementation of other requirements to a later point in time (Wohlin and Aurum, 2005; Greer and Ruhe, 2004). This selection process is often called scoping and is considered as a key activity for achieving economic benefits in product line development (Schmid, 2002). While its importance has already been reported in several studies, research has not yet put broad attention to the issues of product line scoping. In particular, following Schmid (2002), we agree that existing work in domain engineering in software product lines focus mainly on the identification aspect of scoping e.g. (Kishi et al, 2002; Savolainen et al, 2007). On the other hand, some researchers have already addressed the issue of understanding underlying reasons for the inclusion of certain requirements in a specific release (Wohlin and Aurum, 2005), while others investigated one of the root causes for changing requirements, namely requirements uncertainty (Ebert and DeMan, 2005).

The problem with many changes in the scoping process for product line projects has recently been identified by one of our industrial partners from the embedded systems domain. This issue has been particularly challenging for the case company, since their current requirements management tool could not provide a sufficient method to visualize and characterize this phenomena. As a remedy, the Feature Survival Chart (FSC) concept was proposed by the authors and acknowledged by the practitioners as a valuable support. This paper extends the contributions of (Wnuk et al, 2008) with (1) findings from industrial application in three projects and (2) scope tracking measurements. The proposed visualization shows the decision process of including or excluding features that are candidates for the next release. Our technique can spot the problem of setting too large a scope compared to available resources as well as increase the understanding of the consequences of setting a limited scope early. By using graphs, we can identify which features and which time frames to analyze in order to find scoping issues related to uncertainties in the estimations that decisions rely on. The charts have also shown to be useful in finding instabilities of the scoping process.

The proposed set of scope tracking measurements complements the proposed visualization technique, and they aim at further increasing the understanding of the rationale and dynamics of scope changes. The measurements are analyzed both theoretically and empirically using data from three large industrial projects that contain hundreds of high-level features related to thousands of system requirements. We also present findings from discussions on the results with practitioners that ranked the usefulness of the proposed measurements and expressed their opinions about their value in scope management.

The paper is structured as follows: Section 2 provides background information about the context of our industrial case study. Section 3 describes the methodology used in this study. Section 4 explains our visualization technique. Section 5 describes the results from applying our technique to three industrial projects. Section 6 defines and evaluates the proposed measurements. Section 7 provide conclusions and discusses their limitations.

2 The case company

Our results are based on empirical data from industrial projects at a large company that is using a product line approach (Pohl et al, 2005). The company has more than 5000 employees and develops embedded systems for a global market. There are several consecutive releases of the platform, a common code base of the product line, where each of them is the basis for one or more products that reuse the platform's functionality and qualities. A major platform release has approximately a two year lead time from start to launch, and is focused on functionality growth and quality enhancements for a product portfolio. Minor platform releases are usually focused on the platform's adaptations to the different products that will be launched with different platform releases. This approach creates an additional requirements flow, which in our case company is handled as a *secondary flow*, and arrives to the platform project usually in the middle of its life cycle. This flow enables flexibility and adaptation possibilities of the platform project, while the *primary flow* is dedicated to address functionality of the highest importance.

There are several groups of specialists associated with various stages of the requirements management process in the case company. For this case, the most essential groups are called *Requirements Teams (RTs)* that elicit and specify high-level requirements for a special technical area, and *Design Teams (DTs)* that design and develop previously defined functionality.

The company uses a stage-gate model with several increments (Cooper, 1990). There are *Milestones (MSs)* and *Tollgates (TGs)* to control the project progress. In particular, there are four milestones for the requirements management and design before the implementation starts: MS1, MS2, MS3, and MS4. For each of these milestones, the project scope is updated and baselined. The milestone criteria are as follows:

MS1: At the beginning of each project, long-term RT's roadmap documents are extracted to formulate a set of features for an upcoming platform project. A *feature* in this case is a concept of grouping requirements that constitute a new functional enhancement to the platform. At this stage the features usually contain a description, its market value and effort estimates. The level of details for the features should be set up in a way that enables judgment of its market value and effort of implementation. Both values are obtained using a cost-value approach (Karlsson and Ryan, 1997). The cost for implementation and the market value of features are the basis for initial scoping inclusion for each technical area. The features are reviewed, prioritized and approved. The initial scope is decided and baselined per RT, guided by a project directive and based on initial resource estimates in the primary receiving DT. The scope is then maintained in a document called Feature List, that is regularly updated each week after a meeting of the *Change Control Board (CCB)*. The role of the CCB is to decide upon adding or removing features according to changes that happen. The history of scope changes is the input data for the visualization technique described in this paper.

MS2: Features are refined to requirements which are specified, reviewed and approved. One feature usually contains ten or more requirements from various areas in the products. The features are assigned to DTs that will take responsibility for designing and implementing the assigned features after MS2. The DTs also allocate an effort estimate per feature.

MS3: The effort estimates are refined and the scope is updated and baselined. DTs refine system requirements and start designing.

MS4: The requirements work and design are finished, and ready to start implementation. The final scope is decided and agreed with the development resources.

According to the company guidelines, most of the scoping work should be done before reaching the second milestone of the process. The secondary flow starts approximately at MS2 and is connected to the start of product projects. Both primary and secondary flows run in parallel under the same MS criteria until they are merged together when the secondary flow reaches its MS4. The requirements are written in domain-specific natural language, and contain many special terms that require contextual knowledge to be understood. In the early phases, requirements contain a high-level customer-oriented description while being refined to detailed implementation requirements at a late stage.

3 Research Methodology

The development of the FSC chart and corresponding scope tracking measures was performed in an interactive manner that involved practitioners from the case company. The persons that participated in the constant evolution and evaluation include one process manager, two requirements managers and one KPI (Key Performance Indicators) manager. This approach involves a set of meetings and discussion points between the researchers and the practitioners that helped to guide the research. As a part of the discussion, the important need to measure the dynamics and the nature of the scope changes emerged. After proposing and theoretically validating the measurements, it was decided to apply them to the real scoping data to empirically confirm the perceived usefulness of the metrics. All ongoing projects in the case company were investigated for possible usage of our technique. Our criteria of interest in analyzing a particular project include (1) the length of analyzed project, (2) the number of features considered in the scope of the project and (3) the possibility to visualize and analyze significant scope changes in the analyzed project. As a result, the three most interesting ones were selected. Furthermore, we have used our technique to define a set of scoping quality measurements that we evaluated by practitioners and validated using empirical data. Finally, we have performed an interview study with platform project requirements managers in order to understand the rationale and implications for scoping decisions.

To gather data for this study, we have implemented an exporter to retrieve the data from the scope parameter of each feature in the Feature List document. This information was later sorted so that each feature is mapped into one row and each value of the scope attribute is mapped to an integer value. After creating graphs, a meeting with practitioners was held in order to present and discuss results as well as address issues for future work. As a result of this meeting, it was decided to introduce and evaluate a set of scope tracking measurements that may give a better insight into the scoping process practices and may help to assess their quality. As one of the measurements, it was decided to include a non-numerical reason for scope exclusion to understand their nature and implications on the stability of the requirements management process. All measurements were calculated on an industrial set of three large platform projects.

4 Feature Survival Charts

In this section, we briefly describe our visualization technique. The *Feature Survival Chart (FSC)*, exampled in Figure 1.1, shows scope changes over time which is illustrated on the X-axis. Each feature is positioned on a specific place on the Y-axis so that the complete lifecycle of a single feature can be followed by looking at the same Y-axis position over time. The various scope changes are visualized using different colors. As a result, each scope change can be viewed as a change of the color. Based on discussions with practitioners we decided to use this coloring scheme: green for features considered as a part of the scope, red for features considered as de-scoped and, if applicable, different shades of green for primary and secondary flows. After sorting the features according to how long they were present in the scope, we get a graph where several simultaneous scope changes can be seen as 'steps' with areas of different colors. The larger the red areas

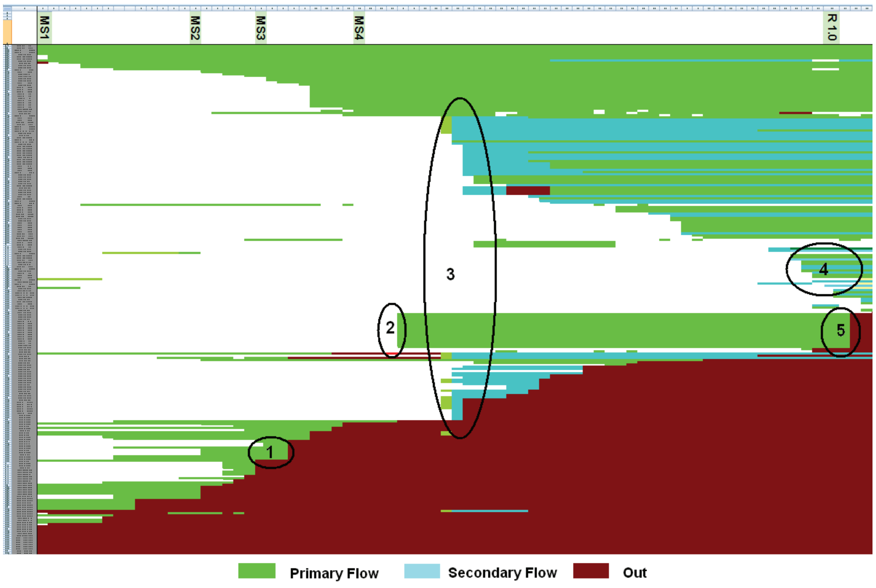


Figure 1.1: Feature Survival Chart for project A.

are, the more features are de-scoped in the particular time of the project. At the top of the graph we can see features that we called 'survivors'. These features represent functionality that was included early while lasting until the end of the scoping process. An FSC is also visualizing overall trends in scoping. In Figure 1.1 we can see that most of scoping activity happened after MS2 in the project. (Rn.m denotes formal releases.) Since most of the de-scoping was done rather late in the project, we can assume that a significant amount of effort might have been spent on features that did not survive. Thanks to the graphs, we can see which decisions have been made when and how large impact on the scope they had. The five areas marked in Figure 1.1 are further discussed in Section 6.3.5. The FSC gives a starting point for investigating why the decisions were made, and enables definition of measurements that indicate quality aspects of the scoping process.

5 Evaluation results

In this section, we present results from evaluating our visualization technique. We present FSCs for three large platform projects in the case com-

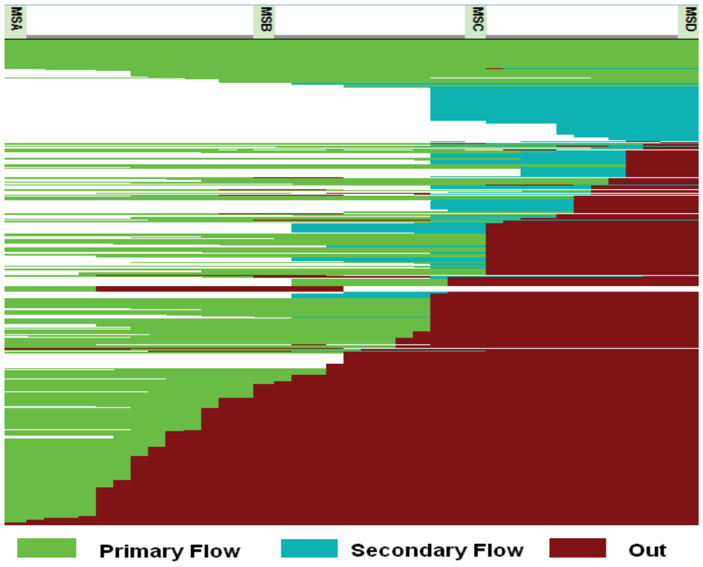


Figure 1.2: FSC for project B.

pany. The data was gathered during autumn 2008 when all three projects were running in parallel and were targeted for product releases in 2008 or 2009. Each project was started at different points in time. At the time when this study was performed one of them had already passed MS4, one had launched the first platform release and the third had passed MS3. In Figures 1.1, 1.2 and 1.3 we present one FSC respectively for three projects denoted A, B and C. Additional information about the projects is presented in Table 1.1.

All analyzed projects have more than 100 features ever considered in the scope. For projects B and C, the significant feature number difference is a result of running these two projects in parallel targeted to be released the same year. The technical areas are similar for all projects. We can assume that the projects affect similar groups of requirements analysts, but differ in size, time of analysis and complexity. Project A was analyzed during a time period of 77 weeks, during which period two releases of the platform were launched. The total number of scope changes in the projects is calculated from MSA and onwards.

Results indicate that we in average experience almost one scope deci-

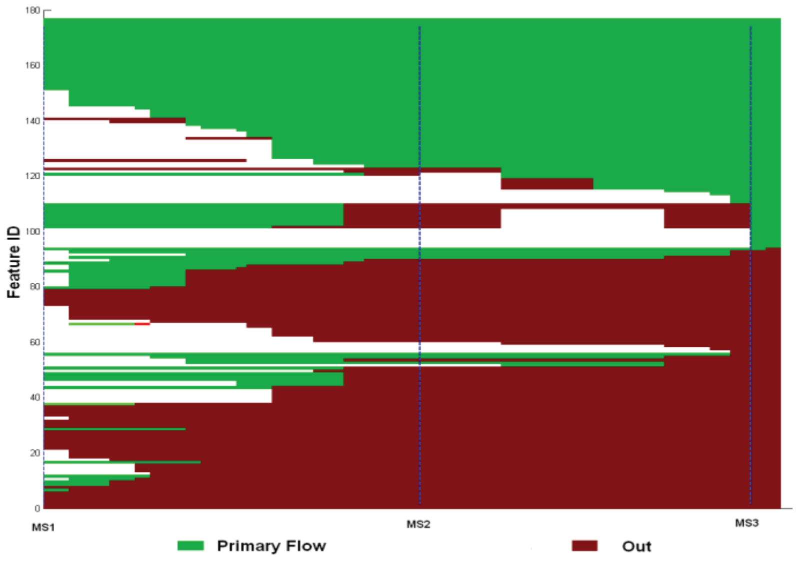


Figure 1.3: FSC for project C.

sion per feature for each project. This fact indicates the need for a better understanding of the scoping process, e.g. by visualizing scope changes. A qualitative analysis of the graphs indicates that for all analyzed projects the dominant trend is de-scoping rather than scope increases. We name this phenomena negative scoping. For all analyzed projects we can observe negative scoping all through the analyzed period.

6 Scope tracking measurements

According to Basili and Rombach (1988), measurement is an effective mechanism for characterizing, evaluating, predicting and providing motivation for the various aspects of software construction processes. The same author states that most aspects of software processes and products are too complicated to be captured by a single metric. Following this thread, we have formulated questions related to external attributes of the scoping process, which in turn is related to internal attributes and a set of five measurements divided into time related measurements and feature related measurements, as described subsequently.

Table 1.1: Characteristics of analyzed projects

Project	Nbr. of features	Nbr. of technical areas	Time Lenght (weeks)	Total number of scope changes
A	223	22	77	237
B	531	23	39	807
C	174	20	20	43

6.1 Definition of measurements

The goal with the measurements is to characterize volatility and velocity of the scoping process, as well as clarity of the reasons behind them. To address this goal, we have defined a set of five scope tracking measurements, which are presented subsequently. Four out of five measurements can be calculated based on the scope attribute value in the feature list document and time stamps for this document, while the last measurement needs a more qualitative approach that requires additional information that complements the graphs.

6.1.1 Time-related scope tracking measurements:

In this category we have defined one measurement:

Number of positive and negative scope changes per time stamp/baseline (M1). We define a positive scope change as an inclusion of a feature into the scope of the project, while a negative change indicates exclusion from the project. We assume that the scope ideally would stabilize in the late phase of the project in order to avoid expensive late changes.

6.1.2 Feature related measurements:

In this category we have defined the following measurements that also can be averaged for the whole platform project:

Time to feature removal (M2) - the time from the feature was introduced until it was permanently removed. The measurement can of course only be calculated for the features that have not survived until the end of the requirements management process. The interpretation of this measurement can be as follows: it is a matter of quality of the requirements management process to remove features that will not be included into the projects due to various reasons as early as possible. This approach saves more resources for the features that will be included into the scope, and increases the efficiency of the scoping process. The pitfall related to this measurement is the uncertainty whether features included into the scope at the end of the requirements management process will not be excluded later due to various reasons. On the other hand, even taking this fact into consideration, we still believe that we successfully can measure M2 and get valuable indications of the final scope crystallization abilities.

Number of state changes per feature (M3) - this measurement is a reflection of the measurement M1. By calculating this measurement for all features and visualizing results in the form of a distribution, we can see the fraction of complex decisions among all decisions. The interpretation of this measurement is that the fewer changes per feature in a project, the more 'stable' the decision process is and less extra effort has to be spent on complex decisions making the project less expensive to manage. As already mentioned, high values for this measurement indicate complex and frequently altered decisions.

Time to birth (M4) - for each feature that has not yet appeared in the scope, we calculate the delay time which is proportional to the number of baselines of the scope document. In our calculations, we took into consideration the fact the feature list document was baselined irregularly, and we based our calculations on the number of days between the baselines. This measurement describes the activity of the flow of new features in time. Here, similarly to M1, we have to decide what is our starting point in the project. Our interpretation assumes that we take MS1 as a starting point. In an ideal situation we expect few features with a long time to birth, since late additions to the scope create turbulence in the project.

Reason for scoping decision (M5) - as the last measurement described in this study, we define reasons for scoping decisions. This measurement will be calculated as a non-numerical value and it can not be automatically derived from our graphs. As already mentioned in M1, inclusion of a new functionality is a different change compared with an exclusion of a functionality. Due to limited access to practitioners, we focused on analyzing removal of functionality. To calculate M5, we mapped each feature to its reason for inclusion, reason for exclusion and existing CCB records.

6.2 Theoretical analysis of measurements

In this section, we present results from a theoretical analysis of the proposed measurements. We have used two approaches: "key stage of formal measurement" (Fenton and Pfleeger, 1996) and the theoretical validation (Kitchenham et al, 1995). By following the key stages of formal measurement, we constructed empirical and mathematical systems and defined a mapping between these two systems. The attributes of an entity can have different measurements associated to them, and each measurement can be connected to different units. Some properties, for example mapping between real world entities to numbers and the fact that different entities can have the same attribute value, are by intuition satisfied for all defined measurements. In Table 1.2 we present defined attributes and relations. We also relate defined measurements with internal and external attributes of the requirements decision process. As we can see in Table 1.2, the defined set of measurements is addressing stability, velocity, volatility and understandability of the scoping process for platform projects. Although four out of five defined measurements are realized as objective numbers, conclusions drawn from them about subjective attributes of requirements management decision process are a matter of interpretation. The subjective interpretation of the results derived by our measurements is a complex task which requires a deep domain knowledge and additional information about the history of the project. We have extended our knowledge by interacting with requirements managers working with platform projects in order to derive values for M5.

6.3 Empirical application of measurements

In this section, we present results from an empirical evaluation of measurements defined in Section 6.1. We have evaluated M1-M4 in three large platform projects described in Section 5, and M5 in one large project. To increase the possibilities of drawing conclusions, we have decided to present time-related measurements as a function of time, while feature-related measurements are presented in the form of distributions for each evaluated project.

6.3.1 Number of positive and negative scope changes per time stamp/baseline (M1).

All three projects turned out to have many scope changes over time. In Figure 1.4 we can see many fluctuations of M1 values both on the positive and negative side rather late in analyzed projects. This result can be explained by a stage-gate model for requirements management projects resulting in high peaks of changes around project milestones. On the other hand, we experience more than four peaks for each project, which is more than the number of milestones in the requirements management process. The distinction of positive and negative changes makes it possible to see in Figure 1.4 that inclusions of new functionality into the project may be correlated with exclusions of some other functionality. The baseline number represents the version of the scope document. The best example is the peak of inclusions for Project A around baseline 38, which immediately resulted in a peak of exclusions. In this example we can also see that the magnitude of the change in both directions is similar.

6.3.2 Time to remove a feature (M2).

For this measurement, we present results in the form of distributions. The distribution presented in Figure 1.5 is showing that many features were

Table 1.2: Results from a theoretical analysis of proposed measurements, by # we mean 'number of'

	Entity	Internal attribute	External attribute	Measure	Domain	Scale	Empirical relation	Mathematical relation
M1	Feature List	Size and direction of scope changes over time.	Stability of the scoping process	# of scope inclusions at the time stamp # of scope exclusions at the time stamp	Feature List	Ratio	negative, positive, bigger smaller, equal to, addition, subtraction, division	<,>,=,+,-, etc.
M2	Feature	The time that was needed to remove the feature from the scope	Velocity of the final scope crystallization process	# days needed to make a final decision about feature exclusion	Feature	Ratio	bigger, smaller, equal to, addition, subtraction, division	<,>,=,+,-, etc.
M3	Feature	Number of scope decisions per feature	Volatility and dynamics of the scope decisions.	# scope changes for non-survivors needed to remove them from the scope.	Feature	Ratio	bigger, smaller, equal to, addition, subtraction, division	<,>,=,+,-, etc.
M4	Feature	Time when a feature was included into the scope of the project	Volatility of the scope decisions.	# days from the beginning of the project until a feature was included	Feature	Ratio	bigger, smaller, equal to, addition, subtraction, division	<,>,=,+,-, etc.
M5	Changes to feature	Rationale for removing features from the scope	Clarity of the reasons for scope decisions	Reasons for scope exclusions	Scope changes	Nominal	equal and different	<>,=

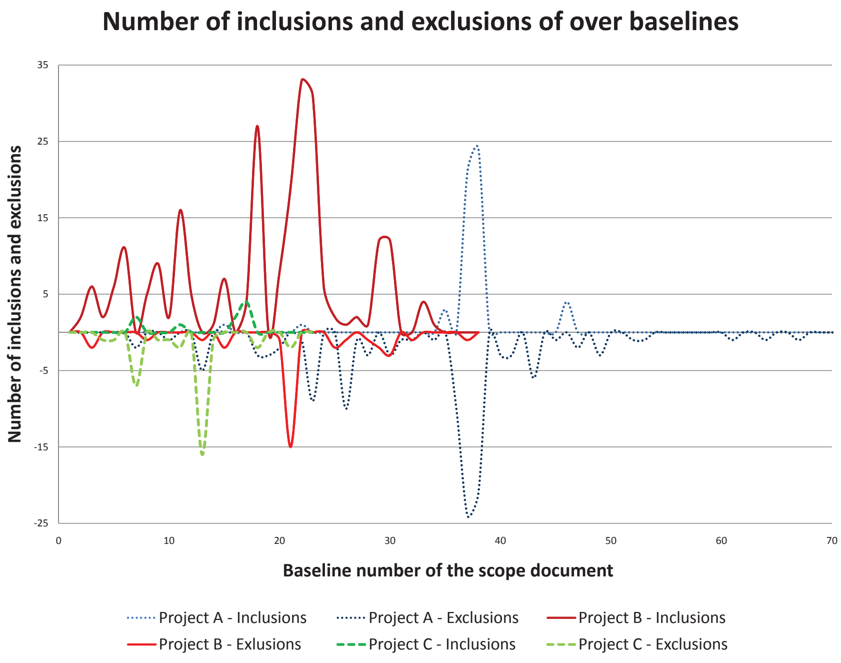


Figure 1.4: Number of positive and negative changes as a function of a baseline number (M1).

removed after a certain number of days in the scope. Our results reveal three different approaches for removing the features from the scope. For Project A we can see an initial scope reduction rather early, then a quite constant number of removed features, and suddenly, after about 300 days from the project start, large scope reductions. For Project B we can see that many features were removed in rather short intervals in time, and also that some significant scope reductions that occurred after 150 days in the project. On the other hand, Project C is behaving more stable in this matter, having only one large peak of removed features around 60 days from the project launch. This type of graph can be useful in assessing how good the process is in crystallizing the final scope of the platform project.

6.3.3 Number of state changes per feature (M3).

For this measurement, we present the results in the form of distributions. As we can see in Figure 1.6, most features required only one decision in the project. This decision usually was an exclusion from the project scope, but in some cases more than one decision per project was needed. This fact indicates that features were shifted between the primary and the sec-

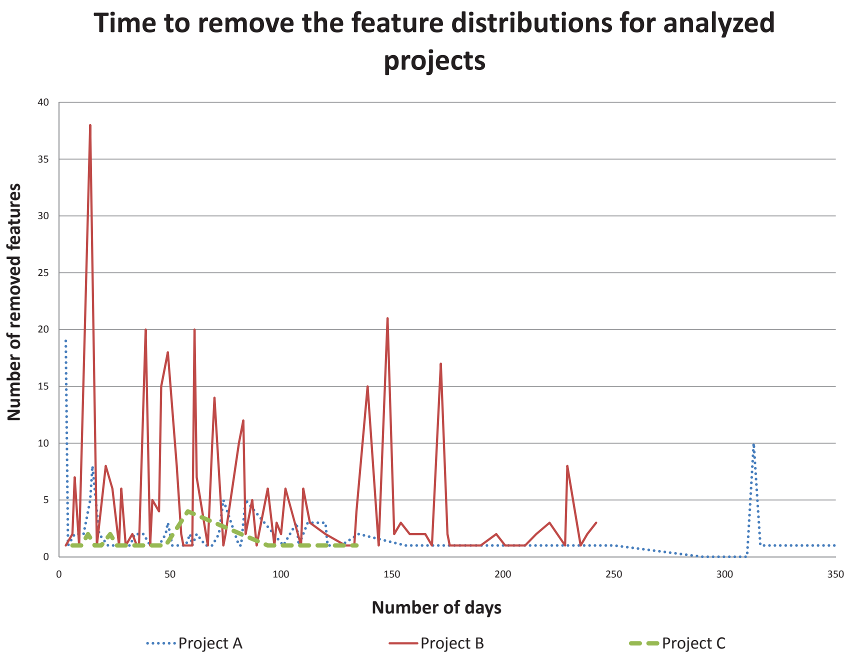


Figure 1.5: Distributions of time to remove the feature (M2).

ondary flow of requirements, or that the management had to reconsider previously made commitments. For a better understanding of more complex decisions, this measurement can be limited to the number of scope changes needed to remove the feature from the scope of the project. This measurement may give valuable insights about the complexity of decision-making. We have calculated a derived measurement, and the results are available online (Wnuk, 2010).

6.3.4 Time to birth (M4).

Empirical application of M4 presented as a distribution over time revealed that some projects have a large peak of new functionality coming into the scope of the project after 100 days from the beginning. In two out of three analyzed cases we experienced large scope extensions at various points in the project timeline. The biggest limitation of this measurement is the fact that the used process allows for a secondary flow of requirements which automatically can create large peaks of births at a certain time. We can notice this fact in Figure 1.7 as a peak of births around day 150 day for Project B, and around day 220 for Project A. Although the mentioned peaks are not necessarily revealing any unplanned behavior, Figure 1.7 reveals that

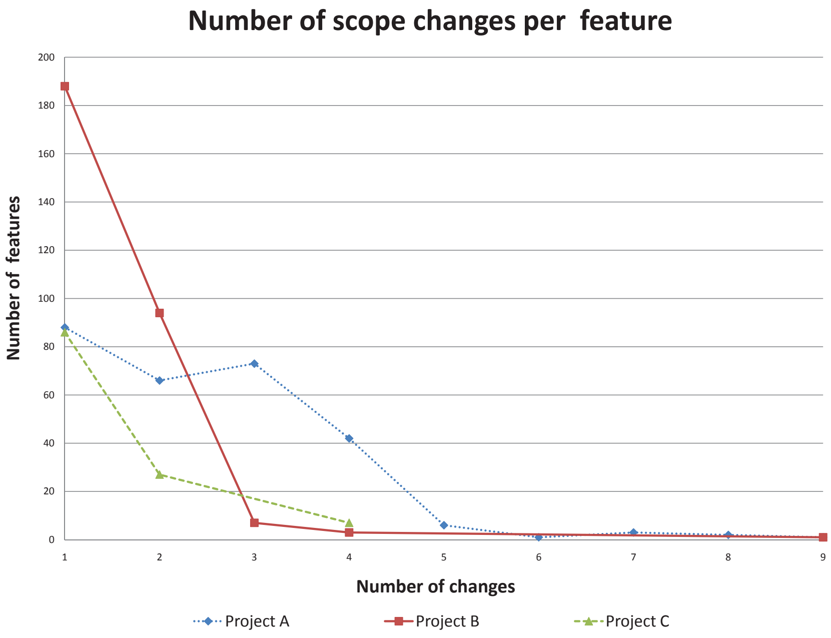


Figure 1.6: Distribution of number of changes per feature (M3).

smaller but still significant scope inclusions appeared for project B both before and after the biggest peak of incoming features.

6.3.5 Reason for scoping decision (M5).

Since M5 is defined as a non-numerical measurement in order to apply it to our industrial data set and gather results, we held a meeting with two requirements managers responsible for managing project scoping information. Each of the requirements managers was responsible for one scoping project. In this paper, we focus on Project A since it was the most interesting in terms of late scope changes. Before the meeting, we prepared five scope-zones which we assumed to be the most interesting to analyze, see Figure 1.1. During the interview, a responsible requirements manager checked the reasons for a particular scope change. The reasons were analyzed both per individual feature, as well as per set of changes in order to identify possible dependencies between various decisions.
Results for scope changes for project A. As we can see in Figure 1.1, we decided to include changes from both before and after MS4. The results are presented below:

Zone 1 - A significant scope reduction after MS3: This zone shows a

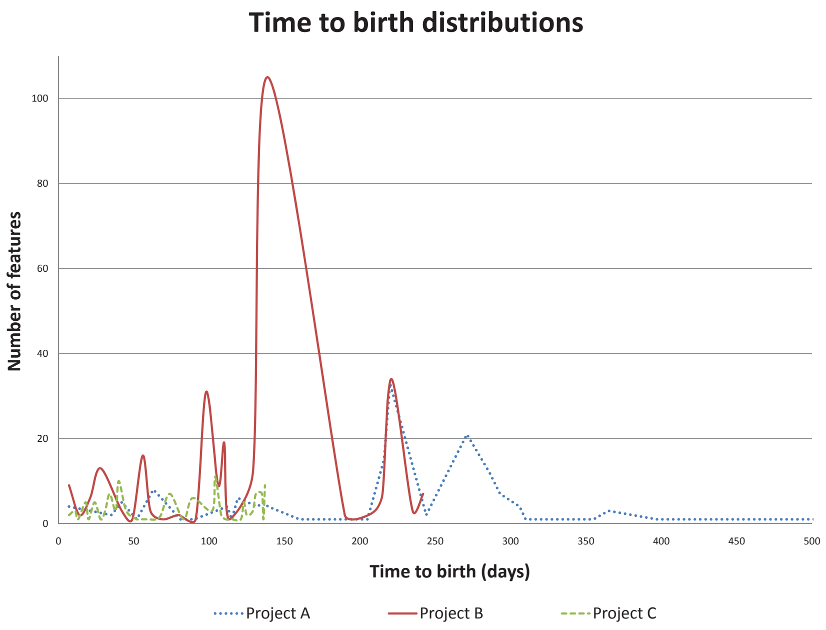


Figure 1.7: Time to birth distributions (M4).

large scope reduction that happened between MS3 and MS4 in the platform project. The analysis revealed that this zone includes two reasons for de-scoping. The first one is the strategic reason and the other one is the cancellation of one of the products from the product line project.

Zone 2 - A large scope inclusion after MS4: This zone shows a large set of features introduced to the scope of the project after MS4. The reasons for this change turned out to be an ongoing work to improve performance requirements. Because of this reason, it was decided shortly after MS4 to include these features into the scope.

Zone 3 - A large scope inclusion together with a parallel scope exclusion: This zone represents a desired behavior of the process used in the company. The large scope inclusions show a new flow of requirements related to one of the platform releases. Our responders confirmed that all three sets of features, separated from each other on the graph, represent an introduction of a new requirements flow. The focus for the analysis in this case was to examine if there was any relation between inclusion of new requirements and exclusion of other requirements at the same time. The set of de-scoped features turned out not to be related to the big scope inclusion. As described by the interviewed requirements manager, the main reasons for these scope changes were defined as "stakeholder business de-

Table 1.3: Results from practitioners' ranking of proposed measurements.

Rank	Responder 1	Responder 2	Responder 3
1	M2	M2	M5
2	M5	M5	M1
3	M1	M4	M2
4	M4	M1	M4
5	M3	M3	M3

cision", which means that the previously defined plan was changed to accommodate other aspects of the product portfolio.

Zone 4 - Some incremental scope inclusions introduced very late in the project: As we can see in Figure 1.1, this zone covers many of the incremental scope inclusions by the end of the analyzed time. Since late scope extensions may put reliability at risk, we investigated why they occurred and found out that there are many reasons behind this phenomena. One of the large changes, that involved four features, was caused by administrative changes in the requirements database. Some additional five features were included into the scope as a result of a late product gap analysis. A gap analysis is a task that requirements managers perform in order to ensure that the scheduled product features are covered by the corresponding platform project. Finally, seven features introduced into the scope turned out to be a result of late negotiations with one of the customers.

Zone 5 - Late removal of previously accepted features: In this zone, we analyze removal of the features that were analyzed in zone 2. We have asked our responders why initially accepted features later were de-scoped. They replied that despite these features were initially approved, a new decision had to be made mainly due to a lack of available development resources. We also performed a quantitative analysis of reasons for de-scoping in Project A. The results are presented in Figure 1.8. We have analyzed 120 de-scoping decisions that belong to project A. The result is shown in percentages in Figure 1.8, summing up to 100%. As we can see, 33% of the de-scoping decision were caused by a stakeholder's business decision, and 29% by a lack of resources, while 9% of the decisions were caused by changes in product portfolios. Our largest category, stakeholder business decision is similar to the category mentioned by Wohlin et al (2005) called "Stakeholder priority of requirement". Therefore we can assume that the dominant reason for both inclusions and exclusion of certain requirements in a specific release does not differ significantly. Furthermore, criteria such as requirements volatility and resource availability seem to appear both in our study and in (Wohlin and Aurum, 2005).

As an additional validation step, we asked three practitioners working

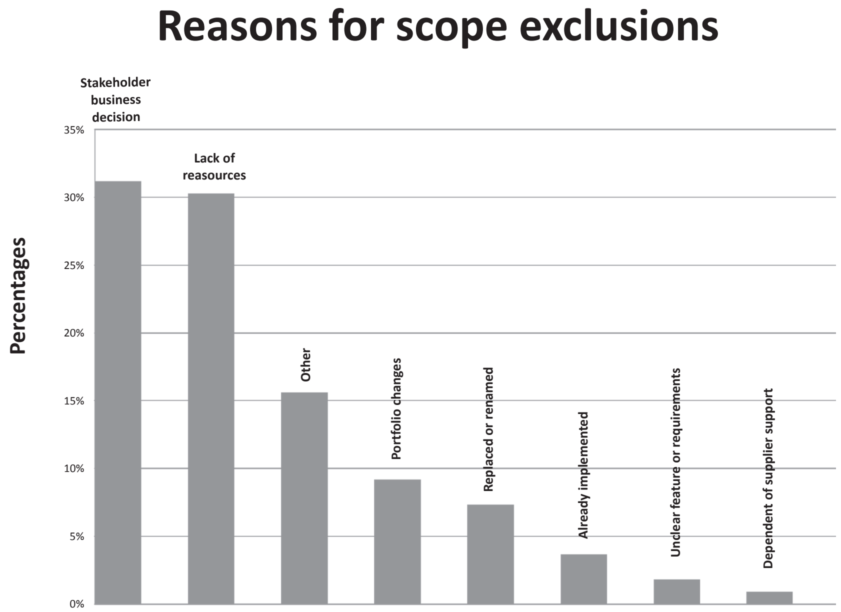


Figure 1.8: Quantitative analysis of reasons for removing features from the scope of the Project A.

with scoping to rank the proposed measurements. As a criterion for ranking, we chose usefulness in understanding the scoping processes and in defining future improvements. The measurement ranked as number one is considered to be the most useful one, while the one ranked in position five is the least useful one. The results are presented in Table 3. As we can see in Table 3, M3 was ranked as the least useful, while M2 and M5 were placed in the top three positions for all responders.

7 Conclusions

According to Basili and Rombach (1988), software engineers and managers need real-time feedback in order to improve construction processes and products in ongoing projects. In the same manner, the organization can use post mortem feedback in order to improve the processes of future projects (Karlsson et al, 2006). Furthermore, visualization techniques used in software engineering have already proven to amplify human cognition in data intensive applications, and support essential work tasks (Botterweck et al, 2008). Our visualization technique provides feedback about ongoing scoping activities as well as a visualization of past project scoping activities. Measurements presented in this paper are complementing our visualization technique by quantitative characterization and qualitative rationale for scoping decisions. The results in terms of usefulness of the proposed visualization technique and scope tracking measurements were acknowledged by practitioners involved in their development as valuable since they confirm the volatility of the scope and provide a tool to analyze the various aspects of this phenomenon. The results were then used by the case company to adjust the process towards more flexibility in scope setting decisions, and a clearer scope responsibility. Our solution has confirmed to outperform the previously used table-based textual method to track the scope changes in the case company. It gives a better overview of the scoping process of the whole project on a single page size graph. The industrial evaluation has indicated that our method can be applied to large scale projects, which demonstrates the scalability of the method. Finally, the managers at the case company decided that our visualization technique should be implemented as a standard practice and is currently in widespread usage at the case company. Even if the characteristics of scope changes found may be particular to this case study, we believe that the manner in which these graphs together with measurements are used to increase the understanding of the performance of the scoping process is generally applicable.

Limitations. As for any empirical study, there are threats to the validity. One threat is related to the mapping between measurements and external attributes. In software engineering we often want to make a statement of an external attribute of an object. Unfortunately, the external attributes are mostly indirect measurements and they must be derived from internal attributes of the object (Wohlin et al, 2000). We are aware that our mapping can be one of several possible mappings between internal and external attributes. We address its correctness by evaluating external attributes with practitioners in the case company. Another threat is related to the generalization of our results. Although the company is large and develops technically complex products, it cannot be taken as a representative for all types of large companies and hence the results should be interpreted with some caution. Finally, theoretical validation is context dependent and thus needs to be redone in every new context.

Further work. Additional studies of scope dynamics visualization in other cases would further increase our understanding of their usefulness. Enhanced tool support with the possibility of zooming interactively may be useful, as well as depiction of size and complexity of features by visualizing their relation to the underlying system requirements. How to optimize usability of such a tool support, and the search for new possibilities while observing practitioners using the visualization techniques, are also interesting matters of further research.

Acknowledgments

This work is supported by VINNOVA (Swedish Agency for Innovation Systems) within the UPITER project. Special acknowledgments to Thomas Olsson for valuable contributions on scope tracking measurements and to Lars Nilsson for valuable language comments.

Bibliography

Basili V, Rombach D (1988) The tame project: Towards improvement-oriented software environments. IEEE Transactions on Software Engineering 14(6):758–773

Botterweck G, Thiel S, Nestor D, bin Abid S, Cawley C (2008) Visual tool support for configuring and understanding software product lines. In: Proceedings of the 12th International Software Product Line Conference (SPLC 2008), pp 77–86

Cooper R (1990) Stage-gate systems: A new tool for managing new products. Business Horizons 33(3):44–54

Ebert C, DeMan J (2005) Requirements uncertainty: Influencing factors and concrete improvements. In: Proceedings of the 27th International Conference on Software Engineering (ICSE 2005), pp 553–560

Fenton N, Pfleeger S (1996) Software Metrics A Rigorous & Practical Approach. Thomson Publishing

Greer D, Ruhe G (2004) Software release planning: an evolutionary and iterative approach. Information and Software Technology 46(4):243–253

Karlsson J, Ryan K (1997) A cost-value approach for prioritizing requirements. IEEE Software 14(5):67–74

Karlsson L, Regnell B, Thelin T (2006) Case studies in process improvement through retrospective analysis of release planning decisions. International Journal of Software Engineering and Knowledge Engineering 16(6):885–915

Kishi T, Noda N, Katayama T (2002) A method for product line scoping based on decision-making framework. In: Proceeding Second International Software Product Lines Conference (SPLC 2002), pp 53–65

Kitchenham B, Pfleeger SL, Fenton N (1995) Towards a framework for software measurement validation. IEEE Transactions on Software Engineering 21(12):929–944

Pohl K, Bockle G, van der Linden F (2005) Software Product Line Engineering: Foundations, Principles and Techniques. SpringerVerlag

Savolainen J, Kauppinen M, Mannisto T (2007) Identifying key requirements for a new product line. In: Proceedings of the 14th Asia-Pacific Software Engineering Conference (APSEC 2007), pp 478–485

Schmid K (2002) A comprehensive product line scoping approach and its validation. In: Proceedings of the 24th International Conference on Software Engineering (ICSE 2002), pp 593–603

Wnuk K (2010) Distributions of derived m3 can be accessed at. `http://www.cs.lth.se/home/Krzysztof_Wnuk/RE_09/NumberOfChangesNeededToRemoveTheFeature.bmp`

Wnuk K, Regnell B, Karlsson L (2008) Visualization of feature survival in platform-based embedded systems development for improved understanding of scope dynamics. In: Proceedings of the Third International Workshop on Requirements Engineering Visualization (REV 2008), pp 41–50

Wohlin C, Aurum A (2005) What is important when deciding to include a software requirements in a project or release? In: Proceedings of the International Symposium on Empirical Software Engineering (ISESE 2005), pp 246–255

Wohlin C, Runeson P, Höst M, Ohlsson M, Regnell B, Wesslen A (2000) Experimentation in Software Engineering An Introduction. Kluwer Academic Publishers

Paper II

Replication of an Experiment on Linguistic Tool Support for Consolidation of Requirements from Multiple Sources

Krzysztof Wnuk, Martin Höst, Björn Regnell
Department of Computer Science,
Lund University, Sweden
{Krzysztof.Wnuk,Martin.Host,Bjorn.Regnell}@cs.lth.se

Empirical Software Engineering,
Volume 17 Number 3, 2012, Pages 305-344

Abstract

Large market-driven software companies continuously receive large numbers of requirements and change requests from multiple sources. The task of analyzing those requests against each other and against already analyzed or implemented functionality then recording similarities between them, also called the requirements consolidation task, may be challenging and time consuming. This paper presents a replicated experiment designed to further investigate the linguistic tool support for the requirements consolidation task. In this replication study, 45 subjects, working in pairs on the same set of requirements as in the original study, were assigned to use two methods for the requirements consolidation: (1) lexical similarity and (2) searching and filtering. The results show that the linguistic method used in this experiment is not more efficient in consolidating requirements than the searching and filtering method, which contradicts the findings of the original study. However, we confirm the previous results that the assisted method (lexical similarity) can deliver more correct links and miss fewer links than the manual method (searching and filtering).

1 Introduction

Requirements engineering in a market-driven context can be characterized by continuous elicitation, time-to-market constraints, and strong market competition (Natt och Dag, 2006a; Regnell and Brinkkemper, 2005). In this context, requirements are continuously arriving from multiple sources, throughout the development process (Regnell et al, 1998). When the company is growing and expanding, more products are created which result in a more complex variability structure, and more effort is needed to handle product customizations, for example by utilizing the Software Product Line (SPL) concept (Pohl et al, 2005). This constant flow of requirements needs to be analyzed from the perspective of new market opportunities and technical compliance. In a case when a company is large and develops complex software solutions, the quantity of information to constantly analyze and assess may severely impede the analytical capacity of requirements engineers and managers (Gorschek et al, 2007; Leuser, 2009). Providing a method that can assist in analyzing large numbers of natural language requirements for the purpose of finding and recording similarities between them can significantly reduce time needed to perform the task (Cleland-Huang et al, 2007), help to miss fewer requirements links (Natt och Dag et al, 2006) and increase the accuracy of the task.

The process of analyzing incoming requirements from customers or customer representatives (also called proxy-customers) against requirements already present in the requirements repository can be called *requirements consolidation*. This process includes gathering incoming documents, finding similarities, and merging or linking similar descriptions into a consolidated single description that covers all analyzed aspects. This process can also be a part of the broader impact analysis task. The core of the requirements consolidation process is finding the similarities between requirements and recording them by making links between them (Natt och Dag et al, 2006). However, the number of possible links grows exponentially with the increase of the number of requirements to analyze, which may result in overwhelming the company's management and analytical skills (Leuser, 2009). As a remedy to this problem, Natt och Dag et al (2006) developed and evaluated a method for requirements consolidation that utilizes linguistic techniques and provides a list of requirements that are the most similar to the currently analyzed requirement. The evaluation of this method showed that using the method can significantly improve the performance of the consolidation process as well as the number of correctly linked requirements, and that it can help to miss fewer requirements links (Natt och Dag et al, 2006). However, the *unsupported* method used in the original experiment was limited to a simple search functionality, while most currently available requirements management tools offer more advanced filtering and searching techniques.

This replication study has been designed to assess whether the tool with a linguistic analysis of the similarity between requirements can still perform better than currently available commercial requirements management tools in the task of requirements consolidation. A replicated experiment has been chosen due to its falsifiable nature. Replications provide a possibility to evaluate whether the output parameters of a system remain stable if one or more input parameters are systematically changed.

In this experiment, two subject groups, working in pairs, were asked to consolidate two requirements sets by finding and linking requirements that address the same underlying functionality. This replication reuses the original procedures in terms of the study design, experimental steps and the two requirement sets. The changes to the original experiment are: (1) using another set of subjects which were asked to work in pairs due to unexpected housing issues and (2) changing one of the treatments. Due to a limited number of available computers in the laboratory room, the subjects were asked to work in pairs on the assignment. Given this context, this replication study can be classified according to Shull et al (2008) as a dependent replication. However the classification provided by Shull does not define if a replication where both the population and one of the methods used to test the hypotheses is changed can also be categorized as an exact replication. Shull et al (2008) only mention changing either the population or the artifact on which the technique is applied. According to the classification by Basili et al (1999), this replication type is the one that varies the research hypotheses. The unchanged object in this replication study, also called the *assisted* method, is a research prototype tool, called ReqSimile (Natt och Dag, 2006b), that utilizes linguistic analysis to assist in the task of finding similar requirements. The second object, which was changed compared with the original experiment, is called the *manual* method, and it utilizes searching and filtering functionalities implemented in a tool called Telelogic Doors (IBM, 2010a).[1]

The objectives of the study are twofold: firstly to assess if a significant differences between the two methods tested in the original experiment can be confirmed in a replicated experiment setup and secondly to compare the results for the same methods between the two experimental sessions. The objectives are refined to two main research questions in Section 4.

The paper is structured as follows: Section 2 provides industrial problem description. Section 3 provides related work. Section 4 describes the experimental design. Section 5 explains experiment execution procedures. Section 6 describes the experiment results analysis. Section 7 provides an interpretation of results. Section 8 concludes the paper.

[1]The Telelogic DOORS tool has recently changed its vendor and its name to Rational DOORS. However, since the Telelogic Doors version 8.3 was used in this experiment, we will refer to this tool throughout this paper as Telelogic Doors. Both methods were compared for the task of requirements consolidation meaning that comparing the two tools in general is outside of the scope of this paper.

2 Industrial Problem Description

New requirements and changes to existing requirements are inevitable situations at all stages of the system development process (Kotonya and Sommerville, 1998). The two principal requirements management activities that address this phenomena are: (1) change control and (2) change impact assessment. The change control ensures that, if a change is accepted, its impact on design and implementation artifacts will be addressed. The change impact assessment warrants that proposed changes have a known impact on the requirements and software system (Kotonya and Sommerville, 1998). A company that is operating in a market-driven mode should continuously monitor the market situation by checking competitors' latest achievements, researching market needs and collecting all possible feedback from the market in a chase for achieving or maintaining the competitive advantage within its operational business. This pursuit after an optimal market window, together with other reasons, creates a constant flow of new requirements and ideas throughout the entire software product lifetime (Karlsson et al, 2002). As a result, the requirements process for market-driven contexts needs to be enriched with procedures to capture and analyze this constant flow of requirements (Higgins et al, 2003).

As pointed out by practitioners from large companies (Berenbach et al, 2009), when development projects grow in complexity and new products are released to the market with many features, the importance of good practices in requirements management grows. In the case when a company is large and operates globally, the diversity of customers and the complexity of software products can make the list of sources of new requirements and change requests extensively long, including: customers and proxy-customers (marketing, customer representatives and key account managers), suppliers, portfolio planners and product managers. The company's requirements analysts should, in this, case analyze all incoming requirements and change requests in order to find an optimal set of requirements that will address the needs of as many customers as possible. In this context, the concept of Software Product Lines (SPL) (Pohl et al, 2005) is often used to increase the reuse of common components while providing necessary diversity of similar products, requested by various customers.

Change management in a Software Product Lines context can be particularly challenging, for example, because of the extensive and often exhaustive variability analysis that has to be performed while analyzing the impact of a change. Moreover, the requirements analyst has to consider if a certain new requirement or request has already been analyzed and what was the result of this analysis. One of the methods to assist with the analysis of incoming requirements versus those already present in the requirements database is to find and record similarities, making traceability links. In the industrial case example, provided by the original experiment, the experts became frustrated during the analysis because they had to identify compliance to the same or very similar requirements multiple times. Large parts of the new versions of requirements request documents, arriving from the same customer are typically the same as previous versions. Furthermore, the same and very similar requirements can appear in the request from different customers (Natt och Dag et al, 2006). Providing an automatic or semi-automatic method of analyzing similarity between incoming requirements could significantly decrease the amount of time needed to perform this task.

The process of finding and recording similarities between software development artifacts is a part of the requirements traceability activity, which has been widely recognized as a useful method for recording relations and dependencies between software project artifacts for the purposes of change and impact analysis tasks (Ramesh et al, 1995; Wiegers, 2003; Antoniol et al, 2002; Jarke, 1998; Gotel and Finkelstein, 1994). The core of the requirements traceability task is to find and record the dependencies between the traced elements, which are assumed to be exhibited by their lexical similarity (Natt och Dag, 2006a). The importance of requirement traceability is significant; the U.S. Department of Defense invested in 2001 about 4 percent of its total IT budget on traceability issues (Ramesh and Jarke, 2001). Other large companies, have also stressed the importance of implementing traceability in their industry projects (Samarasinghe et al, 2009; Berenbach et al, 2009; Konrad and Gall, 2008; Panis, 2010; Leuser, 2009).

However, despite recognition of its importance, implementing a successful traceability in practice is challenging (Cleland-Huang et al, 2002). The task of finding relationships between the elements and establishing traces between them is a “mind numbing” (Hayes et al, 2003), error prone and time consuming activity. Moreover, maintaining a traceability scheme is difficult because the artifacts being traced continue to change and evolve as the system is developed and extended (Zowghi and Offen, 1997; Strens and Sugden, 1996). Furthermore, as pointed out by Leuser (2009), current traceability approaches used in practice are cumbersome and very time consuming, mainly because they are almost completely manual. The size of requirements specifications in large industrial projects may reach thousands of requirements (Leuser, 2009; Konrad and Gall, 2008). To tackle these issues, several researchers proposed using Information Retrieval (IR) methods such as the Vector Space Model (VSM), also used in this experiment, (Antoniol et al, 2002; Cleland-Huang et al, 2007; Natt och Dag et al, 2004), the Probabilistic Network Model (Cleland-Huang et al, 2005, 2010), and Latent Semantic Indexing (LSI) (De Lucia et al, 2007; Huffman Hayes et al, 2006; Lormans and van Deursen, 2006; Marcus and Maletic, 2003) for semi-automatic recovery of traceability links.

3 Related Work

Replications play an important role in software engineering by allowing us to build knowledge about which results or observations hold under which conditions (Shull et al, 2008). Unfortunately, replications in software engineering are still rarely reported (Ivarsson and Gorschek, 2009). A recent survey of controlled experiments in software engineering revealed that replications are still neglected by empirical researchers, only 18% of the surveyed experiments are reported as replications (Sjøberg et al, 2005). Moreover only 3.9% of analyzed controlled experiments can be categorized according to the IEEE taxonomy as requirements/specification related (Sjøberg et al, 2005; IEEE, 2010).

The awareness of new possibilities that Natural Language Processing (NLP) can bring to requirements engineering has been present from the beginning of the requirements engineering discipline, when Rolland et al (1992) discussed the natural language approach for requirements engineering. Shortly after, Ryan (1993) warned that although natural language processing provides a variety of sophisticated techniques in the requirements engineering field, they can only support sub-activities of requirements engineering and that the process of using natural language processing techniques has to be guided by practitioners. The possibilities mentioned by Ryan (1993) and Rolland et al (1992) have later been explored by a number of research studies and publications, where applications of various NLP techniques in supporting requirements management activities were evaluated and discussed. Among those that include some kind of empirical evaluations, the vast majority of natural language process tools are used to examine the quality of requirements specifications in terms of, for example, the number of ambiguities (Fantechi et al, 2003) by assigning ambiguity scores to sentences depending on the degree of syntactic and semantic uncertainty (Macias and Pulman, 1995), or detecting ambiguities by applying an inspection technique (Kamsties et al, 2001). Furthermore, Rupp et al (2000) produced logical forms associated with parsed sentences to detect ambiguities. Among other quality attributes of requirements artifacts that natural language processing attempts to analyze and improve, Fabbrini et al (2001) proposed a tool that assesses understandability, consistency, testability, and correctness of requirements documents. Providing measurements that can be used to assess the quality of a requirements specification document is the aim of the ARM tool proposed by Wilson et al (1997). Mich et al (2002) reported on an experiment designed to assess the extent to which an NLP tool improves the quality of conceptual models. Finally, Gervasi et al (2000) used natural language processing techniques to perform a lightweight validation (low computational and human costs) of natural language requirements.

Apart from the quality evaluation and assurance tasks, NLP techniques have also been applied to the task of extracting abstractions from textual documents (Aguilera and Berry, 1991; Goldin and Berry, 1997) and helping combining crucial requirements from a range of documents that include standards, interview transcripts, and legal documents (Sawyer et al, 2002). Sawyer at al (2005) have also reported how corpus-based statistical language engineering techniques are capable of providing support for early phase requirements engineering. Rayson et al (2001) reported experiences from a project where probabilistic NLP techniques were used to identify and analyze domain abstractions. Their results were further supported by a later study by Sawyer et al (2004), where ontology charts of key entities were produced using collocation analysis. The continued interest in this issue has been reported by Gacitua et al (2010) who proposed a new technique for the identification of single- and multi-word abstractions named Relevance driven Abstraction Identification (RAI). Finally, Gervasi et al (1999) used lexical features of the requirements to cluster them according to specific criteria, thus obtaining several versions of the requirements document.

The ReqSimile tool evaluated in this paper uses a correlation to measure lexical similarity and thus rank candidate requirements for linking, presenting to the user the "top" subset of those requirements. The linguistic method, also called the *cosine* measure, uses a vector-space representation of requirements where each requirement is represented using a vector of terms with the respective number of occurrences (Natt och Dag et al, 2004; Manning and Schütze, 2002). Each term can be seen as a dimension in an N-dimensional space while a whole requirement can be represented as a point in the N-dimensional space. Similar requirements will be represented in this space as *m* points closely clustered. From the matrix, which shows how many times a term appears in each requirement, the information may be derived about how many terms the two requirements have in common. The very similar requirements will result in closely clustered points in this vector space (Manning and Schütze, 2002). In the evaluated method (Natt och Dag, 2006a) a frequency of terms has been used, instead of counting the occurrences. The cosine correlation measure is often chosen in text retrieval applications for the purpose of finding similar requirements, as it does not depend on the relative size of the input (Manning and Schütze, 2002).

$$\sigma(f,g) = \frac{\sum_t w_f(t) * w_g(t)}{\sqrt{\sum_t w_f(t)^2 * \sum_t w_g(t)^2}} \tag{2.1}$$

The measure in equation 2.1 is used for calculating the degree of similarity, where *f* and *g* are two requirements, *t* ranges over terms, and *w(t)* denotes the weight of term *t*. The term weight is typically a function of the term frequency, since while the number of times a word occurs is relevant, its relevance decreases as the number gets larger (Manning and Schütze, 2002). However, there is a challenge in the way stemming rules are used in this method. For example, the stemming rules do not reduce the verb *containerization* and the noun *container* to the same stem. From a semantic point of view this is perfectly correct, but as the two terms concern the same domain concept their association should be utilized to increase the similarity measure. The realization of the Vector Space Model used in this paper does not support this association. Another potential problem has to do with synonyms as they are not considered in the model. Finally, as mentioned in (Natt och Dag, 2006a), there is no guarantee that two requirements that are similar according to the $\sigma(.)$ measure are indeed related. The method evaluated does not consider hypernyms and hyponyms (Jackson and Moulinier, 2002).

The ReqSimile tool evaluated in this paper is not the only research tool that provides support for requirements traceability. Hayes et al (2007) proposed a REquirements TRacing On-target (RETRO) tool that uses the LSI technique to find similarities between analyzed elements and help the analyst with making traceability links. Lin et al (2006) proposed a Web-based tool called POIROT that supports traceability across distributed heterogeneous software artifacts. A probabilistic network model is used by POIROT to generate traces between requirements, design elements, code and other artifacts stored in distributed third party case tools.

The second method evaluated in this study uses searching and filtering functionalities provided by Telelogic DOORS tool (IBM, 2010b). According to the product documentation, the "finding text in a module" function can search for all the objects that contain a specific search string. The tool displays the options of the search that have already been set in the search window. The possible options are: (1) highlight matches, (2) match case and (3) use regular expressions. To change the search options, the *Advanced* tab in the same window has to be used. Additionally, it is possible to select the attributes included in the search, for example object heading or object text. The tool provides UNIX-style regular expressions support when searching for text. For example, using **c.t** will search for all three letter words that start with **c** and end with **t**. Using **200[123]** will search for either 2001, 2002, or 2003. Subjects during the experiment can also use filters to control what data is displayed on the screen. The tool provides two types of filters: simple and advanced. Simple filters can be used to filter: (1) the contents of every attribute of type text or string, (2) object heading number, (3) the content of any column or the value of a single attribute of any type. Additionally, it is possible to filter on the basis of whether the object has links or is either the current object or a leaf object. Using advanced filters gives the possibility to combine simple filters to create complex filters, specify filter options that control what is displayed.

Using filters requires more steps than using the searching functionality. First, the filter has to be defined and its attributes have to be set. After this step, the user has to define if the filter should match case option or regular expressions. Finally, it is possible to filter only objects that have certain types of links, such as in-links, objects that are leafs and filter the content of columns. While using advanced filters, it is possible to combine previously defined simple filters by using **And, Or** and **Not** to combine them into logical expressions. It is also possible to control the output of applying a filter. The possible options are: (1) show ancestors or (2) show descendants of the object that match the filter criteria and (3) display all table cells and their contents regardless of whether they match the filter criteria. According to the product documentation, the Telelogic DOORS tool does not provide any auto-completion, stemming or proximity search options. Exclusion searches are possible by combining two simple filters where one is negated by using the "NOT" logical expression. [2]

The requirements consolidation task can, in a broad perspective, be considered as the more general task of negotiating the scope of the future project with multiple stakeholders. Assuming this broad definition of the task, we discuss alternative approaches of supporting this negotiation process. Fricker et al (2010) propose a negotiation process, called handshaking with implementation proposals. The process has been used to communicate requirements effectively, even in situations where almost no written requirements exist and where distance separates the customer from the developers. The architectural options are used in this case to understand requirements and make implementation decisions that will create value. Sommerville and Sayer (1997) proposed a viewpoint-based approach to requirements engineering which may be used to structure the requirements description and expose conflicts between different requirements. As systems usage is heterogeneous, viewpoints can organize different types of information needed to specify the system and by that help to structure the process of requirements elicitation. Breaux (2009) uses grounded theory to analyze regulations and legal documents for the purpose of ensuring that the software system to be built is demonstrably compliant with relevant laws and policies.

4 Experimental Design

The two main objectives of this study, presented in Section 1, can be further refined to the following research questions:

Q1: Can significant differences between the *assisted* and the *manual* methods that were achieved in the original experiment be confirmed in a replicated experiment where the original *manual* method is replaced with a key-

[2]The information about searching and filtering functionalities has been based on the manual for DOORS version 8.3. The instruction of how to use Telelogic Doors for linking used in this experiment is available at `http://fileadmin.cs.lth.se/serg/ExperimentPackages/ReplicationReqSimile/HelpSheetDoors.pdf`

word searching and filtering tool?

Q1a: Is the *assisted* method more efficient in consolidating two requirements sets than the *manual* method (where efficiency is calculated as the number of analyzed requirements)?

Q1b: Is the *assisted* method more correct in consolidating two requirements sets by assigning more correct links than the *manual* method (correctness is calculated as a number of correctly linked requirements)?

Q1c: Does the *assisted* method help to miss fewer requirements links than the *manual* method?

Q2: Is there any difference between the original and replicated experiment sessions for the same method?

Q2a: Is there any difference in the results for the *assisted* method between the original and the replicated experiments?

Q2b: Is there any difference in the results for the *manual* methods between the original and the replicated experiments?

The aim of Q1 is to assess if the results obtained in the original experiment holds even if one of the tools is changed. The research question Q1 is divided into three sub-questions, where each of them is explicitly addressing various quality aspects of the consolidation process. Question Q2 aims to assess the difference between the two experiments. The possible differences provide valuable input regarding the nature of the consolidation task and the subjects used in both experiments.

The central part of the requirements consolidation task is finding similarities between the two sets of requirements as described in detail in Section 3. The methods evaluated in the experiment were implemented in two tools: (1) ReqSimile (Natt och Dag et al, 2006) and (2) Telelogic Doors (IBM, 2010a). As mentioned in Section 1, the goal of the study is not to evaluate the tools in general, but to compare the methods that they provide. The planning phase was based on the original experiment (Natt och Dag et al, 2006) and, when possible, the original material is reused and extended according to the guidelines of designing experiments presented by Wohlin et al (2000). In order to draw more general conclusions, the authors put additional effort into minimizing the difference between this experiment design and the original experiment design.

Since most of the design has been reused from the original experiment, the evaluation of the experiment design for the replication sake was limited to checking additional changes. The changes concern questionnaire improvements and new instructions regarding the use of the Telelogic Doors tool (IBM, 2010a). The experiment design was evaluated by an additional researcher, experienced in conducting empirical research in software engineering, before executing the experiment. The same researcher partici-

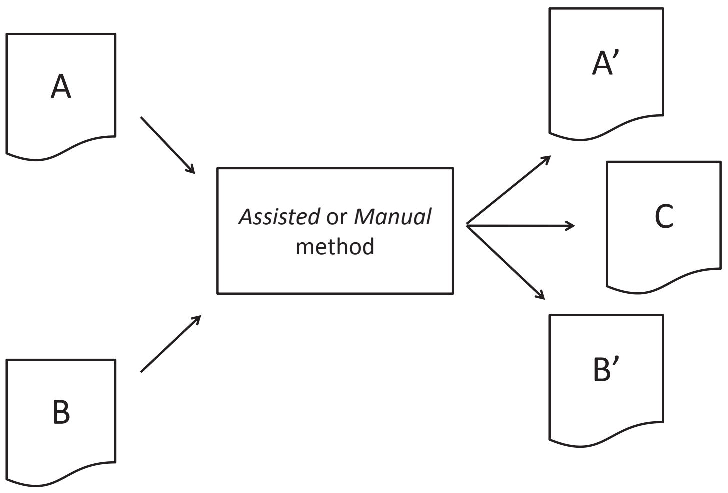


Figure 2.1: The process of using the support tool for requirements consolidation.

pated in the pilot study where both tools were used to find similar requirements and create links between them. Comments and suggestions regarding readability and understandability of the laboratory instructions were given and later implemented. Finally, since the requirements sets used in this experiment were the same as in the original experiment, the correct answer to the consolidation task remained unchanged [3].

Similarly to the original experiment, this replication study was also conducted in the form of a laboratory experiment, since it captures the consolidation problem in an untainted way. Figure 2.1 depicts the conceptual solution of the consolidation activity. To the left in Figure 2.1, two requirement sets A and B are shown. They represent two consecutive submissions of requirements specifications from the same key customer. We can also assume that the earlier specification is set A in this case, and that it would have already been analyzed and the result from the analysis is available in the central requirements database. The subjects use one of the tools, either Telelogic Doors for the *manual* method (IBM, 2010a) or ReqSimile for the *assisted* method (Natt och Dag et al, 2006), to find requirements in the set B that were already analyzed in the set A and to mark them by assigning a link between them. The output of the process is shown to the right

[3]The experiment pack can be accessed at `http://fileadmin.cs.lth.se/serg/ExperimentPackages/ReplicationReqSimile/Experiment.rar`

of Figure 2.1. The subset A′ comprises all requirements that are not linked to any requirement in the set A. The subset B′ represents all new requirements that have not previously been analyzed. Finally, there is the subset C, which comprises all requirements in the new specification that previously have been analyzed. The analyst would then send the requirements in set B′ to the experts for analysis. The experts are thus relieved from the burden of re-analyzing the requirements in subsets A′ and C.

4.1 Goals, Hypothesis, Parameters and Variables

The variables in this experiment were kept unchanged from the original study (Natt och Dag et al, 2006). They can be grouped into independent, controlled and dependent:

- The independent variable is the method used in the experiment. The two methods compared are *manual* and *assisted*.
- The controlled variable is the experience of the participants. In order to analyze the individual experience of the subjects, a questionnaire was used.

The dependent variables are:
T - time used for the consolidation
N - the number of analyzed requirements
N_{cl} - number of correct links
N_{il} - number of incorrect links
N_{cu} - number of correctly not linked
N_{iu} - number of missed links (incorrectly not linked)

These dependent variables are used to analyze the hypotheses. The number of analyzed requirements is used in case the subjects are not able to analyze all requirements, which will affect N_{iu} and N_{cu}. The hypotheses for comparing the *manual* and the *assisted* method remain unchanged from the original experiment design. The rationale of the proposed hypotheses is based on the following theory regarding using the *assisted* method. The *assisted* method provides a list of candidate requirements ranked by their similarity degree to a currently analyzed requirement. As a result the requirements analysts has to review only a subset of all possible combinations, for example the top ten candidate links. Thus we state a hypothesis that the *assisted* method can help to analyze requirements faster. Moreover, since the most similar requirements are placed next to each other on the list of candidates it is easier to read all potential candidates that exhibit high degree of lexical similarity. The result is expected to be an increased number of correct links, better precision and accuracy. Finally, the sorting according to lexical similarity should, in our opinion, help to miss fewer correct requirement links, since there is a high probability that all possible links to analyze will be show in the top 10 or 20 candidate requirements. Presented below are six null hypotheses:

(H_0^1) The *assisted* method results in the same number of requirements analyzed per minute, N/T, as the *manual* method.

(H_0^2) The *assisted* method results in the same share of correctly linked requirements, $N_{cl}/(N_{cl} + N_{iu})$, as the *manual* method.

(H_0^3) The *assisted* method results in the same share of missed requirements links, $N_{iu}/(N_{cl} + N_{iu})$, as the *manual* method.

(H_0^4) The *assisted* method results in the same share of incorrectly linked requirements, N_{il}/N, as the *manual* method.

(H_0^5) The *assisted* method is as precise, $N_{cl}/(N_{cl} + N_{il})$, as the *manual* method.

(H_0^6) The *assisted* method is as accurate, $(N_{cl} + N_{cu})/(N_{cl} + N_{il} + N_{cu} + N_{iu})$, as the *manual* method.

Since the subjects may not use exactly the same time for the task, the performance is normalized as the number of analyzed requirements divided by the total time spent on the consolidation task (in minutes).

4.2 Subjects

In this study, a different set of subjects compared to the original experiment, although from the same kind of population was used. The sample includes participants of the course in Requirements Engineering at (Lund University, 2011a). The course is an optional master-level course offered for students at several engineering programs including computer science and electrical engineering. It gives 7.5 ETCS points (ECTS, 2010) which corresponds to five weeks full time study. Although the experiment was a mandatory part of the course, the results achieved by the subjects had no influence on their final grade from the course. The students were between 24 and 41 years old with an average of 27 years. There were 4 female and 41 male students. Before conducting the experiment, the subjects had been taught requirements engineering terminology and had gained practical experiences through their course project. The result from the pre-test questionnaire revealed that the difference in English reading and writing were small, varying from "very good knowledge" for the majority of subjects to "fluent knowledge" for some of them. When it comes to the industrial experience in software development of the subjects, most of them reported no experience at all (28 out of 45 students). Among the subjects that reported any degree of industrial experience, the length of the experience varied be-

Table 2.1: The number of years of industrial experience in software development for pairs of the subjects that participated in this replication. The remaining pairs of subjects exhibited no industrial experience for both pair members. The letter A indicates that a pair of subjects used the *assisted* method while the letter M indicates that a pair of subjects used the *manual* method. The IDs (Mx and Ay) are consistent with Table 2.5. The rows highlighted **bold text** indicate data points that were removed from the analysis (outliers).

Pair of subjects	Experience of the first subject in the pair (in years)	Experience of the second subject in the pair (in years)
A1	0.5	1.5
A3	1	1
A7	**1**	**2**
A10	0	1
A11	0.5	1
M4	1	2
M5	0.5	0
M6	0.25	0
M7	0.5	1
M8	0.25	1.5

tween three months and two years with an average value of 11 months. The analysis of industrial experience in pairs of subjects revealed that ten pairs had varying degrees of industrial experience which was not always equal. The analysis of the experience of both pair members is presented in Table 2.1. The difference in experience varied between three months and 15 months with an average value of nine months. The impact of the industrial experience on the results achieved by these subjects is outlined and discussed in detail in Section 7.1 and Table 2.7.

We have also performed an analysis of the experience of subjects from the project course that the requirements used in this experiment were developed here (Lund University, 2011b). The results of the analysis are presented in Table 2.2. The results revealed that 22 out of the 45 subjects have not taken the course that the requirements originate from, while the rest had taken the course and acted in various roles during the project phase of the course.

Next, the roles taken in the course that the requirements originate from in the pairs formed by subjects were analyzed. For 9 pairs, outlined in Table 2.2, the pairs are formed by an inexperienced person and an expe-

Table 2.2: The experience and the roles of the subjects that participated in the replication from the course that the requirements originate from. The letter A indicates that a pair of subjects used the *assisted* method while the letter M indicates that a pair of subjects used the *manual* method (the numbers are consistent with Table 5). The rows highlighted with **bold text** indicate data points that were removed from the analysis (outliers).

Pair of subjects	Role of the first subject in the pair	Role of the second subject in the pair
M2	Has not taken the course	Tester
M3	Has not taken the course	System group member
M4	Development Manager and Developer	Test Manager and Tester
M5	Tester	Has not taken the course
M6	System Group Member	Project Manager
M7	Has not taken the course	Developer
M9	System Group Member	Developer
M10	**Development Manager**	**Project Manager**
M11	**Has not taken the course**	**System Group Member**
A1	Project Manager	Has not taken the course
A2	Project Manager	Tester
A3	Test Manager and Tester	Has not taken the course
A4	Developer	Developer
A5	Has not taken the course	Tester
A6	Project Manager	Has not taken the course
A7	**Developer**	**Has not taken the course**
A12	Developer	

rienced person from the course. This may have a positive impact on the task, since the more experienced person can help the inexperienced person to understand the nature and origin of the requirements set. However, the more experienced person can bias the consolidation task by bringing knowledge about the requirements sets and possible similar requirements from the course. The remaining seven pairs represented experience from various roles including: developer, development manager system group manager, project manager and tester. Only one pair had the same experience from being a developer, in other cases the roles taken in the course project did not overlap.

When it comes to the experience in analyzing and reviewing requirements, 80% of the subjects declared to have experience only from courses. Among the remaining 20% of the subjects, one pair (M5) had experience from both courses and industry (less than one year). In this case, the second pair member had only experience from courses. Furthermore, in cases (A1,A3 and A11), one pair member reported both experience from courses and less than a year of industrial experience in analyzing and reviewing requirements. In all three cases, these participants were paired with subject reporting only experience from the courses. Finally, in the case of pair M8, one member reported more than one year of industrial experience, while the other pair member reported no experience at all. The analysis of the results achieved by these subjects in relation to their experience is discussed in Section 7.1.

A further question concerned the subject's experience with the tool that implements the *manual* method, that is Telelogic Doors. The analysis indicated that 91% of subjects reported no experience with Telelogic Doors and that they had never heard about the tool. Although four persons have heard about the tool, they have never used it. We can conclude that the subjects are homogenous in this matter and that we can exclude this threat from aspects influencing the results.

4.3 Treatments

The treatments of the experiment are the methods used in supporting the requirements consolidation task. The assisted method is implemented in the ReqSimile tool using linguistic engineering to calculate the degree of similarity between requirements by lexical similarity as a way of approximating semantic similarity (Natt och Dag et al, 2004) (for more details see Section 3).

The other treatment is the *manual* method which comprises searching and filtering functionalities provided by the Telelogic Doors tool (IBM, 2010a). The goal of the experiment is not to compare the two tools in general, but the functionality that they provide to support the requirements consolidation task. The objects used in the original and the replicated experiment are listed in Table 2.3. Compared to the original experiment,

Table 2.3: The treatments and tools used in the original and the replicated experiments.

	Original experiment		**Replicated experiment**	
Treatment	Assisted method	Manual method	Assisted method	Manual method
Tool	ReqSimile	ReqSimileM	ReqSimile	Telelogic Doors

one of the tools was kept unchanged while the second one was changed. The change comprises substituting ReqSimileM from the original design (Natt och Dag et al, 2006) by Telelogic Doors (IBM, 2010a) for the manual method. More information regarding tools used in the replication can be found in Section 4.5.

4.4 Requirements

Two requirements sets were reused from the original experiment. The requirements specifications were produced as a part of a course "Software Development of Large Systems" (Lund University, 2011b). The course comprises a full development project, including: requirements specification, test specification, high-level design, implementation, test, informal and formal reviews and acceptance testing. At the end of the course, the students deliver a first release of the controller software for a commercial telecommunication switch board. Two requirements specifications were randomly selected from the course given in years 2002 and 2003. The requirements have been specified in use case style or features style (Lauesen, 2002), and all are written using natural language. Two requirements sets containing 30 and 160 requirements respectively, were imported to ReqSimlieA and Telelogic Doors. An example of requirements from the specification comprising 30 requirements is depicted in Table 2.4. However, the requirements were neither written by a native English language writer, nor given to a native English language speaking, experienced requirements analyst for editing and rephrasing.

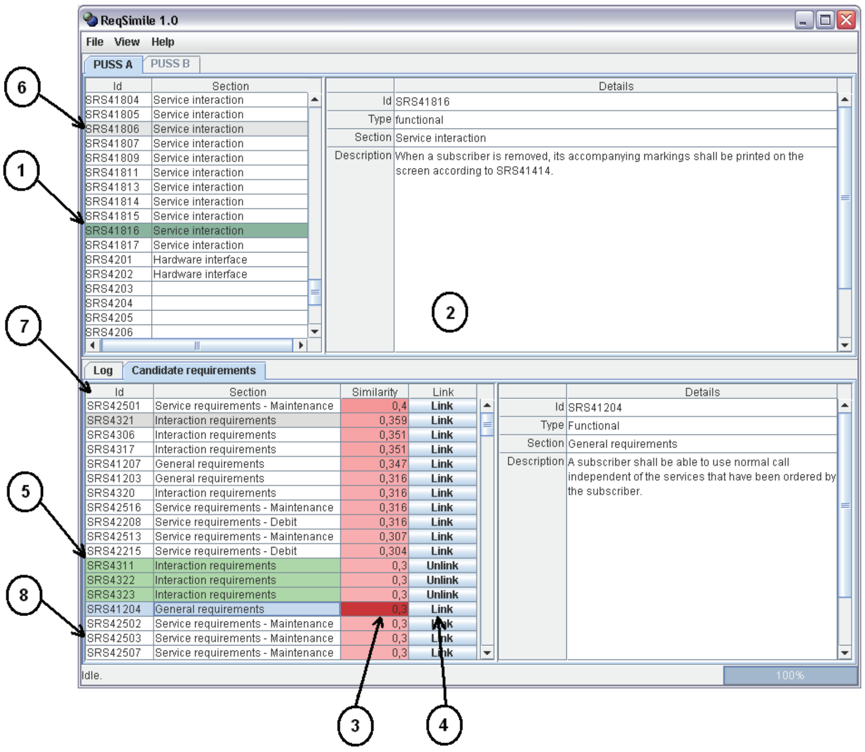
4.5 Tools

In this experiment, one tool remained unchanged from the original experiment while the other tool was changed. The tool that implements the *assisted* method, that is ReqSimile (Natt och Dag, 2006b), was kept unchanged. The user interface of ReqSimile is presented in Figure 2.2. The left side of the top pane of the window presents a list of requirements. The

Table 2.4: Example requirements from the specification comprising 139 requirements.

Key	Id	Type	Selection	Description
3	Scenario13	Functional	Service: Regular call	Regular call-busy Actors: A:Calling subscriber, B:Called subscriber, S:System Prerequisites: Both A and B are connected to the system and are not unhooked. Step 13.1. A unhooks. Step 13.2. S starts giving dial tone to A Step 13.3. A dials the first digit in B_s subscriber number Step 13.4. S stops giving dial tone to A. Step 13.5. A dials the remaining three digits in B_s subscriber number Step 13.8. S starts giving busy tone to A Step 13.9. A hangs up Step 13.10. S stops giving busy tone to A
80	SRS41606	Functional	Service: Call forwarding	Activation of call forwarding to a subscriber that has activated call forwarding shall be ignored by the system. This is regarded as an erroneous activation, and an error tone is given to the subscriber. (Motivation: Together with SR41607, avoids call forwarding in closed loops)
111	SRS41804	Functional	Service interaction	The service call forwarding shall be deactivated if a customer removes either the subscriber from which calls are forwarded or the subscriber to which calls are forwarded.

data has already been pre-processed by the tool so the user can start analyzing requirements for similarities. Selecting a requirement (1) makes the requirement's details display on the right (2) and a list of similar requirements in the other set appear in the bottom pane (7), sorted on the similarity value (3). Requirements that have already been linked in the set of analyzed requirements are highlighted using another (gray) color (6). Requirements that have been linked to the currently selected requirements (1) are highlighted using another (green) color (5). Unlinked requirements are not highlighted (8). Links can be made between the selected requirement (2) and the requirement with the associated link button (4). Once a requirement has been selected (1), the user has to perform two operations, click on the requirement that is supposed to be linked and click the button "link" to make the link. The second tool, described by Natt och Dag et al



Id	Section
SRS41804	Service interaction
SRS41805	Service interaction
SRS41806	Service interaction
SRS41807	Service interaction
SRS41809	Service interaction
SRS41811	Service interaction
SRS41813	Service interaction
SRS41814	Service interaction
SRS41815	Service interaction
SRS41816	Service interaction
SRS41817	Service interaction
SRS4201	Hardware interface
SRS4202	Hardware interface
SRS4203	
SRS4204	
SRS4205	
SRS4206	

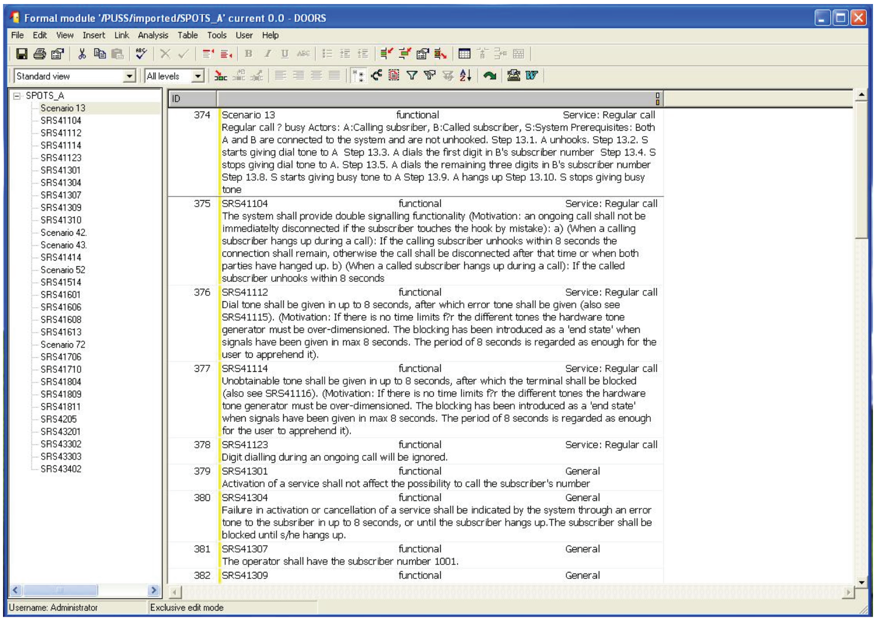
	Details
Id	SRS41816
Type	functional
Section	Service interaction
Description	When a subscriber is removed, its accompanying markings shall be printed on the screen according to SRS41414.

Id	Section	Similarity	Link
SRS42501	Service requirements - Maintenance	0,4	Link
SRS4321	Interaction requirements	0,359	Link
SRS4306	Interaction requirements	0,351	Link
SRS4317	Interaction requirements	0,351	Link
SRS41207	General requirements	0,347	Link
SRS41203	General requirements	0,316	Link
SRS4320	Interaction requirements	0,316	Link
SRS42516	Service requirements - Maintenance	0,316	Link
SRS42208	Service requirements - Debit	0,316	Link
SRS42513	Service requirements - Maintenance	0,307	Link
SRS42215	Service requirements - Debit	0,304	Link
SRS4311	Interaction requirements	0,3	Unlink
SRS4322	Interaction requirements	0,3	Unlink
SRS4323	Interaction requirements	0,3	Unlink
SRS41204	General requirements	0,3	Link
SRS42502	Service requirements - Maintenance	0,3	Link
SRS42503	Service requirements - Maintenance	0,3	Link
SRS42507	Service requirements - Maintenance	0,3	Link

	Details
Id	SRS41204
Type	Functional
Section	General requirements
Description	A subscriber shall be able to use normal call independent of the services that have been ordered by the subscriber.

Figure 2.2: The user interface of the ReqSmiliA tool used in the experiment. The full-size color figure can be found at http://fileadmin.cs.lth.se/serg/ExperimentPackages/ReplicationReqSimile/ReqSimiliA.bmp

(2006) as ReqSimileM was changed in this experiment to Telelogic Doors



374 Scenario 13 functional Service: Regular call
Regular call ? busy Actors: A:Calling subsriber, B:Called subscriber, S:System Prerequisites: Both A and B are connected to the system and are not unhooked. Step 13.1. A unhooks. Step 13.2. S starts giving dial tone to A Step 13.3. A dials the first digit in B's subscriber number Step 13.4. S stops giving dial tone to A. Step 13.5. A dials the remaining three digits in B's subscriber number Step 13.8. S starts giving busy tone to A Step 13.9. A hangs up Step 13.10. S stops giving busy tone

375 SRS41104 functional Service: Regular call
The system shall provide double signalling functionality (Motivation: an ongoing call shall not be immediatelty disconnected if the subscriber touches the hook by mistake): a) (When a calling subscriber hangs up during a call): If the calling subscriber unhooks within 8 seconds the connection shall remain, otherwise the call shall be disconnected after that time or when both parties have hanged up. b) (When a called subscriber hangs up during a call): If the called subscriber unhooks within 8 seconds

376 SRS41112 functional Service: Regular call
Dial tone shall be given in up to 8 seconds, after which error tone shall be given (also see SRS41115). (Motivation: If there is no time limits f?r the different tones the hardware tone generator must be over-dimensioned. The blocking has been introduced as a 'end state' when signals have been given in max 8 seconds. The period of 8 seconds is regarded as enough for the user to apprehend it).

377 SRS41114 functional Service: Regular call
Unobtainable tone shall be given in up to 8 seconds, after which the terminal shall be blocked (also see SRS41116). (Motivation: If there is no time limits f?r the different tones the hardware tone generator must be over-dimensioned. The blocking has been introduced as a 'end state' when signals have been given in max 8 seconds. The period of 8 seconds is regarded as enough for the user to apprehend it).

378 SRS41123 functional Service: Regular call
Digit dialling during an ongoing call will be ignored.

379 SRS41301 functional General
Activation of a service shall not affect the possibility to call the subscriber's number

380 SRS41304 functional General
Failure in activation or cancellation of a service shall be indicated by the system through an error tone to the subsriber in up to 8 seconds, or until the subscriber hangs up.The subscriber shall be blocked until s/he hangs up.

381 SRS41307 functional General
The operator shall have the subscriber number 1001.

382 SRS41309 functional General

Figure 2.3: The user interface of Telelogic Doors used in the experiment. The full-size color figure can be found at http://fileadmin.cs.lth.se/serg/ExperimentPackages/ReplicationReqSimile/DOORS.png

(IBM, 2010a). The user interface of Telelogic Doors is shown in Figure 2.3. The two sets of requirements were opened in Doors from separated modules and placed next to each other on the screen. Figure 2.3 illustrates one of the requirements sets opened in a module. This orientation is similar to ReqSimile's view and enables easy visual comparing between the two sets of requirements. The finding and filtering capabilities were used in Telelogic Doors to perform the consolidation task. These capabilities can be accessed respectively from the Edit menu and the Find command or the Tools menu and the Filters command. The subjects were given detailed instructions with screen-shots of each step and each dialog window that was related to finding and filtering capabilities. After finding similar requirements, links were established using the tool's built in traceability solution. During the planning activities, it was discovered that making links in Telelogic Doors is not as straightforward as in ReqSimile, where only one mouse click is required. In this case, the user has to perform two operations in order to search for or filter out desired requirements. To make a link, the user must perform two operations: initiate the link on the source side and terminate the link on the destination side. [4] The subjects received no training in using the tools other than the instruction given at the beginning of the experiment and some practice time, to get familiar with the tool.

4.6 Correct Consolidation

To enable measurement of the subjects' accuracy of linking requirements that are semantically similar, the original key for assigning correct links has been reused. This original key was created by the first author of the original experiment article (Natt och Dag et al, 2006), having many years of experience from this course in various roles. It is therefore justifiable to consider this key as one provided by an expert in the domain. The key was created a priori to any analysis of the subjects' assigned links in order to reduce any related validity threats. More information regarding the correct consolidation key, together with the distribution of the position at which the correctly similar requirements are placed by the tool in the ranked lists, is available in the original experiment article (Natt och Dag et al, 2006).

4.7 Instrumentation

In this experiment, most of the original experiment's guidelines were kept unchanged. In particular, the instructions for how to use the *assisted* method (ReqSimile tool) was reused. A new set of instructions describing how to use the *manual* method (Telelogic Doors) to find similar requirements and assign links between them, was developed and evaluated by an independent researcher. Since Telelogic Doors has a more complex user interface,

[4]The instructions for linking requirements in Telelogic Doors is available at http://fileadmin.cs.lth.se/serg/ExperimentPackages/ReplicationReqSimile/HelpSheetDoors.pdf.

the instructions were significantly longer than those for ReqSimile, consisting of eight pages of text and figures. Due to its length (8 pages), it was decided that subjects using Telelogic Doors should get more time to read through the instructions for that application. The pre- and post-test questionnaires were updated according to the changes made from the original study. In the pre-test questionnaire, the authors added one question about the experience using Telelogic Doors to be able to measure the impact of this phenomenon on the results. Furthermore, two questions related to English skills that were separated in the original design were merged into one. The rationale for this decision was that the subjects of the experiment will only read requirements so their skills in writing are not relevant. A pre-test questionnaire including five questions about the subjects' industrial experience in software development, experience with analyzing and revising requirements and possible knowledge and skills in Telelogic Doors was prepared. Before collecting the data, an additional experienced researcher evaluated the questionnaire to check the understandability of questions and their relevance for this study.

Due to a limited number of available computers in the laboratory room, the subjects were asked to work in pairs for the experiment. This deviates from the original experiment design, where subjects were performing the task individually and demands additional analysis to ensure that groups were formed equally. Some changes were also made to the post-test questionnaire. The original questions regarding (1) the time spent on the consolidation task, (2) the number of finished requirements, (3) the number of found duplicates, similar and new requirements and (4) the usefulness of used methods were kept unchanged. Moreover, two questions about the scalability of used methods and possible improvements were kept unchanged comparing to the original experiment design. [5]

4.8 Data Collection Procedure

The data collection procedure was kept as similar as possible to the original experiment design. The subjects were given the introduction and problem description by the moderator. After the introduction, subjects were given some time to read through the assigned tool's instruction and make themselves familiar with its user interface. At this stage, the groups assigned to work with Telelogic Doors were given some extra time (approximately 5-10 minutes) since the tool interface was more complex and the instruction was longer. One of the important changes here is that the subjects answered pre-study test right before starting the task. The results of the pre-study survey were analyzed afterward and are presented in Section 4.2. Next, the subjects were asked to work for 45 minutes on the consolidation

[5] Both questionnaires are available at `http://fileadmin.cs.lth.se/serg/ExperimentPackages/ReplicationReqSimile/PreTest.pdf` and http://fileadmin.cs.lth.se/serg/ExperimentPackages/ReplicationReqSimile/PostTest.pdf

task. The results from the actual experiment were collected by analyzing the information recorded in the tools about the number of links made and requirements analyzed by the subjects. The experimental results were also checked against the results of the post-questionnaire to ensure consistency. The post-questionnaire was asked after performing the task.

4.9 Validity Evaluation

As for every experiment, questions about the validity of the results must be addressed. Threats to validity are presented and discussed using the classification of threats to conclusion, internal, construct and external validity as proposed by Wohlin et al (2000).

Conclusion validity. In order to not have too low power of the statistical tests, parametric tests (t-test) were used after having investigated the normality of the data.

Subjects were selected from the same education program to limit their heterogeneity. Moreover, the questionnaire about the experience of subjects from industry, experience from the course where requirements originate from and experience in reviewing requirements was used to assess the heterogeneity of the subjects in these aspects. However, the threat related to the fact that subjects were asked to work in pairs may impact the conclusion validity. It affects in particular the random heterogeneity of subjects, since created pairs may manifest differences in industrial experience or experience from the previous courses. We discuss this threat in the analysis of the pre-study questionnaire results in Section 4.2 and the results achieved by the subject in relation to their experience in Section 7.1. The way how the subjects took seats in the laboratory room and thus the way how they were assigned to the methods can also be questioned here. As pointed out by Wilkinson et al (1999), random assignment is sometimes not feasible in terms of the control or measure of the confounding factors and other source of bias. Elements outside the experimental setting that may disturb the results were minimized. Students were not interrupted during the experiment sessions and no significant noise was present. In order to minimize random irrelevance in experimental setting, the experiment moderators ensured that any discussions in pairs of subjects should be made as quietly as possible.

Although all subjects have taken the same education program for 2.5 years, the individual differences in industrial experience, experience in the course from which the requirements originate, and knowledge of English may affect the results. The searching for a specific result threat was addressed by not notifying the subjects which method is supposed to perform better than the other. The threat to the reliability of measurements is addressed by reusing the original measurements for the replication case. Moreover, all subjects received the same instruction for using the treatments which helps to standardize the application of treatments to subjects.

Finally, the error rate of the significance level and the use of Bonferroni correction (Arcuri and Briand, 2011) are among the threats to conclusion validity. The possibility of using the Bonferroni correction to adjust the level of significance is discussed in Section 7.

Internal validity. In this case, threats related to the history, maturation etc. have to be mentioned. The history threat to internal validity is minimized by applying one treatment to one object. Both days when the experiment sessions were held where normal working days not followed by any holidays (the first session took place on Tuesday and the second session on Friday). The maturation threat was minimized by dedicating only 45 minutes for the consolidation task (we assume that the subjects won't get bored by the task in 45 minutes). The instrumentation threat is addressed in two ways: (1) by reusing the original experimentation instrumentation, if no changes were needed, and (2) reviewing the instrumentation documentation by an independent researcher. However, since subjects were not divided into groups according to the results of the pre-study questionnaire (the questionnaire has been filled in right before the experiment's execution), the statistical regression threat can not be as easily addressed as in the original experiment. The analysis related to this threat is presented in Sections 4.2 and 7.

The incentives of participants are, next to their experience, an important factor that may influence the results of this study. According to the classification presented by Höst et al (2005), both the original experiment and replication can be classified as I2 E1 where I2 means that the project is artificial (in terms of incentive). The subjects typically have no prior knowledge of the artifacts that they are working with and the requirements sets used in the experiment were developed by the researcher or borrowed from an industrial organization. The E1 level on the experience scale means that the subjects are undergraduate students with less than three months recent industrial experience, where recent means less than two years ago. Although the identical comparison of two I2E1 cases is not present in Höst et al (2005), the example of two experiments classified as E1 I1 (where I1 means an isolated artifact) shows no significant difference in their outcomes. Moreover, three other pairs of experiments, classified in the same category, also shows the same outcomes (Höst et al, 2005).

The selection threat, as in the original design, may influence the results since the subjects are not volunteers and the laboratory session where the experiment was performed is a mandatory part of the course. The social threat to internal validity is addressed since the subject had nothing to gain from the actual outcome of the experiment; the grading in the course is not based on results of, or preparation for, the experiment. Unlike the original experiment design, the experiment groups were not separated, however no information about which method is expected to perform better was revealed to the subjects. The possibility of looking at other subjects' results during the experiment execution was minimized by placing the subject in a way that separated each of the two treatments by the other treatment. Compensatory rivalry may be a problem in this case, since the group that will use the open-source solution (ReqSimile) or the commercial solution (Telelogic DOORS) may try to perform better to make their favor type of software win. This threat was addressed by explicitly stating in the beginning of the experiment that there is no favorite or assumingly better method. The potentially more problematic threat is that the subjects had to analyze and link requirements written in English when they had themselves used only Swedish to specify their own requirements in the domain. Further, the participants ability to objectively evaluate their skill in English language is a subject to question. In further work it would be interesting to execute the experiment on a set of native English subjects, also handling the original set of requirement to a native speaker experienced requirements analyst who will edit and rephrase them.

Construct validity. In this case, the theory is that the *assisted* method implemented in the ReqSimile tool provides better assistance for a particular task than the method implemented in Telelogic Doors. We base this theory on the fact that the *assisted* method provides a list of candidate requirements augmented with the degree of lexical similarity. As a result, the analyst can only look at a subset of possible candidate requirements (the most similar requirements, up to a certain threshold) thus saving time required to do the task. Moreover, we believe that the list of possible candidates helps the analyst to miss fewer requirements links and increase the precision of making the links. In contrast to the original experiment design, none of the authors have developed any of the tools. However, the originally mentioned threat related to the awareness of subjects about their own errors is still present in this experiment. This threat may have influenced the number of correct and faulty links. Also, as pointed out by Natt och Dag et al (2006), when subjects know that the time is measured, it is possible that they become more aware of the time spent and the performance results may be affected.

The level of experience of the subjects, especially in reviewing requirements, may influence the outcome of the experiment. This threat to construct validity is addressed by the analysis and discussion of the results achieved by the subjects having experience in reviewing requirements in Section 7. Because the same 30 and 160 requirements were used by all subjects in both treatments and experimental sessions, this may result in a situation where the cause construct is under-represented. This threat is discussed in Section 7 where alternative designs for this experiment are outlined. Moreover, keeping the requirements sets unchanged opens up the possibility of discussing other factors as well as differences between the original and replicated experiment to assess their influence on the results. The interaction of different treatments threat to construct validity is minimized by involving the subjects in only one study. The differences in the user interfaces and their usability may have influenced the results achieved by the subject. In particular, this threat could influence the numbers of requirements analyzed and links assigned by the subjects. This threat has been addressed in two ways: (1) by providing detailed instructions on how to make links in the tools and by giving the subjects as much time as they requested to get familiar and comfortable with the user interface, (2) by making the user interfaces look as similar as possible by placing the two requirements sets next to each other in Telelogic DOORS. Finally the evaluation apprehension threat is minimized by: (1) clearly stating that the performance of the subject has no effect on their grade in the course and (2) by using requirements that were not written by the subjects.

External validity. The largest threat in this category is the number of analyzed requirements. Since only a relatively small number of requirements was analyzed during the experiment, it is hard to generalize the results to large sets of requirements, which often is the case in industry projects (Berenbach et al, 2009; Leuser, 2009). Using students as subjects is another large threat. Even though the subjects were on their last year of studies, they can be considered as rather similar to an ordinary employee. However, as mentioned by Kitchenham et al (2002) students are the next generation of software professionals and they are relatively close to the population of interest. Since they participated in the requirements engineering course, they are familiar with the application domain. Finally, the diversity of experience of subject from industry and from analyzing and reviewing requirements, although hindering the conclusion validity, has a positive influence on the heterogeneity of the population sample used in this experiment.

The time spent on the task is also among potential threats to external validity. To analyze 30 requirements in 45 minutes subject should spend on average 90 seconds on each requirement. It remains an open question whether or not this is enough time for the subjects to perform both lexical and semantic analysis. However, this threat was partly addressed by stating at the beginning of the experiment that the subjects don't have to analyze all 30 requirements in 45 minutes (the subjects had to record how many requirements were analyzed and how they browsed the list of candidate requirements).

5 Experiment execution

The replication was run in two two-hour laboratory sessions in January 2008. The first 15 minutes of each session were dedicated to the presentation of the problem. During this presentation, the importance of the industrial applicability of the results and the goal of the experiment were stressed. All students were given the same presentation. The general overview and differences between the included methods and tools were presented without favoring one method over the other. To avoid bias-ing, no hypotheses were revealed and it was made very clear that it is not known which approach will perform better. After the presentation of the problem, students were given time to get familiar with the instructions of how to use the tools and report that they are ready to start working on the task. The starting time was recorded in the questionnaire as students required varying times to get familiar with the instructions. After the 45 minutes time assigned for the task, subjects were asked to stop, record the number of analyzed requirements, the time, and to fill in the post-questionnaire. The remaining time was used for exchanging experiences and discussion the about tools used. This approach is similar to the original experiment execution described by Natt och Dag (2006). The difference from the original experiment is that subjects used both methods in both experimental sessions. Half of the subject pairs in each session (there were two sessions in total) were assigned to the assisted method while the other half to the manual method. The subjects were to use only one method during the experiment and participate in only one of the two sessions. The difference from the original experiment here is that the methods were not separated in different sessions. That is, in each session the pairs using the *assisted* and the *manual* methods were mixed.

After the presentation, the subjects were assigned to the methods. Because only one laboratory room could be used for each session and this room did not have enough computers for all subjects (which was not known while planning the experiment), the subjects were asked to work in pairs. Each pair was randomly assigned to the method later used for the consolidation task. There were no name tags or other indicators of the method on the laboratory desks when subjects took their seats in the laboratory room. Therefore, subjects could not take a preferable method seat or be attracted by the name on the desk. Subjects were asked to discuss the solutions only within their own pair. Since the nearest group was not using the same method, the possibility of comparing or discussing results was avoided. The subjects were allowed to ask questions of the moderator, if they experiences any problems. Only answers related to the difficulties of using tools were given in a straightforward manner. No answers related to assessing similarity between requirements were given. The material used in the experiments comprised:

- The ReqSimile application with automated support of similarity calculations and a database containing: (1) 30 randomly selected requirements from the first set, (2) all 160 requirements from the second set. These requirements should be browsed through by the subjects.

- The Telelogic Doors application with the same two sets of requirements imported into two separated modules. The application's graphical user interface was set as presented in Figure 2.3 in order to make it as similar to the ReqSimile user interface as possible.

- The documentation comprising: (1) An industrial scenario describing the actual challenge (one page). (2) A general task description (one page). (3) Detailed tasks with space for noting down start and end times (one page). (4) A short FAQ with general questions and answers about the requirements (one page). (5) A screen shot of the tool user interface with the description of the different interface elements in the ReqSimile case (one page) or a 8 pages instruction with screen shots from the steps needed to analyze requirements and make links using Telelogic Doors.
- The instruction to the students was as follows: (1) Review as many of the requirements as possible from the list of 30 requirements shown in the tool. For each reviewed requirement, decide if there are any requirements in the other set that can be considered identical or very similar (or only a little different) with respect to intention. (2) Assign links between requirements that you believe are identical or very similar. (3) Note down the start and finish time. (4) When finished, notify the moderator.

Given the experience from the original study, it was decided to dedicate 45 minutes to the consolidation task. The subjects were notified about the time left for the task both 15 and five minutes before the end of the lab session. After approximately 45 minutes, subjects were asked to stop working on the task unless they, for any reason, spent less than 40 minutes on the task. All students were asked to fill in a the post-test questionnaire described in Section 4.7. Apart from noting the finishing time and the number of analyzed requirements, subjects were also asked to assess the usefulness of used methods in terms of the given task and, if applicable, propose improvements. Right after executing the experiment, it was known which data points had to be removed due to tool problems or subjects' attitude. Three groups had problems with the tools used which resulted in loss of data and one group performed unacceptably analyzing only three requirements during 45 minutes and making only two links. These four groups were treated as outliers and removed from the analysis.

6 Experiment results analysis

In this section, methods of analyzing the results are described. In order to keep the procedures as similar to the original experiment design as similar as possible, the same statistical methods were used to test if any of the null hypotheses can be rejected. Additional analysis was also performed in order to assess if working in pairs influences the performance of subjects. Hypotheses were analyzed separately, while any relations and accumulated results are presented in Section 7. The standard t-test has been used, as the data was confirmed to have a normal distribution. Just as in the original experiment from which requirements were reused, in this experiment one analyzed requirement can be linked to several others. The results from measuring dependent variables can be found in Table 2.5. Subjects that used ReqSimile are marked with the letter A (as an abbreviation of the *assisted* method), and subjects that used Telelogic Doors with the letter M (as an abbreviation of the *manual* method). The dependent variables are described in Section 4.1. Table 2.5 presents the results from both experimental sessions (Tuesday and Friday sessions).

Rows M10, M11, A7 and A8 in Table 2.5 represent data points that were removed from the analysis. The pair M10 was removed from the results due to the inconsistency between the results stated in the post-task questionnaire and the results saved in the tool. The pair M11 was removed due to loss of data. Similar problems caused the authors to remove group A8 from the analysis since the links were not saved in the tool. Finally, group A7 was removed due to their lack of their commitment to the task.

The time spent on the task is presented in column 2 of Table 2.5. The results for the number of finished requirements, derived from the post questionnaire and confirmed with the results recorded in the tool used, are listed in column 3. Next, other dependent variables values are presented in the remaining columns. The values were calculated based on the results saved in the tools and from the answers to the questionnaires questions.

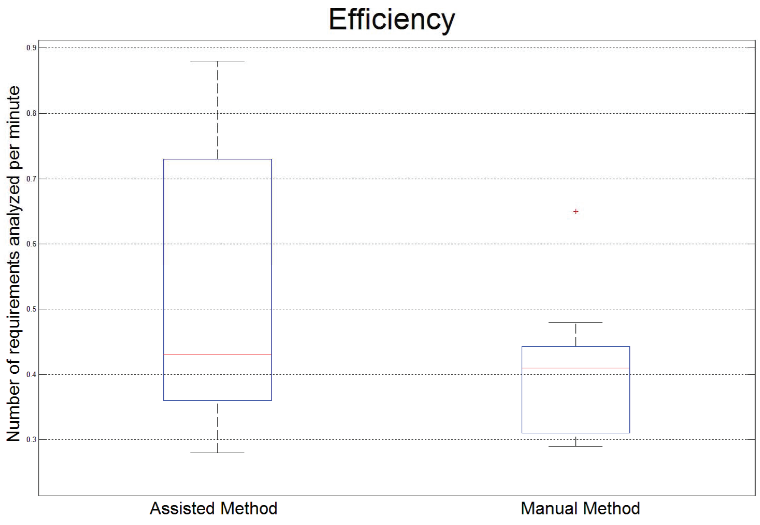


Figure 2.4: The results for the number of analyzed requirements.

The results for the number of analyzed requirements per minute are depicted as a box plot in Figure 2.4. It can be seen that there is no statistically significant difference in the number of analyzed requirements be-

Table 2.5: The results from measuring dependent variables. The rows highlighted **bold text** indicate data points that were removed from the analysis (outliers).

Pair of subjects	T (min)	N	Links assigned	Correctly linked (N_{cl})	Correctly not linked (N_{cu})	Incorrectly linked (N_{il})	Missed (N_{iu})
M1	38	12	6	1	4	5	6
M2	41	13	10	5	4	5	3
M3	46	30	15	11	10	4	9
M4	44	13	10	5	4	5	3
M5	45	18	16	8	11	8	12
M6	48	20	5	2	6	3	9
M7	45	22	21	6	6	15	8
M8	44	19	9	4	7	5	8
M9	46	19	15	7	5	8	5
M10	**45**	**16**	**9**	**4**	**4**	**5**	**5**
M11	**45**	**18**	**?**	**?**	**?**	**?**	**?**
A1	41	14	13	5	5	8	5
A2	45	20	25	9	4	16	3
A3	49	18	19	5	3	14	6
A4	45	13	25	5	0	20	3
A5	50	20	15	8	4	7	4
A6	50	21	19	6	3	13	6
A7	**29**	**3**	**11**	**1**	**0**	**12**	**0**
A8	**44**	**30**	**?**	**?**	**?**	**?**	**?**
A9	34	30	23	13	7	10	7
A10	41	30	16	12	8	4	8
A11	50	23	19	7	7	12	7
A12	35	30	20	13	8	7	7

tween the *manual* and the *assisted* method. The group that used the *manual* method analyzed on average 0.41 requirements per minute while the group that used the *assisted* method analyzed on average 0.51 requirements per minute. In this case, we observe that the medians are most likely equal, while the lower and upper quartiles values differ significantly. The t-test gave a p-value of 0.20 which gives no basis to reject the null hypothesis H_0^1. The notches of the box plot overlap. To summarize, the *assisted* method turned out not to be more efficient in consolidating two requirements sets than the *manual* method (research question Q1a).

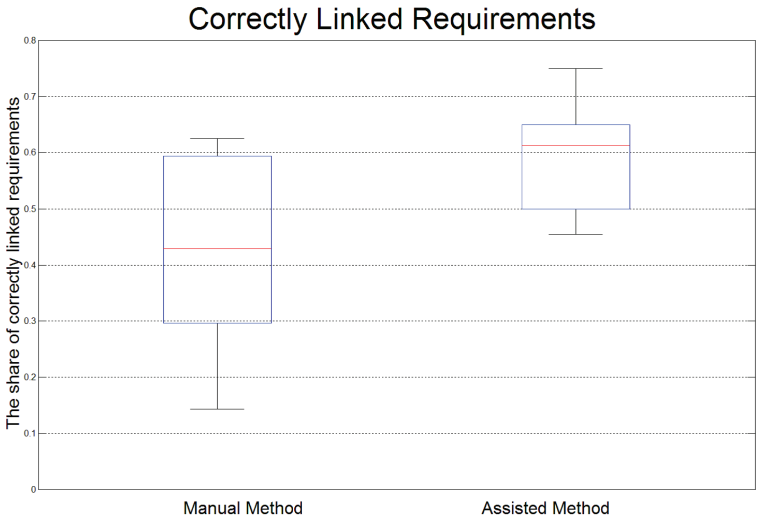


Figure 2.5: The results for the share of correctly assigned links.

The results for the number of correct links assigned by subjects are depicted in Figure 2.5. The group that used the *assisted* method correctly assigned on average 58% of the links that the expert assigned, while the group that used the *manual* method correctly assigned on average 43% of the correct links. The medians differ significantly from 61% for the *assisted* method to around 42% for the *manual* method. The t-test gave in this case the p-value 0.013 which makes it possible to reject hypothesis H_0^2. Thus, we can state a positive answer to research question Q1b, the *assisted* method is superior to the manual method when consolidating two requirements sets when measured by which method delivers more correct links.

To address hypothesis H_0^3, requirements analyzed by each pair of subjects were reviewed, and the number of links that should have been assigned but were not, was calculated. In the case when subjects did not analyze all requirements, only requirements that had been analyzed were taken into consideration. Each pair of subjects stated in their post-test

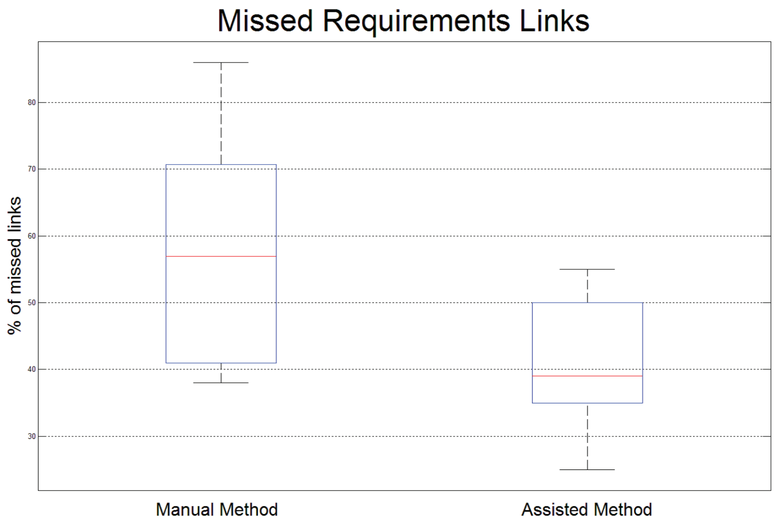


Figure 2.6: The results for the percentage of missed links.

questionnaire how many requirements were analyzed and how they had worked through the list of requirements. This information was used to correctly count the number of missed links. The results are depicted in Figure 2.5. The group that used the *assisted* method missed on average 41% of the links, while the group that used the *manual* method missed on average 57% of the links. The medians in this case are 38% for the *assisted* method and 57% for the *manual* method. The t-test gives a p-value of 0.0207 which means that we can reject H_0^3 and confirm the original experiment's conclusions by making the conjecture that the *assisted* method helps the subjects to miss significantly fewer requirements links than the *manual* method (research question Q1c).

For the number of incorrectly assigned links (N_{il}) (related to research question Q1), the t-test resulted in a p-value of 0.14, so the hypothesis H_0^4 cannot be rejected. Furthermore, for the hypothesis H_0^5 (related to research question Q1) the t-test gave the p-value 0.62 and for the hypothesis H_0^6 (related to research question Q1) the t-test resulted in the p-value 0.72. To summarize, research question Q1 can be answered as "yes", for some aspects. Our results confirm the results from the original experiment for correctness and number of missed links but we can't confirm the result for the efficiency. As in the original experiment the hypotheses H_0^4, H_0^5, H_0^6 could not be rejected here. Compared to the original experiment, this experiment confirms no statistical difference in the number of incorrect links, precision and accuracy between the two analyzed treatments. The question regarding different results for H_0^1 is discussed in Section 7. The summary of the

Table 2.6: The results of the t-tests for original and replicated experiments.

Hypotheses	The p-values in the original study (Natt och Dag et al, 2006)	The p-values in this replication
H_0^1 Efficiency	0.0034	0.20
H_0^2 Correct links	0.0047	0.013
H_0^3 Missed links	0.0047	0.02
H_0^4 Incorrect links	0.39	0.14
H_0^5 Precision	0.39	0.62
H_0^6 Accuracy	0.15	0.72

original and the replicated experiments is depicted in Table 2.6.

7 Experiment results interpretation and discussion

This section presents an interpretation of the results presented in Section 6. Since this experiment was conducted on a set of students, it is important to emphasize here that the results from this study are interpreted in the light of the population where the experiment was held (Kitchenham et al, 2002). The section discusses the results of this experiment in isolation as well as in relation to the results of the original experiment.

7.1 Interpretation of this experiment

The results achieved in this replicated experiment allow for the rejection of two out of six stated null hypotheses (see Table 2.6). As already mentioned, four data points were removed from the analysis for various reasons, as described in Section 6. The results achieved by group A7 show that the subjects were not motivated to do the task or misunderstood the task (they analyze only 3 requirements in 29 minutes). It is surprising since both subjects in this pair had one or more years of industrial experience (pairs with similar experience (A3 and M4) performed significantly better on this task), but no experience in reviewing requirements. It is an open question how they could influence the results if properly motivated. Similarly, due to unexpected tool issues (the results were not saved in the tool) we can only assume that the results achieved by groups A8 and M11 could positively influence the results for both assisted and manual method (group A8 achieved efficiency of 0.68 requirement per minute, group M11 0.4 requirement per minute and group M10 0.35 requirements per minute). Adding these three incomplete data points (A8, M10 and M11) to the analysis of the performance will not change the result of the hypothesis H_0^1 testing (although it can minimize the p-value to 0.0890).

As for the H_0^1 (performance) hypothesis (research question Q1a), the lack of a statistically significant difference can be interpreted in the following way: we have not found this property (namely lower efficiency of the manual method comparing to the assisted method) on a different requirements management tool, namely Telelogic DOORS. The lack of statistical significance can be explained by a rather large variation in the *assisted* method (the minimum value for the performance is 0.29 requirement per minute while the maximum value is 0.89 requirement per minute). Furthermore, albeit the medians are almost identical for both the *assisted* and the *manual* method with respect to the performance, the range of the third quartile is much larger in the *assisted* method. This fact can be interpreted in favor of practical significance (Kitchenham et al, 2002) in the following way: if we assume that both groups assigned to the methods are rather homogeneous, we can also assume that in both groups there are similar numbers of more and less motivated subjects. In the light of the fact that motivation has been reported to be an important determinant of productivity and quality of work in many industries (Baddoo et al, 2006), the practical significance of the results is that the *assisted* method gives the possibility to achieve higher values of the performance than the *manual* method. As more motivated subjects usually achieve better results with a given task, we can assume that the top scores for both methods correspond to the most motivated pairs of subjects. The evidence reported by Badoo et al(2006), albeit studied on developers rather than requirements engineers, confirms that the traditional motivators of software developers, e.g. intrinsic factors, but also opportunity for achievement, technically challenging work and recognition have a strong influence on the developer's performance. Thus, when comparing the top score of both methods, we could conclude that the *assisted* method may boost the performance of the motivated subjects more than less motivated subjects.

The analysis of the results for efficiency versus the experience of the subjects revealed that the subjects with experience reviewing requirements (pairs A1, A3 and A11) were not the fastest (the values were lower or around the median value). We can postulate here that the industrial experience led these pairs to be more cautious when analyzing requirements. On the other hand, the top score in this group (A9) was achieved by a pair of subjects that reported no industrial experience and no experience from the course from which the requirements originated. Surprisingly, the two lowest values of performance were achieved by the pairs having either one year of industrial experience, including experience with reviewing requirements (pair A3), or experience from the course from which the requirements originated (pair A4). In the light of these facts, we can not draw any strong conclusions about the effect of both industrial experience and experience with reviewing requirements on the performance of subjects. However, our results show indications of negative impact of experience on the performance of the subjects. The full analysis of results of experienced versus inexperienced subjects is presented later in this section and in Tables 2.7 and 2.8.

The results concerning the number of correct links (research question Q1b) can be interpreted as follows. The group that used the *assisted* method assigned on average 58% of the correct links, while the group that used the *manual* method assigned on average 43% of the correct links. The results of the t-test allows us to reject H_0^2. This fact may be interpreted in the following way in favor of the *assisted* method: even if the *assisted* method is put next to a rather sophisticated requirements management tool, it can still provide better support for assessing more correct links between requirements. The fact that both in the original and the replicated studies the *assisted* method provided a better support in linking similar requirements may lead to the following two interpretations: (1) the method is better in this matter, and (2) working in pairs has a minimum or equal impact on the two methods when it comes to the number of correctly linked requirements.

The results for the number of missed requirements links (research question Q1c) confirm the results of the original experiment. The t-test confirms that the *assisted* method can help to miss fewer requirements links than the *manual* method. Missing fewer links may be important when large sets of requirements have to be analyzed, which is a reasonable practical interpretation of this result. This result also confirms the interpretation that in the case of the *assisted* method, showing a list of similar requirements candidates limits the solution space for the analyst which results in a smaller number of missed requirements links.

Similarly to the original experiment, the results from the experiment can also not reject hypotheses H_0^4 , H_0^5 and H_0^6 (research question Q1). The lack of statistically significant differences in these cases may be interpreted as the possible existence of additional factors that affect the consolidation of requirements process which were not controlled in the experiment. For example, since it is much easier to make a link in ReqSimile than in Telelogic DOORS this may affect the number of incorrect links, precision and accuracy. This threat to construct validity is described in Section 4.9 and is considered as one of the topics for further work.

The fact that subjects worked in pairs may also influence the results. Even though working in pairs has generally been considered having a positive impact on the task, for example in pair programming (Begel and Nachiappan, 2008), the results among researchers are inconsistent (Hulkko and Abrahamsson, 2005; Parrish et al, 2004). Therefore, assessing the impact of working in pairs in more decision-oriented software engineering tasks is even more difficult. Thus, it can be assumed that working in pairs may sometimes influence the performance of these types of tasks positively, and sometimes negatively. In this case, we assume that subjects were similarly affected by this phenomenon both in the *assisted* and in the *manual* method.

The influence on the results of fluency in reading and reviewing requirements in the English language can be interpreted in the following way. Since subjects reported either "very good knowledge" or "'fluent knowledge' in reading and writing English our interpretation of this fact is that this aspect equally influenced all subjects. Moreover, the subjects were allowed to ask questions for clarification, including understanding the requirements during the experiment. However, it remains an open question what can be the results of the experiment when performed on native English language speaking subjects.

The analysis of the influence of the industrial experience of the subjects of the results achieved is depicted in Table 2.7. The data has been analyzed for all pairs of subjects, as well as for subjects using the same method. Four out of the total 11 pairs of subjects using the *assisted* method reported having some industrial experience. For the *manual* method, 5 out of 9 pairs of subjects included in the data analysis reported having some industrial experience. Subjects were paired in a way that minimizes the difference in the experience of pair members. Moreover, only in two cases (pairs M5 and M6) were pairs formed of one experienced and one inexperienced subject (see section 4.2 for more detail about the subjects).

The analysis of the relationship of industrial experience to the results is based on the arithmetic average values of the results achieved. Table 2.7 shows that in most cases industrial experience negatively influenced the results (and tested hypotheses). The cells colored bold in Table 2.7 indicate cases where industrial experience has a positive effect on the analyzed aspect. For all hypotheses for subjects using the *manual* method, the industrial experience had a negative impact on the results achieved. In the case of the *assisted* method, the experienced subjects made fewer incorrect links and had better precision and accuracy. The results from comparing all subjects show the same pattern as the results for the *assisted* method. While the results are implausible, they may be an indicator that general software engineering industrial experience may not be useful in the task of analyzing requirements, at least when the experience is minimal to small. Thus, we state a hypothesis that experience in reviewing requirements and the domain knowledge should significantly help in achieving better results by our subject. Keeping in mind the scarceness of the data, we provide some examples that we used to support the hypothesis in the text that follows.

Six pairs of subjects reported having experience analyzing and reviewing requirements outside of the courses taken in their education. In one case (M8), a person with more than one year of industrial experience was paired with a person with no experience. Although it may be expected that an experienced pair member can significantly influence the results of this

Table 2.7: The analysis of the industrial experience of the subjects in relation to their results. The values in **bold text** indicate cases where industrial experience had positive effect on the analyzed aspect.

	All data		Assisted method		Manual method	
	Exp.	Unexp.	Exp.	Unexp.	Exp.	Unexp.
Efficiency H_0^1 **[N/T]**	0.44	0.48	0.51	0.52	0.40	0.42
Correct H_0^2 **[%]**	44	55	53	61	39	47
Missed H_0^3 **[%]**	55	44	46	38	60	52
Incorrect H_0^4 **[%]**	**39**	**54**	**40**	**66**	38	33
Precision H_0^5 **[%]**	**45**	**43**	**50**	**41**	42	46
Accuracy H_0^6 **[%]**	**46**	**43**	**49**	**40**	44	48

pair, the efficiency of this pair was only 0.02 higher than the median value of the efficiency for the subjects using the *manual* method. The number of correct links is 10% lower, the number of incorrect links is 10% higher while precision and accuracy are very close to the average values for the entire group using the *manual* method. In the case of pair M7, a person with experience only from courses was paired with a person with less than a year of industrial experience in analyzing and reviewing requirements. The results achieved by this pair are higher than the average in terms of efficiency (7% higher), close to the average for the share of correct links, and below the average for the remaining attributes (30% more incorrect links than the average, 16% lower than the average for the precision and 12% lower than the average for the accuracy). The remaining 3 pairs of subjects (A1 and M5) were composed of a person with only academic experience with a person with less than a year of industrial experience. Pair M5 analyzed 0.4 requirement per minute which is close to the average value (0.41), achieved 40% of correct links (the average value is 43% for this group) and 44% of incorrect links. The precision achieved by pair M5 is 50% which is 4% higher than the average. When it comes to the accuracy, pair M5 achieved 46% accuracy (the average value was 46%). The results for efficiency for pairs (A1, A3 and A11) were below or about the average values, and these data points could have been partly responsible for the fact that hypothesis H_0^1 could not be rejected. The results for these pairs for the number of correct links and the share of missed links were also below the average and the median values. The results for the number of incorrect links were around the mean value and above the median value. Finally, the results for the precision and accuracy were below the median values. To summarize, the influence of experience in analyzing and reviewing requirements can't be clearly defined and statistically confirmed, as subjects report both results above and below the average values.

As the last step of the analysis, we investigate whether prior experience in the course that originated the requirements in some manner influences the results achieved by the subjects. We can assume that this experience can somehow be compared to specific domain knowledge. In the course model, all team members are actively involved in analyzing and reviewing requirements. Thus, we only distinguish between subjects that took and did not take the course. For six pairs of subjects, both subjects have experience from the course, for seven other pairs only one pair member had experience from the course. Finally for six pairs, both subjects reported no experience from the course. The pairs where both members had experience and where both members had no experience were equally distributed between the methods (three pairs for each method). The *assisted* method had four pairs with experience and lack of experience. The analysis is depicted in Table 2.8. Pairs where both pair members had experience from the course are abbreviated with the letter "E", where only one pair member had experience are abbreviated with "E and U" and where none of the two pair members had any experienced from the course is abbreviated with the letter "U".

Analyzing the differences between the average results for all three subgroups for both methods we can see that the expected behavior can only be confirmed for the number of correct links. Pairs with experience in the course achieved better correctness than inexperienced pairs, independent of whether one or both members had the experience. Another interesting observation here is that inexperienced subjects missed on average only 34% of requirements links, while experienced subjects respectively missed 50% (both pair members experienced) and 57% (when one of the pair members had some experience). The lowest average precision and accuracy levels were recorded for pairs where one pair member had experience from the course from which the requirements were taken. The analysis of the pairs working with the same method confirms the analysis of all data points. For the *manual* method experienced pairs turned out to be more correct than the inexperienced pairs. For the *assisted* method, pairs where both members where experienced were more correct, but pairs where only one member had experience where not as correct as the inexperienced pairs. Finally, the pairs where only one person was experienced performed worse in all aspects for both methods analyzed than the pairs where both persons were experienced.

7.2 Interpretation of the results from both experiments

In this section, we provide the analysis and discussion of the results achieved in both the original and the replicated experiments. We have used standard t-tests to test if there are any significant differences between the two experiments. From the results of the t-tests between the same methods depicted in Table 2.9, we can see no significant difference for any of the cases (research question Q2). However, some interesting differences between the two experiments for the efficiency of the subjects using the *assisted* method

Table 2.8: The analysis of the experience from the course where requirements originate from. *E* denoted that both pair members are experienced, *E & U* denoted one experienced and inexperienced person working together and *U* denoted that both pair members were inexperienced.

	All data			Assisted method			Manual method		
	E	E & U	U	E	E & U	U	E	E & U	U
Efficiency H_0^1 **[N/T]**	0.45	0.39	0.57	0.53	0.38	0.69	0.37	0.40	0.46
Correct H_0^2 **[%]**	**56**	**51**	**46**	68	53	58	46	48	34
Missed H_0^3 **[%]**	50	57	34	56	59	31	44	55	37
Incorrect H_0^4 **[%]**	58	54	30	85	58	56	31	50	27
Precision H_0^5 **[%]**	42	39	50	40	37	56	45	42	44
Accuracy H_0^6 **[%]**	43	41	49	39	39	53	47	45	46

can be seen from the box-plot visualization in Figure 2.7. As can be seen in Figure 2.7, the results for the efficiency of the *assisted* method in this experiment have a much larger range of values, which may be the reason why the hypothesis H_0^1 could not be rejected (research question Q2a). As described in Section 7.1 the two lowest values of performance were achieved by the pairs having either one year of industrial experience, including experience from reviewing requirements (pair A3), or experience from the course from which the requirements originate (pair A4).

However, one of the possible explanations for the difference between the performance in this experiment and in the original experiment differs may be that more advanced searching and filtering functionalities have been used in the current *manual* method. Contrary to the original experiment, the *manual* method in this experiment uses advanced searching and filtering functionalities which may (to some extent) be comparable to the lexical similarity analysis because they also present only a subset of analyzed requirements to the analyst. The analyst using the filtering and searching functionality has to provide a meaningful search string to filter out similar requirements, while in the lexical similarity case the analysis is done automatically. Our interpretation is supported by Figure 2.8 which depicts the results of the performance for the *manual* method between the original and the replicated experiments. The median value for this replication study is 0.41 and is 0.05 higher than in the original experiment (0.36). However, the second quartile has more diverse values than in the original experiment. Moreover, we can assume that the filtering method has a higher degree of uncertainty which is shown by the results for the accuracy.

7.3 Discussion

In this section, we discuss alternative designs for this experiment as well as the differences between the two experiment sessions (the replication has

Table 2.9: The results of the t-tests for the original and the replicated experiments for the same methods.

Hypotheses	Assisted old/new (p-value) (research question Q2a)	Manual old/new (p-value) (research question Q2b)
H_0^1 Efficiency	0.48	0.27
H_0^2 Correct links	0.93	0.30
H_0^3 Missed links	0.37	0.20
H_0^4 Incorrect links	0.21	0.73
H_0^5 Precision	0.81	0.45
H_0^6 Accuracy	0.90	0.41

been run in two sessions). Using alternative design could be beneficial for the comparative analysis of the original and the replicated experiment (RQ2). For example the paired t-test could have been used to support comparison between this replication study and the original study (Kachigan, 1991). However, the it remains an open question if paired units are similar with respect to "noise factors" for both the *assisted* and the *manual* methods used. It could also have been beneficial to the validity if the subjects answered the pre-questionnaire before running the study and were then assigned to treatments based on the result of this questionnaire. The manual analysis of the two experiment sessions did not reveal any significant differences between the Tuesday and the Friday sessions. However, to fully address this threat to validity, additional statistical tests should be used. Finally, changing the design of the study to use random samples of 30 and 160 requirements for each subject, generated from a much large dataset of requirements is one of the options for further work.

During this experiment six hypotheses were tested using the same data set, and more tests were performed comparing the data from the original and replicated experiments. The result of performing multiple comparisons on the same data is increased probability of Type I error, which in case of only one comparison is equal to the obtained p-value (Arcuri and Briand, 2011). Thus, the Bonferroni correction should be discussed here. In this case, we performed 18 tests, 6 tests comparing the assisted and manual method (to answer the research question Q1), 6 tests comparing the old/new experiments with regard the assisted method (to answer the research question Q2a) and 6 tests comparing the old/new experiments with regard to the manual method (to answer the research question Q2b). This yields a significance level of 0.05/18 = 0.0027 according to Bonferroni. In this case it is no longer possible to reject hypotheses H_0^2 and H_0^3. The correction has no impact on the results of the tests between the original and

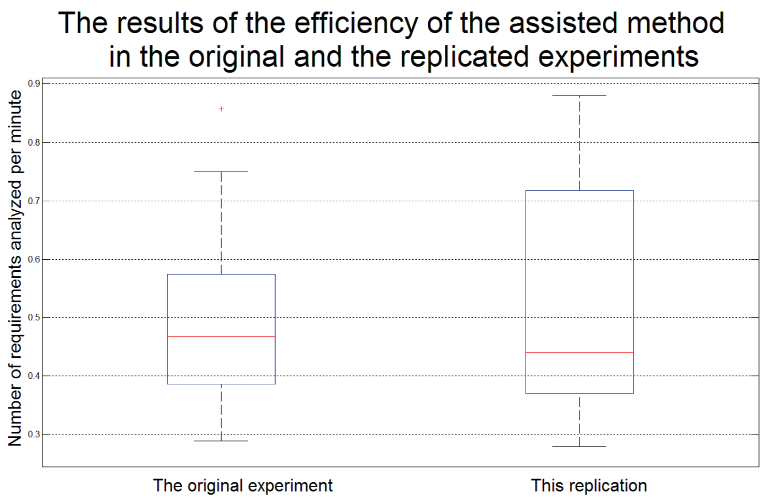


Figure 2.7: The result of comparing the efficiency of the *assisted* method in two experiment sessions.

the replicated experiments. However, the correction has not been used in the original experiment and has been criticized by a number of authors (Arcuri and Briand, 2011; Perneger, 1998; Nakagawa, 2004) where some of them do not recommend using the Bonferroni adjustment (Arcuri and Briand, 2011). In the light of this criticism, it is an open question for this work as to whether or not this correction should be used. Therefore, we report the obtained p-values for all performed tests in case the readers want to evaluate the results using the Bonferroni correction or other adjustment techniques (Arcuri and Briand, 2011).

The relationship between the efficiency and the practical experience of the subjects may have been investigated using multivariate analysis. For example, understanding if the efficiency of the subjects was related to their accuracy could have been investigated by recording the efficiency of linking requirements at random as a reference point. Since this has not been done, we consider this analysis as possible future work and thus outside the scope of this article.

8 Conclusions

Large market-driven software companies face new challenges that emerge due to their extensive growth. Among those challenges, a need for efficient methods to analyze large numbers of requirements, issued by various customers and other stakeholders, has emerged (Regnell and Brinkkemper,

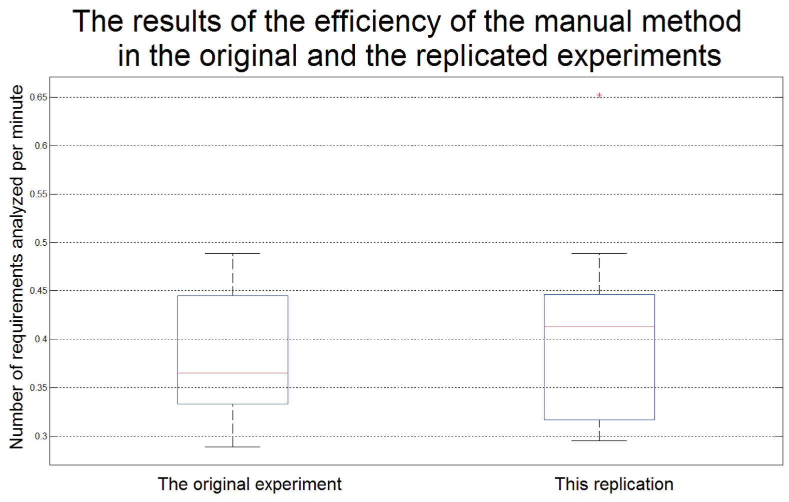


Figure 2.8: The results of the efficiency achieved by the manual method in the original and the replicated experiments.

2005). The result is an increasing effort dedicated to analyzing incoming requirements against those requirements already analyzed or implemented. This task is also called requirements consolidation. The core of the requirements consolidation process is finding the similarities between requirements and recording them by making links between them (Natt och Dag et al, 2006).

In this paper, we present a replicated experiment that aims to assess whether a linguistic method supports the requirements consolidation task better than a searching and filtering method. In this experiment, two methods implemented in two different tools were compared for the requirements consolidation task. The *assisted* method, which utilizes natural language processing algorithms to provide a similarity list for each analyzed requirements, was compared with the *manual* method, which utilizes searching and filtering algorithms to find similar requirements. After deciding which requirements were similar, the subjects assigned links between the requirements. The conclusions of this paper are as follows:

- Subjects using the *assisted* method were statistically not more efficient in consolidating requirements than the subjects using the *manual* method (research question Q1a), which is a different result compared to the original study

- The *assisted* method was confirmed as significantly more correct in consolidating requirements than the *manual* method (the manual method was changed from the original experiment) (research question Q1b), which is inline with the original study.

- The *assisted* method helps to miss fewer requirements links than the *manual* method (research question Q1c), which is the same result as in the original study.

- The hypotheses that could not be rejected in the original study (in terms of the number of incorrect links, precision and accuracy related to research question Q1) could also not be rejected in this experiment. Further investigation is required to understand the reasons for these results.

- The analysis of the results achieved for the same method (assisted or manual) between the original and the replicated study (research question Q2) shows no significant difference in any of the cases. However, some differences in favor of the searching and filtering method have been observed between the results of the performance of the subjects using the *manual* methods.

To summarize, for two of our hypotheses the results reported in this replication study confirm the results achieved in the original experiment. The first confirmed hypothesis (H_0^2) is that the *assisted* method helps to make more correct links than the *manual* method. The second confirmed hypothesis (H_0^3) indicates that the *assisted* method helps to miss fewer requirements links than the *manual* method. The statistical significance in performance of the *assisted* method achieved over the *manual* method (hypothesis H_0^1) is not confirmed in this study. The remaining three hypotheses regarding the number of incorrect links (hypothesis H_0^4), precision (hypothesis H_0^5) and accuracy (hypothesis H_0^6) could not be rejected, which is the same situation as reported in the original experiment (Natt och Dag et al, 2006). In order to investigate the possible reasons for the difference in the remaining case, this paper provides a cross-case analysis of the same methods across the two experiment sessions as well as detailed analysis of the relations between the experience of the subjects and their results.

The analysis revealed that the pairs of subjects with experience in the course that originated the requirements achieved better correctness than inexperienced pairs, independent of whether one of both members had the experience. At the same time, the pairs of subjects without any experience missed on average fewer requirements links than the experienced pairs of subjects. Furthermore, the pairs where only one person had experience performed worse in all aspects than the pairs where both persons were experienced. The analysis revealed no statistical difference for any of the cases (which refers to the research question RQ2). However, the analysis of the efficiency of the subjects using the *assisted* method in the two experiments, depicted in Figure 2.7, revealed that the values for the efficiency achieved in this replication have a much higher range. The results for the efficiency of the manual methods in the two experiments, depicted in Figure 2.8, shows similar range of values, but different medians. It should be noted that there are validity threats to the study as described in Section 4.9, e. g. experience of the subjects and their incentives, the number of subjects participating, requirements used in the experiment, and the multiple comparison threats.

Performing a third experiment with experienced practitioners and requirements sets from industry is left for future work. Moreover, it would be interesting to further investigate the influence of working in pairs on the requirements consolidation task as well as to analyze the influence of the construction of the user interfaces on the efficiency of correctness of the subjects.

Acknowledgments

This work is supported by VINNOVA (Swedish Agency for Innovation Systems) within the UPITER project. Special acknowledgments to Richard Berntsson-Svensson for participating in the pilot study and reviewing the paper. We are also thankful to Lars Nilsson and David Callele for reviewing the paper and excellent language comments.

Bibliography

Aguilera C, Berry D (1991) The use of a repeated phrase finder in requirements extraction. J Syst Softw 13:209–230

Antoniol G, Canfora G, Casazza G, De Lucia A, Merlo E (2002) Recovering traceability links between code and documentation. Software Engineering, IEEE Transactions on 28(10):970 – 983

Arcuri A, Briand L (2011) A practical guide for using statistical tests to assess randomized algorithms in software engineering. In: Proceeding of the 33rd international conference on Software engineering, ACM, New York, NY, USA, ICSE '11, pp 1–10

Baddoo N, Hall T, Jagielska D (2006) Software developer motivation in a high maturity company: a case study. Software Process: Improvement and Practice" pp 219–228

Basili V, Shull F, Lanubile F (1999) Building knowledge through families of experiments. IEEE Transactions on Software Engineering 25(4):456 –473

Begel A, Nachiappan N (2008) Pair programming: what's in it for me? In: Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement: ESEM '08, ACM, New York, USA, pp 120–128

Berenbach B, Paulish DJ, Kazmeier J, Rudorfer A (2009) Software & Systems Requirements Engineering: In Practice. Pearson Education Inc.

Breaux T (2009) Exercising due diligence in legal requirements acquisition: A tool-supported, frame-based approach. Atlanta, GA, United states, pp 225 – 230

Cleland-Huang J, Chang CK, Ge Y (2002) Supporting event based traceability through high-level recognition of change events. In: COMPSAC '02: Proceedings of the 26th International Computer Software and Applications Conference on Prolonging Software Life: Development and Redevelopment, IEEE Computer Society, Washington, DC, USA, pp 595–602

Cleland-Huang J, Settimi R, Duan C, Zou X (2005) Utilizing supporting evidence to improve dynamic requirements traceability. In: Proceedings of the 13th IEEE International Conference on Requirements Engineer (RE 2005), pp 135–144

Cleland-Huang J, Berenbach B, Clark S, Settimi R, Romanova E (2007) Best practices for automated traceability. Computer 40(6):27–35

Cleland-Huang J, Czauderna A, Gibiec M, Emenecker J (2010) A machine learning approach for tracing regulatory codes to product specific requirements. In: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering,ICSE '10, ACM, New York, NY, USA, pp 155–164

De Lucia A, Fasano F, Oliveto R, Tortora G (2007) Recovering traceability links in software artifact management systems using information retrieval methods. ACM Trans Softw Eng Methodol 16(4):13

ECTS (2010) The ECTS grading systems defined by European Commission. `http://en.wikipedia.org/wiki/ECTS\_grading\_scale`, accessed 07/09/2011

Fabbrini F, Fusani M, Gnesi S, Lami G (2001) An automatic quality evaluation for natural language requirements. In: Proceedings of the 7th International Workshop on Requirements Engineering Foundation for Software Quality (REFSQ 2001), pp 4–5

Fantechi A, Gnesi S, Lami G, Maccari A (2003) Applications of linguistic techniques for use case analysis. Requir Eng 8(3):161–170

Fricker S, Gorschek T, Byman C, Schmidle A (2010) Handshaking with implementation proposals: Negotiating requirements understanding. IEEE Software 27:72–80

Gacitua R, Sawyer P, Gervasi V (2010) On the effectiveness of abstraction identification in requirements engineering. In: Requirements Engineering Conference (RE), 2010 18th IEEE International, pp 5 –14

Gervasi V (1999) Environment support for requirements writing and analysis. PhD thesis, University of Pisa

Gervasi V, Nuseibeh B (2000) Lightweight validation of natural language requirements: A case study. In: Proceedings Fourth International Conference on Requirements Engineering. ICRE 2000, IEEE Comput. Soc, pp 113–133

Goldin L, Berry D (1997) Abstfinder, a prototype natural language text abstraction finder for use in requirements elicitation. Autom Softw Eng 4:375–412

Gorschek T, Garre P, Larsson S, Wohlin C (2007) Industry evaluation of the requirements traction model. Requirements Engineering 12(3):163–190

Gotel O, Finkelstein A (1994) An analysis of the requirements traceability problem. In: Proceedings of the First International Conference on Requirements Engineering, RE'94, pp 94 –101

Hayes J, Dekhtyar A, Osborne J (2003) Improving requirements tracing via information retrieval. In: Proceedings of the 11th IEEE International Requirements Engineering Conference (RE'03), pp 138 – 147

Higgins S, Laat M, Gieles P, Geurts E (2003) Managing requirements for medical it products. IEEE Software 20(1):26-33

Höst M, Wohlin C, Thelin T (2005) Experimental context classification: Incentives and experience of subjects. In: Proceedings of the 27:th International Conference on Software Engineering (ICSE), pp 470–478

Huffman Hayes J, Dekhtyar A, Sundaram S (2006) Advancing candidate link generation for requirements tracing: The study of methods. IEEE Trans Softw Eng 32(1):4–19

Huffman Hayes J, Dekhtyar A, Sundaram S, Holbrook E, Vadlamudi S, April A (2007) Requirements tracing on target (retro): improving software maintenance through traceability recovery. Innovations in Systems and Software Engineering 3:193–202

Hulkko H, Abrahamsson P (2005) A multiple case study on the impact of pair programming on product quality. In: Proceedings of the 27:th International Conference on Software Engineering ICSE '05, ACM, New York, NY, USA, pp 495–504

IBM (2010a) Rational doors (former telelogic doors) product description. `http://www-01.ibm.com/software/awdtools/doors/productline/`, accessed 07/09/2011

IBM (2010b) Rational doors product description (former telelogic doors), accessed 15/04/2010. `http://www-01.ibm.com/software/awdtools/doors/productline/`, accessed 07/09/2011

IEEE (2010) The ieee keyword taxonomy webpage. `http://www.computer.org/mc/keywords/software.htm`, accessed 07/09/2011

Ivarsson M, Gorschek T (2009) Technology transfer decision support in requirements engineering research: a systematic review of REj. Requir Eng 14(3):155–175

Jackson P, Moulinier I (2002) Natural language processing for online applications. Text retrieval, extraction and categorization, Natural Language Processing, vol 5. Benjamins, Amsterdam, Philadelphia

Jarke M (1998) Requirements tracing. Commun ACM 41:32–36

Kachigan S (1991) Multivariate statistical analysis: A conceptual introduction. Radius Press

Kamsties E, Berry D, Paech B (2001) Detecting ambiguities in requirements documents using inspections. In: Proceedings of the First Workshop on Inspection in Software Engineering (WISE 2001), pp 68–80

Karlsson L, sa G Dahlstedt A, Natt Och Dag J, Regnell B, Persson A (2002) Challenges in market-driven requirements engineering - an industrial interview study. In: Proceedings of the Eighth International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ 2002)

Kitchenham B, Pfleeger S, Pickard L, Jones P, Hoaglin D, Emam E, Rosenberg J (2002) Preliminary guidelines for empirical research in software engineering. IEEE Trans Softw Eng 28(8):721–734

Konrad S, Gall M (2008) Requirements engineering in the development of large-scale systems. In: Proceedings of the 16th International Requirements Engineering Conference (RE 2008), pp 217–222

Kotonya G, Sommerville I (1998) Requirements Engineering. John Wiley & Sons

Lauesen S (2002) Software Requirements – Styles and Techniques. Addison–Wesley

Leuser J (2009) Challenges for semi-automatic trace recovery in the automotive domain. In: Proceedings of the 2009 ICSE Workshop on Traceability in Emerging Forms of Software Engineering TEFSE'09, IEEE Computer Society, Washington, DC, USA, pp 31–35

Lin J, Chou Lin C, Cleland-Huang J, Settimi R, Amaya J, Bedford G, Berenbach B, Khadra O, Duan C, Zou X (2006) Poirot: A distributed tool supporting enterprise-wide automated traceability. In: Requirements Engineering, 14th IEEE International Conference, pp 363 –364

Lormans M, van Deursen A (2006) Can lsi help reconstructing requirements traceability in design and test? In: Proceedings of the Conference on Software Maintenance and Reengineering CSMR'06, IEEE Computer Society, Washington, DC, USA, pp 47–56

Lund University (2011a) The requirements engineering course (ets170) at the lund university. `http://www.cs.lth.se/ETS170/`, accessed 07.09.2011

Lund University (2011b) The software development of large systems course page at lund university. `http://cs.lth.se/etsn05/`, accessed 07.09.2011

Macias B, Pulman S (1995) A method for controlling the production of specifications in natural language. Comput J 48(4):310–318

Manning C, Schütze H (2002) Foundations of Statistical Natural Language Processing. MIT Press

Marcus A, Maletic J (2003) Recovering documentation-to-source-code traceability links using latent semantic indexing. In: Proceedings of the 25th International Conference on Software Engineering ICSE'03, IEEE Computer Society, Washington, DC, USA, pp 125–135

Mich L, Mylopoulos J, Nicola Z (2002) Improving the quality of conceptual models with nlp tools: An experiment. Tech. rep., University of Trento

Nakagawa S (2004) A farewell to bonferroni: the problems of low statistical power and publication bias. Behavioral Ecology 15(6):1044–1045

Natt och Dag J (2006a) Managing natural language requirements in large-scale software development. PhD thesis, Lund University, Sweden

Natt och Dag J (2006b) The reqsimile tool website. `http://reqsimile.sourceforge.net/`, accessed 07/09/2011

Natt och Dag J, Gervasi V, Brinkkemper S, Regnell B (2004) Speeding up requirements management in a product software company: Linking customer wishes to product requirements through linguistic engineering. In: Proceedings of the 12th International Requirements Engineering Conference (RE 2004), pp 283–294

Natt och Dag J, Thelin T, Regnell B (2006) An experiment on linguistic tool support for consolidation of requirements from multiple sources in market-driven product development. Empirical Software Engineering 11(2):303–329

Panis M (2010) Sucessful deployment of requirements traceability in a commercial engineering organization ... really. In: Proceedings of the 18th IEEE International Requirements Engineering Conference, pp 303–307

Parrish A, Smith R, Hale D, Hale J (2004) A field study of developer pairs: Productivity impacts and implications. IEEE Softw 21(5):76–79

Perneger T (1998) What's wrong with Bonferroni adjustments, vol 316

Pohl K, Bockle G, van der Linden F (2005) Software Product Line Engineering: Foundations, Principles and Techniques. Springer

Ramesh B, Jarke M (2001) Toward reference models for requirements traceability. IEEE Trans Softw Eng 27(1):58–93

Ramesh B, Powers T, Stubbs C, Edwards M (1995) Implementing requirements traceability: a case study. In: Proceedings of the Second IEEE International Symposium on Requirements Engineering RE'95, IEEE Computer Society, Washington, DC, USA, p 89

Rayson P, Emmet L, Garside R, Sawyer P (2001) The revere project: Experiments with the application of probabilistic NLP to systems engineering. In: Natural Language Processing and Information Systems, Springer, Lecture Notes in Computer Science, vol 1959, pp 288–300

Regnell B, Brinkkemper S (2005) Engineering and Managing Software Requirements, Springer, chap Market–Driven Requirements Engineering for Software Products, pp 287–308

Regnell B, Beremark P, Eklundh O (1998) A market-driven requirements engineering process: Results from an industrial process improvement programme. Requirements Engineering 3(2):121–129

Rolland C, Proix C (1992) A natural language approach for requirements engineering. In: Advanced Information Systems Engineering, Springer Berlin / Heidelberg, Lecture Notes in Computer Science, vol 593, pp 257–277

Rupp C (2000) Linguistic methods of requirements engineering (NLP). In: Proceedings of the EuroSPI 2000, pp 68–80

Ryan K (1993) The role of natural language in requirements engineering. In: Proceedings of the IEEE International Symposium on Requirements Engineering, San Diego California, IEEE Computer Society Press, pp 240–242

Samarasinghe R, Nishantha G, Shutto N (2009) Total traceability system: A sustainable approach for food traceability in smes. In: Industrial and Information Systems (ICIIS), 2009 International Conference on, pp 74 –79

Sawyer P, Cosh K (2004) Supporting measur-driven analysis using nlp tools. In: Proceedings of the 10th International Workshop on Requirements Engineering: Foundations of Software Quality (REFSQ 2004), pp 137–142

Sawyer P, Rayson P, Garside R (2002) REVERE: Support for requirements synthesis from documents. Inf Syst Front 4(3):343–353

Sawyer P, Rayson P, Cosh K (2005) Shallow knowledge as an aid to deep understanding in early phase requirements engineering. IEEE Trans Softw Eng 31(11):969–981

Shull F, Carver J, Vegas S, Juristo N (2008) The role of replications in empirical software engineering. Empir Software Eng 13(2):211–218

Sjøberg D, Hannay J, Hansen O, Karahasanovic V, Liborg A, Rekdal N (2005) The survey of controlled experiments in software engineering. IEEE Trans Softw Eng 31(9):733–753

Sommerville I, Sawyer P, Sawyer P (1997) Viewpoints: principles, problems and a practical approach to requirements engineering. Ann Softw Eng 3:101–130

Strens M, Sugden R (1996) Change analysis: a step towards meeting the challenge of changing requirements. In: Engineering of Computer-Based Systems,1996. Proceedings., IEEE Symposium and Workshop on, pp 278 –283

Wiegers K (2003) Microsoft Press

Wilkinson L (1999) Statistical methods in psychology journals: Guidelines and explanations. American Psychologist 54(8):594–604

Wilson W, Rosenberg L, Hyatt L (1997) In: Proceedings of the 19th international conference on Software engineering ICSE'97, ACM, New York, NY, USA, pp 161–171

Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslen A (2000) Experimentation in Software Engineering An Introduction. Kluwer Academic Publishers

Zowghi D, Offen R (1997) A logical framework for modeling and reasoning about the evolution of requirements. In: Proceedings of the Third IEEE International Symposium on Requirements Engineering, pp 247 –257

Paper III

Obsolete Software Requirements

Krzysztof Wnuk[1], Tony Gorschek[2], Showayb Zahda[2]
[1]Department of Computer Science,
Lund University, Sweden
{Krzysztof.Wnuk}@cs.lth.se
[2]School of Computing Software Engineering Research Lab,
Blekinge Institute of Technology,
SE-371 79 Karlskrona, Sweden
Tony.Gorschek@bth.se, shzc10@student.bth.se

under revision for
Information and Software Technology 2012

ABSTRACT

[Context] Coping with rapid requirements change is crucial for staying competitive in the software business. Frequently changing customer needs and fierce competition are typical drivers of rapid requirements evolution resulting in requirements obsolescence even before project completion. [Objective] Although the obsolete requirements phenomenon and the implications of not addressing them are known, there is a lack of empirical research dedicated to understanding the nature of obsolete software requirements and their role in requirements management. [Method] In this paper, we report results from an empirical investigation with 219 respondents aimed at investigating the phenomenon of obsolete software requirements. [Results] Our results contain, but are not limited to, defining the phenomenon of obsolete software requirements, investigating how they are handled in industry today and their potential impact. [Conclusion] We conclude that obsolete software requirements constitute a significant challenge for companies developing software intensive products, in particular in large projects, and that companies rarely have processes for handling obsolete software requirements. Further, our results call for future research in creating automated methods for obsolete software requirements identification and management, methods that could enable efficient obsolete software requirements management in large projects.

1 Introduction

Software, as a business, is a demanding environment where a growing number of users, rapid introduction of new technologies, and fierce competition are inevitable (DeBellis and Haapala, 1995; Regnell and Brinkkemper, 2005; Gorschek et al, 2010). This rapidly changing business environment is challenging traditional Requirements Engineering (RE) approaches (Ramesh et al, 2010; Gorschek et al, 2007a,b). The major challenges in this environment are high volatility and quick evolution of requirements, requirements that often tend to become obsolete even before project completion (DeBellis and Haapala, 1995; Cao and Ramesh, 2008; Wnuk et al, 2009; Hood et al, 2008b). At the same time the product release time is crucial (Chen et al, 2005; Wohlin et al, 1995; Sawyer, 2000) for the success of the software products, especially in emerging or rapidly changing markets (Chen et al, 2005).

Coping with rapid requirements change is crucial as time-to-market pressures often make early pre-defined requirements specifications inappropriate almost immediately after their creation (Cao and Ramesh, 2008). In Market-Driven Requirements Engineering (MDRE), the pace of incoming requirements (Regnell and Brinkkemper, 2005) and requirements change is high. Software companies have to identify which requirements are obsolete or outdated. The rapid identification and handling of potentially obsolete requirements is important as large volumes of degrading requirements threatens effective requirements management. In extreme cases, obsolete requirements could dramatically extend project timelines, increase the total cost of the project or even cause project failure; and even the successful identification of the obsolete requirements without handling adds little or no product value (Murphy and Rooney, 2006; Stephen et al, 2011; Merola, 2006). Thus, the identification, handling, and removal of obsolete requirements is crucial.

The phenomenon of obsolete requirements and the implications of not handling them are known (Hood et al, 2008c; Murphy and Rooney, 2006; Stephen et al, 2011; Savolainen et al, 2005; Loesch and Ploederoeder, 2007; Mannion et al, 2000; Herald et al, 2009). At the same time, several researchers focused on topics related to the phenomenon of obsolete requirements, e.g. requirements volatility and scope creep (Robertson and Robertson, 1999; Iacovou and Dexter, 2004; DeMarco and Lister, 2003; Houston et al, 2001; Gulk and Verhoef, 2008; Zowghi and Nurmuliani, 2002; Loconsole and Borstler, 2005). However, very little research has been performed into obsolete requirements management or guidelines, see e.g. (Wiegers, 2003; Lauesen, 2002; Kotonya and Sommerville, 1998; Sommerville and Sawyer, 1997; Lamsweerde, 2009; Aurum and Wohlin, 2005a). Standards (IEEE, 1997; Institute, 2011) do not explicitly mention the phenomenon of Obsolete Software Requirements (OSRs). The term itself is only partly defined and empirically anchored (Savolainen et al, 2005).

In this paper, we present the results from an empirical study, based on a survey with 219 respondents from different companies. The survey investigated the phenomenon of obsolete requirements and included, an effort to define the phenomenon based on the perceptions of industry practitioners. The study also aimed to collect data on how obsolete requirements are perceived, how they impact industry, and how they are handled in industry today.

This paper is structured as follows: section 2 provides the background and related work, section 3 describes the research methodology, section 4 describes and discusses the results of the study, and section 5 concludes the paper.

2 Background and Related Work

Requirements management, as an integral part of requirements engineering (Hood et al, 2008b; Sommerville and Sawyer, 1997), manages the data created in the requirements elicitation and development phases of the project. Requirements management integrates this data into the overall project flow (Hood et al, 2008b) and supports the later lifecycle modification of the requirements (Hood et al, 2008b). As changes occur during the entire software project lifetime (Hood et al, 2008a), managing changes to the requirements is a major concern of requirements management (Lauesen, 2002; Kotonya and Sommerville, 1998) for large software systems. Moreover, in contexts like MDRE, a constant stream of new requirements and change requests is inevitable (Regnell and Brinkkemper, 2005). Uncontrolled changes to software may cause the cost of the regression testing to exceed 100 000 dollars (Hood et al, 2008b). Further, the absence of requirements management may sooner or later cause outdated requirements specifications as the information about changes to original requirements is not fed back to the requirements engineers (Hood et al, 2008b). Finally, the requirements management process descriptions in literature seldom consider managing obsolete requirements (Lauesen, 2002; Wiegers, 2003).

Scope creep, requirements creep and requirements leakage (also referred to as uncontrolled requirements creep) (Robertson and Robertson, 1999; Iacovou and Dexter, 2004) are related to OSRs. DeMarco and Lister (2003) identified scope creep as one of the five core risks during the requirements phase and state that the risk is a direct indictment of how requirements were gathered in the first place. Scope creep has also been mentioned as having a significant impact on risk and risk management in enterprise data warehouse projects (Legodi and Barry, 2010). Houston et al (2001) studied software development risk factors and 60% of 458 respondents perceived that requirements creep was a problem in their projects. Anthes (1994) reported that the top reason for requirements creep in 44% of the cases is a poor definition of initial requirements. Scope creep can lead to significant scope reductions as overcommitment challenges are addressed. This, in turn, postpones the implementation of the planned functionality and can cause requirements to become obsolete (Wnuk et al, 2009) or project failure (Iacovou and Dexter, 2004).

Despite its importance as a concept, in relation to managing requirements for software products, the phenomenon of OSRs seems to be underrepresented in literature. To the best of our knowledge, only a handful of articles and books mention the terms obsolete requirements or/and obsolete features. Among the existing evidence, Loesch and Ploederoeder (2007) claim that the explosion of the number of variable features and variants in a software product line context is partially caused by the fact that obsolete variable features are not removed. Murphy and Rooney (2006) stress that requirements have 'a shelf life' and suggest that the longer it takes from defining requirements to implementation, the higher the risk of change (this inflexibility is also mentioned by Rue et al (2010)). Moreover, they state that change makes requirements obsolete, and that obsolete requirements can dramatically extend project timelines and increase the total cost of the project. Similarly, Stephen et al (2011) list obsolete requirements as one of the symptoms of failure of IT project for the UK government. While the report does not define obsolete requirements *per se*, the symptom of failure is ascribed to obsolete requirements caused by the inability to unlock the potential of new technologies by timely adoption.

The phenomenon of OSRs has not yet been mentioned by standardization bodies in software engineering. Neither the IEEE 830 standard (IEEE, 1997) nor CMMI (v.1.3) (Institute, 2011) mention obsolete software requirements as a phenomenon. Actions, processes and techniques are also not suggested in relation to handling the complexity. On the other hand, Savolainen et al (2005) propose a classification of atomic product line requirements into these categories: non-reusable, mandatory, variable and obsolete. Moreover they propose a short definition of obsolete requirements and the process of managing these requirements for software product lines "by marking them obsolete and hence not available for selection into subsequent systems". Mannion et al (2000) propose a category of variable requirements called obsolete and suggest dealing with them as described by Savolainen et al (2005).

OSRs are related to the concept of requirements volatility. SWEBOOK classifies requirements into a number of dimensions and one of the them is volatility and stability. SWEBOK mentions that some volatile requirements may become obsolete (IEEE Computer Society, 2004). Kulk and Verhoef (2008) reported that the maximum requirements volatility rates depend on size and duration of a project. They proposed a model that calculates the "maximum healthy volatility ratios" for projects. Loconsole and Börstler (2005) analyzed requirements volatility by looking at the changes to use case models while Takahashi and Kamayachi (1989) investigated the relationship between requirements volatility and defect density. On the other hand, Zowghi and Nurmuliani (2002) proposed a taxonomy of requirement changes where one of the reasons for requirements changes is obsolete functionality, defined as "functionality that is no longer required for the current release or has no value for the potential users". For this paper, we understand requirements volatility as a factor that influences requirements change but different from requirements obsolescence. OSRs are, according to our understanding, any type of requirement (stable, small, large, changing) that is not realized or dismissed, but which accumulates in the companies' databases and repositories. Requirements obsolescence is defined as a situation where volatility becomes outdated and remains in the requirements databases (Harker et al, 1993; McGee and Greer, 2009).

Looking at previous work, software artifact obsolescence has been mentioned in the context of obsolete hardware and electronics in, for example, military, avionics or other industries. Among others, Herald et al proposed an obsolescence management framework for system components (in this case hardware, software, and constraints) that is mainly concerned with system design and evolution phases (Herald et al, 2009). While, the framework contains a technology roadmapping component, it does not explicitly mention OSRs. Merola (Merola, 2006) described the software obsolescence problem in today's defense systems of systems (the COTS software components level). He stressed that even though the issue has been recognized as being of equal gravity to the hardware obsolescence issue, it has not reached the same visibility level. Merola outlines some options for managing software obsolescence, such as negotiating with the vendor to downgrade the software license, using wrappers and software application programming interfaces, or performing market analysis and surveys of software vendors.

Due to the limited number of studies in the literature dedicated to the OSR phenomenon, we decided to investigate the concept utilizing a survey research strategy. We investigated the extent to which obsolete software requirements are perceived as a real phenomenon and as a real problem in industry. Moreover, we investigated how OSRs are identified and managed in practice, and what contextual factors influence OSRs.

3 Research methodology

This section covers the research questions, the research methodology, and the data collection methods used in the study.

3.1 Research questions

Due to the limited number of related empirical studies identified in relation to OSRs, we decided to focus on understanding the OSR phenomenon and its place in the requirements engineering landscape. Thus, most of the research questions outlined in Table 3.1 are existence, descriptive, as well as classification questions (Easterbrook et al, 2008). Throughout the research questions, we have used the following definition of OSRs, based on the literature study and the survey:

"An obsolete software requirement is a software requirement, implemented or not, that is no longer required for the current release or future releases, and which has no value or business goals for the potential customers or users of a software artifact for various reasons.[1]*"*

3.2 Research design

A survey was chosen as the main tool to collect empirical data, enabling us to reach a larger number of respondents from geographically diverse locations (Singer et al, 2008). Automation of data collection and analysis ensured flexibility and convenience to both researchers and participants, (Easterbrook et al, 2008; Dawson, 2005; L. M. Rea, 1005).

The goal of the survey was to elicit as much information from industry practitioners as possible in relation to OSRs. Therefore, we opted for an inclusive approach to catch as many answers as possible. This prompted the use of convenience sampling (L. M. Rea, 1005). The details in relation to survey design and data collection are outlined below.

3.2.1 Survey design

The questionnaire was created based on a literature review of relevant topics, such as requirements management, volatility, and requirements traceability (see Section 2). The questions were iteratively developed. Each version of the questionnaire was discussed among the authors and evaluated in relation to how well the questions reflected the research questions and the research goals.

The questionnaire contained 15 open and close-ended questions of different formats, e.g. single choice questions and multiple choice questions. In open-ended questions, respondents could provide their own answers as well as select a pre-defined answer from the list. The answers were analyzed using the open coding method Strauss and Corbin (1990). The data analysis was started without a preconceived theory in mind. We read all the answers and coded interesting answers by assigning them to a category with similar meaning. For close-ended questions, we used a Likert scale from 1 to 5, where 1 corresponds to *Not likely* and 5 to *Very likely* (Wikipedia, 2011).

[1]For reader convenience we present the definition in this section, rather than after presentation of the results. The description of how the definition was derived is available in Section 4.

Table 3.1: Research questions

Research question	**Aim**	**Example answer**
RQ1: Based on empirical data, what would be an appropriate definition of Obsolete Software Requirements (OSR)?	Instead of defining the phenomenon ourselves we base the definition on how the phenomenon is perceived in industry.	"An obsolete software requirements is a requirement that has not been included into the scope of the project for the last 5 projects"
RQ2: What is the impact of the phenomenon of obsolete software requirements on the industry practice?	To investigate to what degree is OSR a serious concern.	"Yes it is somehow serious"
RQ3: Does requirement type affect the likelihood of a software requirement becoming obsolete?	Are there certain types of requirements that become obsolete more often than others? Can these types be identified?	"A market requirement will become obsolete much faster than a legal requirement."
RQ4: What methods exist, in industry practice, that help to identify obsolete software requirements?	To enact a process to detect, identify or find obsolete software requirements or nominate requirements that risk becoming obsolete.	"To read the requirements specification carefully and check if any requirements are obsolete"
RQ5: When OSRs are identified, how are they typically handled in industry?	In order to identify possible alternatives for OSR handling, we first need to understand how they are handled today.	"We should mark found obsolete requirements as obsolete but keep them in the requirements database"
RQ6: What context factors, such as project size or domain, influence OSRs?	As a step in understanding and devising solutions for handling OSRs, it is important to identify contextual factors that have an influence on the obsolete requirements phenomenon.	"OSRs are more common in large projects and for products that are sold to an open market (MDRE context)"
RQ7: Where in the requirements life cycle should OSRs be handled?	To position requirements obsolescence in the requirements engineering life cycle.	"They should be a part of the requirements traceability task"

The questionnaire was divided into two parts: one related to OSRs (9 questions), and one related to demographics (6 questions). Table 3.2 shows the survey questions, with a short description of their purpose (2nd column), the list of relevant references (3rd column), and a link to the addressed research question (4th column). It should be observed that an OSR is defined in this work in the context of the current release in order to keep the question fairly simple and avoid introducing other complicating aspects. For reasons of brevity, we do not present the entire survey in the paper. However, the complete questionnaire, including the references that were used to construct the categories for the answers is available online (Wnuk, 2011e).

3.2.2 Operation (execution of the survey)

The survey was conducted using a web-survey support website called SurveyMonkey (Monkey, 2011). Invitations to participate in the questionnaire were sent to the potential audience via:

- Personal emails — utilizing the contact networks of the authors
- Social network websites (Linkedin, 2011) — placing the link to the questionnaire on the board of SE and RE groups and contacting individuals from the groups based on their designated titles such as senior software engineer, requirements engineer, system analyst, and project manager to name a few
- Mailing lists — requirements engineering and software engineering discussion groups (Wnuk, 2011c)
- Software companies and requirements management tool vendors (Wnuk, 2011d)

Masters and undergraduate students were excluded as potential respondents because their experience was judged insufficient to answer the questionnaire. The questionnaire was published online on the 3rd of April, 2011 and the data collection phase ended on the 3rd of May, 2011. In total, approximately 1700 individual invitations were sent out with 219 completed responses collected. The response rate, around 8%, is an expected level (Easterbrook et al, 2008; Singer et al, 2008). The results of the survey are presented in Section 4.

3.3 Validity

In this section, we discuss the threats to validity in relation to the research design and data collection phases. The four perspectives of validity discussed in this section are based on the classification proposed by Wohlin et al (Wohlin et al, 2000).

Table 3.2: Mapping between the questionnaire questions and the research questions

Question	Purpose	Relevant references	RQ
Q1	To derive the definition of Obsolete Software Requirements	(Savolainen et al, 2005; Herald et al, 2009; Zowghi and Nurmuliani, 2002; Merola, 2006)	RQ1
Q2	To investigate the impact of the OSRs on industry practice	(Murphy and Rooney, 2006; Stephen et al, 2011)	RQ2
Q3	To investigate how likely the various types of requirements would become obsolete	The list of requirements types was derived from analyzing several requirements classifications (Harker et al, 1993; McGee and Greer, 2009)	RQ3
Q4	To investigate the possible ways of identifying OSR in the requirements documents	(Loesch and Ploederoeder, 2007; Herald et al, 2009)	RQ4
Q5	To investigate the possible actions to be taken against obsolete requirements after they are discovered	(Loesch and Ploederoeder, 2007; Herald et al, 2009)	RQ5
Q6	To investigate whether there is a correlation between project size and the effects of OSRs	The classification of different sizes of requirements engineering was adopted from Regnell et al (Regnell et al, 2008)	RQ6
Q7	To investigate if OSRs are related to the software context	(Stephen et al, 2011)	RQ6
Q8	To understand where in the requirements life cycle OSRs should be handled	Current standards for requirements engineering and process models (IEEE, 1997; Institute, 2011) do not consider obsolete requirements.	RQ5, partly RQ7
Q9	To investigate if industry has processes for managing OSR	(Loesch and Ploederoeder, 2007; Savolainen et al, 2005)	RQ5

Construct validity. Construct validity concerns the relationship between the observations from the study and the theories behind the research. The phrasing of questions is a threat to construct validity. The authors of this paper and an independent native English speaker and writer-reviewer revised the questionnaire to alleviate this threat. To minimize the risk of misunderstanding or misinterpreting the survey questions, a pilot study was conducted on master students in software engineering. No participant in the pilot study indicated that the requirements categories in question 3 (Wnuk, 2011e) were hard to understand or vague. Still, the reader should keep in mind that the data given by respondents is not based on any objective measurements and thus its subjectivity affects the interpretation of the results. The mono-operational bias (Wohlin et al, 2000) threat to construct validity is addressed by collecting data from more than 200 respondents from 45 countries. Finally, the mono-method bias (Wohlin et al, 2000) threat to construct validity was partly addressed by analyzing related publications. While several related publications have been identified (see Section 2), this threat is not fully alleviated and requires further studies. Finally, considering social threats to construct validity it is important to mention the evaluation apprehension threat (Wohlin et al, 2000). The respondents' anonymity was guaranteed.

Some may argue that using the same questionnaire to define the term and to investigate it threatens construct validity. However, the fact that the presented OSR definition is based on over 50% of the answers and that the definition turned out to be independent of the respondents' roles, the size of the organizations, the length of the typical project, the domain and the development methodologies used gives us the basis to state that the understanding of the measured phenomenon was rather homogeneous among the respondents (Section 4.2). In addition, we do gain one aspect by combining the two, namely the subject's interpretation/understanding of what an obsolete requirement is. We are able to identify if respondents disagree, a fact which is essential for combining results (several respondents answers) for analysis.

Conclusion validity. Conclusion validity is concerned with the ability to draw correct conclusions from the study. To address the measures reliability threat, the questions used in the study were reviewed by the authors of this paper and one external reviewer, a native English speaker. The low statistical power threat (Wohlin et al, 2000) was addressed by using as suitable statistical tests as was possible on the given type of data. Before running the tests, we tested if assumptions of the statistical tests were not violated. However, since multiple tests were conducted on the same data, the risk of type-I error increases and using, for example, the Bonferroni correction should be discussed here. Since the correction was criticized by a number of authors (Arcuri and Briand, 2011; Nakagawa, 2004) it remains an open question if it should be used. Therefore, we report the p-values of all performed tests in case the readers want to evaluate the results using the

Bonferroni correction or other adjustment techniques (Arcuri and Briand, 2011). Finally, the random heterogeneity of subjects (Wohlin et al, 2000) threat should be mentioned here as this aspect was only partly controlled. However, low heterogeneity of subjects allows us to state conclusions of a greater external validity.

Internal validity. Internal validity threats are related to factors that affect the causal relationship between the treatment and the outcome. Reviews of the questionnaire and the pilot study addressed the instrumentation threat (Wohlin et al, 2000) to internal validity. The maturation threat to internal validity was alleviated by measuring the time needed to participate in the survey in the pilot study (15 minutes). The selection bias threat to internal validity is relevant as non-random sampling was used. Since the respondents were volunteers, their performance may vary from the performance of the whole population (Wohlin et al, 2000). However, the fact that 219 participants from 45 countries with different experience and industrial roles answered the survey minimizes the effect of this threat. Finally, the level of education in development processes and methodologies may have impacted the results from the survey. It remains future work to investigate whether this factor impacts the results. However, as the survey participants are professionals (many of whom work in large successful companies) their education might not be the main issue for discussion. What is interesting, however, is that we are investigating the state of current industry practice and not how it might be. Education is a powerful tool, but not the focus of this paper.

External validity. External validity threats concern the ability to generalize the result of research efforts to industrial practice (Wohlin et al, 2000). The survey research method was selected to assure as many responses as possible, generating more general results (Easterbrook et al, 2008; Lethbridge et al, 2005; L. M. Rea, 1005) than a qualitative interview study. Moreover, the large number of respondents from various countries, contexts, and professions contributes to the generalization of results.

4 Results and Analysis

The survey was answered by 219 respondents. When questions allowed multiple answers, we calculated the results over the total number of answers, not respondents. For questions that used a Likert scale, we present the results using average rating and the percentage received by each answer on the scale. All results are presented in percentage form and complemented by the number of answers or respondents when relevant. The answers given to the open questions were analyzed using the open coding method Strauss and Corbin (1990) (Section 3.2.1). Statistical analysis, when relevant, was performed using the chi-square test (Siegel and Castellan, 1998), and the complete results from the analysis, including contingency tables for some of the answers (Wnuk, 2011a), are listed online.

4.1 Demographics

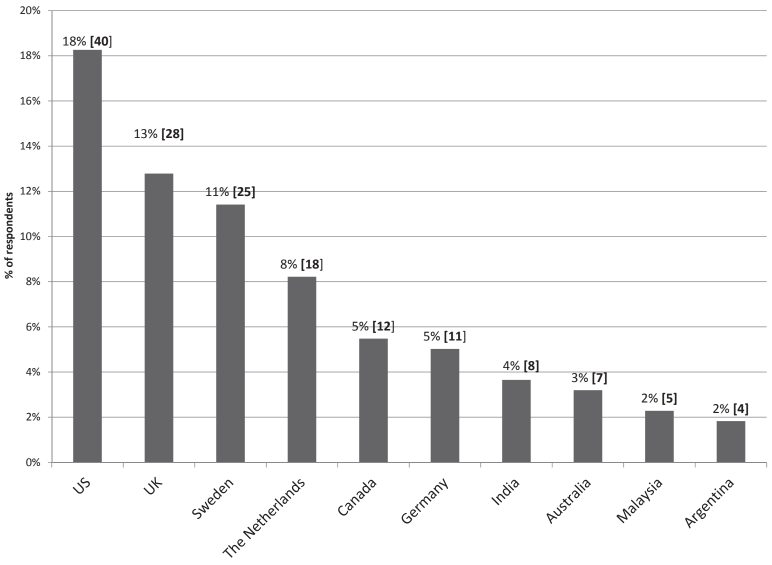


Figure 3.1: Top 10 countries among respondents

Figure 3.1 depicts the top 10 respondent countries (out of 45)[2]. The full list of the countries is available in (Wnuk, 2011b). The US and the UK constitute about 30% of the total respondents and 54% of the respondents came from Europe.

Figure 3.2 depicts the main roles of the respondents in their organizations. About one quarter of the respondents (24.9% or 54 respondents) described their role as requirements engineers, analysts or coordinators. The second largest category, *Other* (with 30 answers), include roles such as *System Engineers, Software Quality Assurance, Process Engineers, and Business Analysts*. The third largest category was *Researchers* or *Academics* (11.5% of all answers). *Software Project Managers* and *Software Architect or Designer* roles had the same number of respondents (22 each). Twelve respondents declared their main role as *Software Product Manager*, a relatively high number since product managers are generally few in an organization. This

[2]The actual category names have been changed for readability purposes. The original names are mentioned using *italics* in the paper and are available in the survey questionnaire (Wnuk, 2011e)

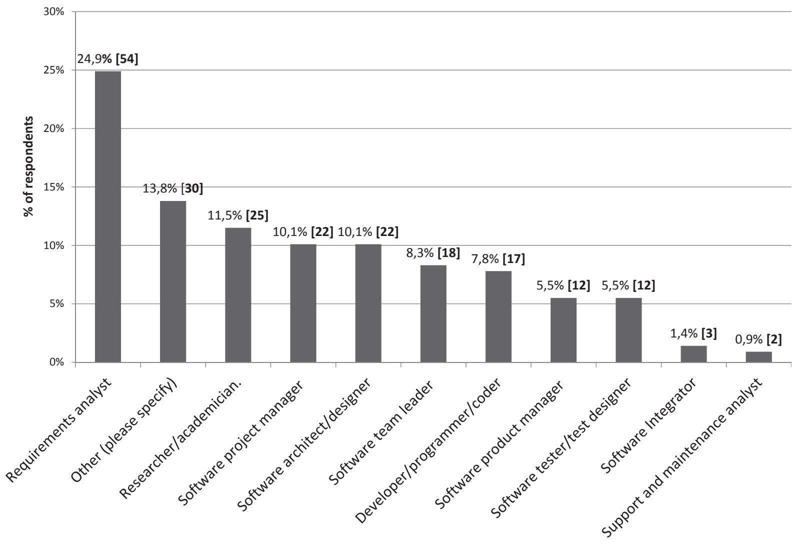


Figure 3.2: Main role of respondents

would seem to indicate that middle and senior managers overall represented a substantial part of the respondents.

Figure 3.3 gives an overview of the business domain of the respondents. A total of 32.8% stated the *IT or Computer and Software Services*. The second largest group (12.5%) is *Engineering* (automotive, aerospace and energy). These were followed by *Telecommunication* (10.7%) and *Consultancy* (9.3%).

Figure 3.4 depicts the sizes of the respondents' organizations. We can see that more than half of the respondents work in large companies (>501 employees).

Figure 3.5 looks at the average duration of a typical project in the respondents' organizations. About half of the respondents (∼45%) were involved in projects that lasted for less than a year, one quarter in projects that lasted between one and two years and one quarter in projects typically lasting more than two years.

Figure 3.6 investigates the development methodologies and processes used by the respondents. Since this question allowed for the possibility of providing multiple answers, the results are based on the number of responses. Agile development tops the list of answers with approximately a quarter (23.6%). Incremental and evolutionary methodology (18.8%) is in second place. Surprisingly, waterfall is still common and widely used (17.7%). In the *Other* category, the respondents reported that they mixed several methodologies "combination of agile and incremental" or "it is a

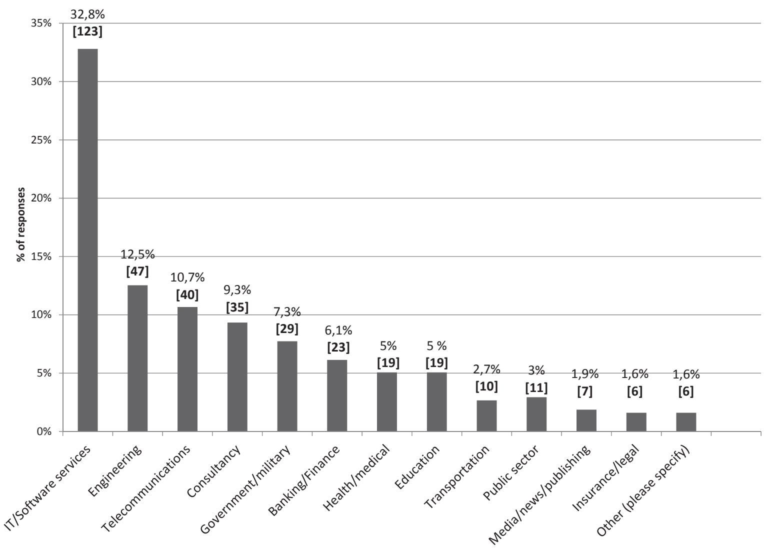


Figure 3.3: Types of business domains of respondents

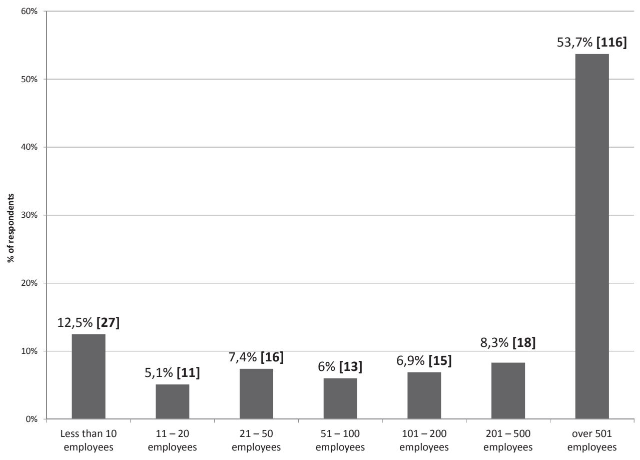


Figure 3.4: Size of respondents' organization

mix of incremental, agile and others". Other respondents used "V-Model", "SLIM", "CMMI Level 3", "CMMI Level 5" or had their own tailored methodology "created for each company by blending methods/processes".

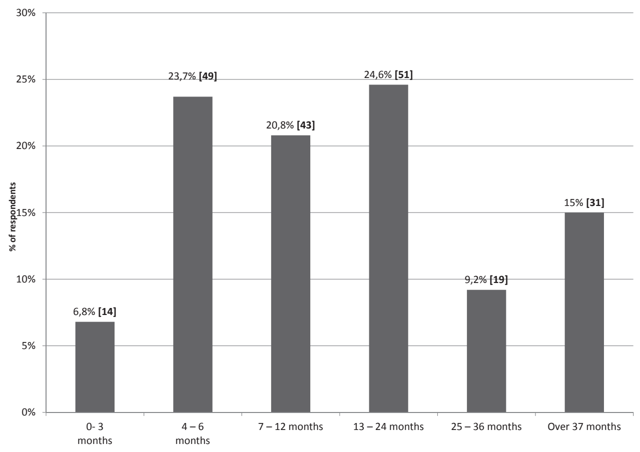


Figure 3.5: Average duration of typical projects from our respondents

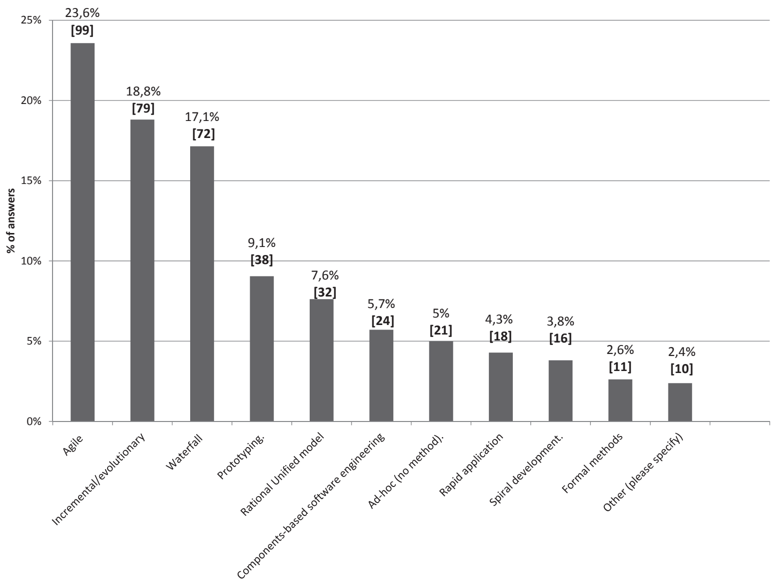


Figure 3.6: Development processes and methodologies

Figure 3.7 investigates the type of requirements engineering the respondents are involved in. Since this question also allowed multiple answers, the results are calculated based on the total number of responses. *Bespoke*

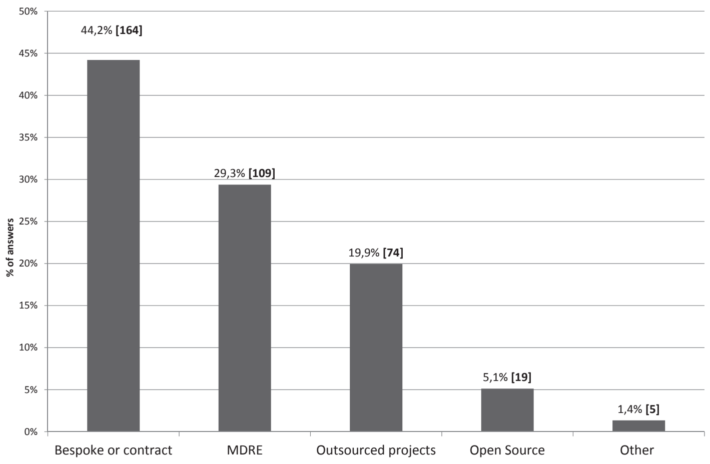


Figure 3.7: Types of Requirements Engineering

or Contract driven requirements engineering received 44.2% of all the answers. *Market-driven requirements engineering* received 29.5%, while *Open source* only 5.1%. *Outsourced projects* appeared in 19.9% of the answers. Six answers were given to the *Other* category. Two respondents suggested none of the following. One was working with "normal flow, requirements from product owner or developers", one with "builds" one mainly with infrastructure projects, and one with "large SAP implementation projects in a client organization".

4.2 Defining obsolete requirements (RQ1)

Defining the term Obsolete Software Requirement (OSR) is central to the understanding of the phenomenon. The categories used in this question were inspired by the definitions of OSR found in literature (Section 2), and are defined in the context of the current release (Section 3.2.1). Figure 3.8 depicts the answers from all respondents. Since the question allows multiple answers, the results are calculated for all the answers, not the respondents. The primary answer selected (29.9%) defines OSR as "*no longer required for the current release for various reasons*". This result is in line with the definition of obsolete functionality provided by Zowghi and Nurmuliani (Zowghi and Nurmuliani, 2002). The definition of an OSR as a requirement that: "*has no value for the potential users in the current release*" received 21% if the responses. This category is similar to the definition of obsolete software applications provided by Merola (Merola, 2006), as ap-

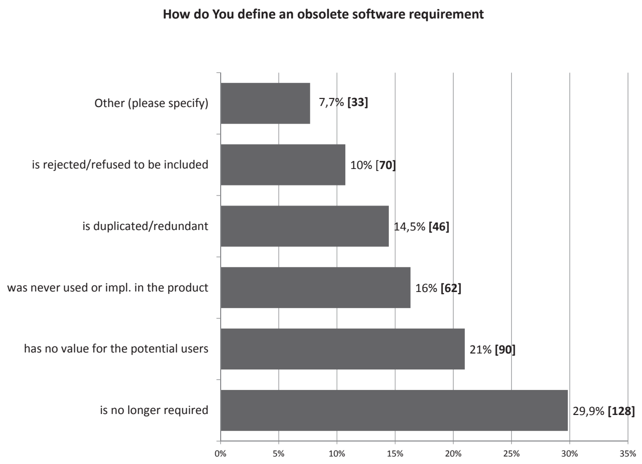


Figure 3.8: Respondents' definition of an OSR

plications are taken off the market due to decrease in product popularity or other market factors.

A total of 33 responses (7.7%) were in the *Other* category. Of these, 8 respondents ($\sim$ 25%) suggested that an OSR is not necessarily confined to the current release, but it also goes to future releases. Respondents stressed that an OSR is a requirement that has lost its business goal or value. Other interesting definitions included: "an OSR is a requirement that evolved in concept but not in documentation", "an OSR will be implemented but will not be tested", and "carved by the IT to showcase technical capabilities to the end user".

As a result, the following definition of an OSR was formulated:

"An obsolete software requirement is a software requirement (implemented or not) that is no longer required for the current release or future releases and, for various reasons, has little or no business value for the potential customers or users of a software product."

We performed statistical analyses to investigate whether there were relationships between the selected definition of OSRs and the respondents' roles, the size of organizations and the development methodologies used. Overall, the relationships were statistically insignificant due to violations of the chi-square test assumptions (some alternative answers had too few respondents, see Table A.2 in (Wnuk, 2011a)). However, significant results could be observed (using the chi-square test) between the top 5 methodologies (Figure 3.6) and the results for choice of OSR definition (p-value 0.011, Table A.2a in (Wnuk, 2011a)). Respondents that reported using a *Rational Unified Process* (RUP) methodology less frequently selected the definition of OSRs as no longer required for the current release (31.3% of all answers compared to over 50%) or never implemented in the product (34.4% of all answers compared to over 40%) than respondents that reported utilizing any of the remaining four methodologies. Moreover, the *RUP* respondents provided more answers in the *Other* category and indicated that OSRs can be "a requirement that evolved in concept but not in documentation" or "an abstract requirement to showcase the technical capability to the end user". Finally, only three *RUP* respondents defined OSR as a requirement that is rejected for inclusiong in the current release, while about 20% of the respondents that selected the other top four methodologies selected this answer. This would seem to indicate that the perceived definition of an OSR for respondents using the *RUP* methodology is more stable than that for respondents using other methodologies.

Since the *RUP* methodology considers iterative development with continuous risk analysis as a core component of the method (IBM, 2011), we can assume that the risk of keeping never used or implemented requirements in the projects is lower. Moreover, the majority of the *RUP* respondents also reported working on bespoke or contract-driven projects, where the number of changes after the contract is signed is limited and usually extensively negotiated. Thus it appears to be possible that the *RUP* respondents could avoid rejected or refused requirements and could manage to achieve more precise and stable agreements with their customers (IBM, 2011) which in turn could result in fewer OSRs.

Reviewing the top five methodologies, the most popular answer was no longer required for the current release. Interestingly, among the respondents working with agile, incremental or evolutionary methodologies, the fourth most popular answer was *never used or implemented in the product*.

In contrast, respondents who worked with waterfall, prototyping or RUP methodologies have the same order of popularity of answers. The definition of an OSR as a *was never used or implemented in the product* requirement was the second most popular answer while the option *is duplicated/redundant in the current release* was the third most popular answer. The possible interpretation of these results is that agile and incremental methodologies less frequently experience OSRs as never used or implemented but experience more OSRs as duplicated requirements and requirements with no value for the potential users.

Further analysis reveals that the definition of OSRs is not significantly related to the size of the companies, the length of the typical project, or the domain (p-values in all cases greater than 0.05). Domain and project length could be seen as qualifiers of OSRs. For example, projects running over long periods could suffer increased requirements creep (Wnuk et al, 2009). However, this would most probably not be visible in the definition of OSRs, but rather in the impact of OSRs, which is investigated in the next section.

4.3 The potential impact of OSRs (RQ2)

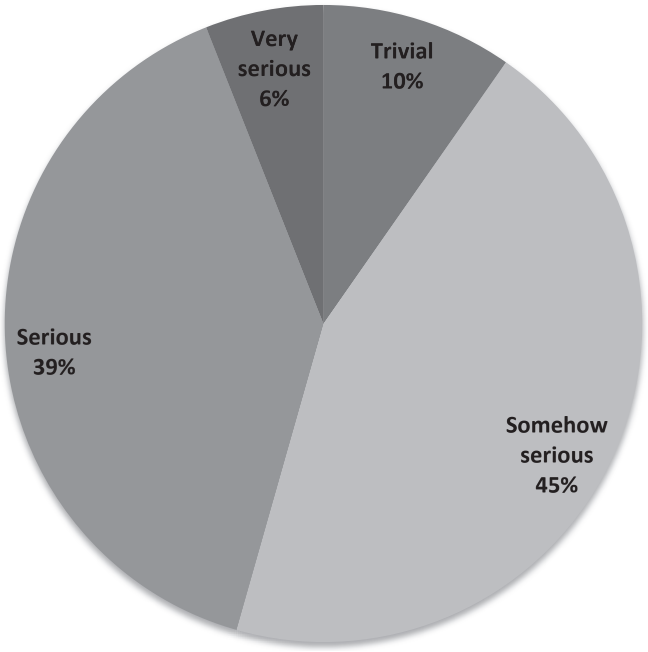


Figure 3.9: Impact of OSRs on industry practice

When queried about the potential impact of OSRs on their product development efforts a total of 84.3% of all respondents considered OSR to be *Serious* or *Somehow serious* (Figure 3.9). This indicates that among the majority of our respondents OSRs seems to have a substantial impact on product development. Our result confirms previous experiences. (See, e.g., Murphy and Rooney (Murphy and Rooney, 2006), Stephen et al (Stephen et al, 2011) and Loesch and Ploederoeder (Loesch and Ploederoeder, 2007)). For 6% of the respondents OSRs are a *Very serious* issue, while 10% (21 respondents) deemed OSR a *Trivial* matter.

To further decompose and test context variables, e.g., company size, respondents' roles and development methodologies, we performed chi-square tests (Table A.1 in (Wnuk, 2011a)) between the context variables and the degree to which OSRs were considered having a substantial impact. The tests resulted in p-values greater than 0.05, which indicates that no statistically significant relationships between the analyzed factors could be seen. We can, however, ascertain that a clear majority of the respondents deemed the phenomenon of OSRs a relevant factor to be taken into consideration in development efforts.

Of the 21 (10%) respondents who considered OSRs to be *Trivial*, approximately 58% worked with requirements or in project management roles. This would seem to indicate that those respondents, contrary to those in software development roles, have less difficulty in managing OSRs. An analysis of the answers to questionnaire question 9 ((Wnuk, 2011a) and Section 4.9) revealed that 10 respondents who considered OSRs to be *Trivial* also confirmed having a process for managing OSRs. Thus, it appears to be a logical conclusion that the negative influence of OSRs on product development could be alleviated by designing and introducing an appropriate process of managing OSRs. More about the current processes discovered among our respondents can be found in Section 4.9.

Further analysis of the respondents who considered OSRs as *Trivial* indicated that more than 80% of them worked for large companies with $>$ 101 employees. Since large companies often use more complex process models (Berenbach et al, 2009), in contrast to small companies which might have budget constraints to prevent hiring highly quality professionals and whose processes are typically informal and rather immature (Quispe et al, 2010), we could assume that the issue of managing OSRs could have been already addressed in these cases.

Further analysis of the *Trivial* group indicated that almost half of them (47.6%) worked in the *IT or computer and software services* domain, In the service domain, the main focus of requirements engineering is to identify the services that match system requirements (Bano and Ikram, 2010). In the case of insufficient alignment of new requirements with the current system, product development may simply select a new, more suitable, service. This, in turn, might imply that the OSRs are discarded by replacing the old service with the new one. Further, the typical product lifetime for IT systems is usually shorter than for engineering-focused long-lead time products (Kossmann et al, 2009) (such as those in the aerospace industry), which in turn could minimize the number of old and legacy requirements that have to be managed. The possible interpretation of our analysis is that OSRs are less critical in IT and service oriented domains. Although this is a possible and plausible explanation, further investigation is needed to reach a conclusion.

Among the respondents who considered OSRs *Very serious* (13 respondents), the majority (53.8%) worked in large companies and used agile, ad-hoc, or incremental methodologies (61.6%). This result seems to indicate that OSRs are also relevant for agile development and not reserved for only more traditional approaches like waterfall. Ramesh et al (2010) pointed out after Boehm (2000) that quickly evolving requirements that often become obsolete even before project completion significantly challenge traditional (waterfall) requirements engineering processes. Murphy and Rooney (2006) stressed that the traditional requirements process seriously contributes to the creation of obsolete requirements by creating a "latency between the time the requirements are captured and implemented". This latency should be lower in agile projects, characterized by shorter iterations and greater delivery frequency. This might indicate that either the latency is present in agile projects as well, or that latency is not the primary determinant of OSRs. It should be observed that 69.2% of the respondents who considered OSRs as *Very serious* reported having no process for handling OSRs. This could indicate why OSRs were considered a *Very serious* problem.

The cumulative cross tabulation analysis of the respondents who considered OSRs *Somehow serious*, *Serious* or *Very serious* (total 196 respondents, 89%) confirmed the severe impact of OSRs on large market-driven and outsourced projects (Section 4.7.2). Moreover, 76.8% of those respondents reported that they had no process, method, or tool for handling OSRs. In addition, 72.3% of respondents who considered OSRs *Somehow serious*, *Serious* or *Very serious* used manual methods to identify OSRs. It is also interesting to observe that the were only small differences between answers fron respondents who declared the following: *Agile software development* or *Incremental or evolutionary development* methodologies, and *Waterfall development*. Respondents using *Waterfall development* (and considered OSRs *Serious* or *Somehow serious* or *Very serious*) were somewhat more prone to dismiss the impact of OSRs compared to respondents using *Agile software development* or *Incremental or evolutionary development* methodologies. This would seem to indicate that, because waterfall-like processes usually restrict late or unanticipated changes and focus on extensive documentation (2007; 2008; 1987), the impact of OSRs in those processes could be minimized. However, it says nothing about the extent that the realized features were useful or usable for the customers. Some waterfall projects may not have perceived OSRs to be a major issue for the project, but they might be for the product *per se*. That is, implementing an outdated feature might not be a perceived as a problem in a project. At the product level, where the overall value of the product for the customer should be maximized through the selection of the right features to implement and best alternative investment should be considered, another feature could be implemented instead of the outdated one. This is a classical case of perspective being a part of the value consideration as described by Gorschek and Davis (2008).

The type of requirements engineering context factor (Figure 3.7) only minimally influenced the overall results for this questionnaire question. Respondents who reported to work with *Bespoke or contract driven requirements engineering* graded OSRs slightly less serious than respondents who reported working with *MDRE*. This seems to indicate that OSRs are a problem in both contract driven (where renegotiation is possible (Regnell and Brinkkemper, 2005)) and market-driven (where time to market is dominant (Regnell and Brinkkemper, 2005)) projects. However, the difference could also indicate that there is a somewhat alleviating factor in contract-based development. That is, contract based development aims at delivering features and quality in relation to stated contract, thus getting paid for a requirement even if it is out of date at delivery time. In an MDRE context, however, the product might fail to sell if the requirements are not fulfilled and the features out of date (Regnell and Brinkkemper, 2005).

4.4 Requirements types and OSRs (RQ3)

The respondents were asked to choose what types of requirements were most likely to become obsolete (Likert scale, 1 = *Not likely*, and 5 = *Very likely*). The classification was based on a review of several classification schemes Aurum and Wohlin (2005b); Shan et al (2010); Bourque and Dupuis (2004); Nurmuliani et al (2004). We chose to use the pre-defined "types" derived from classifications proposed by Harker et al (1993) and McGee and Greer (2009) because Harker et al (1993) categorized software requirements based on their changing nature and McGee and Greer (2009) developed a taxonomy of software requirements change sources based on knowledge and experience of project managers.

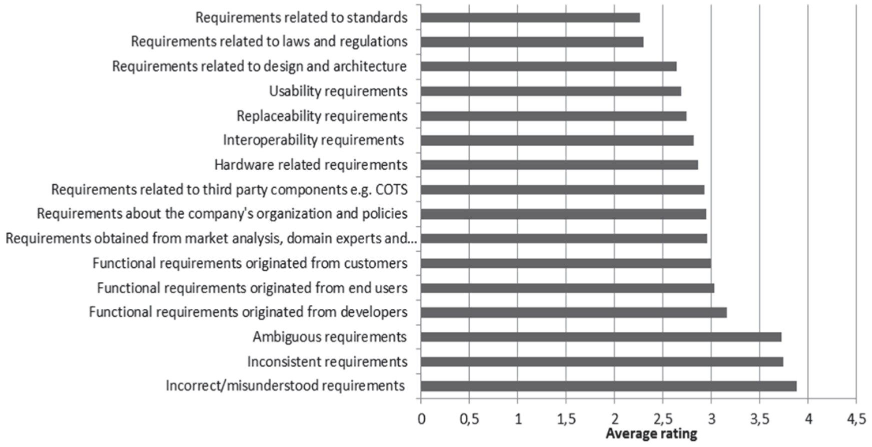


Figure 3.10: Types of OSRs likely to become obsolete

According to the results depicted in Figure 3.10, OSRs seem to belong to the categories of *Incorrect or misunderstood requirements* (mean 3.88), *Inconsistent requirements* (mean 3.74), or *Ambiguous requirements* (mean 3.72). While several studies focused on the problem of inconsistencies between requirements, e.g., by proposing techniques to identify and remove inconsistencies (Robinson and Pawlowski, 1999), decomposing a requirements specification into a structure of "viewpoints" (Russo et al, 1998), or distributing development of specifications from multiple views (Finkelstein et al, 1994) , they didn't study inconsistent requirements as a potential source of OSRs. From a becoming obsolete standpoint, the level and quality of specification should not matter *per se*. However, if the lack of quality of a requirement's specification is seen as an indicator of a lack of investment in the analysis and specification of the requirement, several possible scenarios could emerge. For example, practitioners in industry might have a *gut feeling* that certain requirements will become OSRs and thus, are not worth the effort. Another possibility is that OSRs are harder (require more effort and knowledge) to specify than other requirements types, although, it could just as well indicate that most requirements are specified badly and thus are also OSRs. Further investigation is needed to investigate the potential reasons for the results achieved. The only thing we can say for certain is that requirements becoming obsolete seem to suffer from inadequacies in terms of correctness, consistency, and ambiguous specification.

Interestingly, requirements from domain experts were considered less likely to become obsolete than requirements from customers, end users, and developers respectively. One explanation could be that domain experts possess the knowledge and experience of the domain, and thus their requirements may be less likely to change (Easterbrook, 2004). On the other hand, since the customers are the main source of software requirements and the main source of economic benefits to the company, their requirements are crucial to the success of any software project (Gorschek and Wohlin, 2005). This implies that this category must be kept up to date and thus be less likely to become obsolete. Another possible explanation could be that customer requirements are not as well or unambiguously specified as internal requirements (Gorschek and Wohlin, 2005; Lauesen, 2002), resulting in a tendency of those requirements to become obsolete faster or more frequently.

Obsolescence of customer requirements, rather than internal requirements from domain experts, is confirmed by Wnuk et al (2009). They reported that stakeholder priority dictates removal and postponement of the requirements realization, and domain experts are often part of the prioritization of all requirements. On the other hand, Kabbedijk et al (2010) reported that change requests from external customers are more likely to be accepted than change requests from internal customers. This might imply that some customer requirements are handled as change requests instead of as requirements input to development projects. In both cases, the authors reported high requirements volatility, which is in line with the study by Zowghi and Nurmuliani (2002) who related obsolete requirements related to requirements volatility.

According to our respondents, requirements related to standards, laws and regulations are the least likely to become obsolete, which seems logical, as the lifetime of legislation and standards is often long in comparison to customer requirements. Furthermore, the low average score for the *Requirements related to third party components e.g. COTS* (even lower than for the requirements related to the company's organization and policies) also seems to be logical, especially in relation to the results for RQ2 (Section 4.3) where almost half of the respondents who considered OSRs to be *Trivial* worked with *IT or Computer and software services domain*. We assume, after Bano and Ikram (2010), that COTS are used in the software service domain. The results for the respondents who worked with *Outsourced projects* (question 15 in (Wnuk, 2011e)) are in accordance with the overall results.

The differences between the respondents who worked with *Outsourced*, *MDRE* and *Bespoke or contract driven requirements engineering* projects in relation to the degree of obsolescence of COTS requirements are subtle. This may suggest that other aspects not investigated in this study could influence the results. Although OSRs do not appear to be related to the main challenges of COTS systems, i.e., the mismatch between the set of capabilities offered by COTS products and the system requirements (Kohl, 2001), the nature of the COTS selection process, (e.g. many possible systems to consider and possible frequent changes of the entire COTS solution), may help to avoid OSRs.

Further analysis of the influence of the context factors indicates that the respondents' domains, company size, and methodologies have minimal impact on the results. Not surprising, more respondents who worked with projects running over longer time spans graded *Functional requirements originated from end users* as *Very likely* to become obsolete than respondents who worked with short projects (8.7% of respondents who worked with projects <1 year and 25.7% respondents who worked with projects > 1 year). One explanation could be that long projects, if deprived of direct and frequent communication with their customers and exposed to rapidly changing market situations, can face the risk of working on requirements that are obsolete from the users' point of view. This interpretation is to some extent supported by the results from RQ7 (Table 3.4) where the respondents graded *MDRE* contexts (characterized by limited possibilities to directly contact the end users and continuously arriving requirements (Regnell and Brinkkemper, 2005)) or *Outsourced projects* (where communication is often done across time zones and large distances (Holmstrom et al, 2006)) as more affected by OSRs than bespoke contexts. The success of *Market-driven projects* primarily depends on the market response to the proposed products (Regnell and Brinkkemper, 2005), which if released with obsolete functionality, may simply be required by customers. Thus, we believe that it is important to further investigate additional factors that could render *Functional requirements originated from end users* obsolete.

4.5 Methods to identify OSRs (RQ4)

More than 50% of the answers pointed out that manual ways of discovering OSRs are currently the primary method (Figure 3.11). At the same time, the context factors such as the different methodologies, types of RE, length of the projects, roles of respondents and the domain that respon-

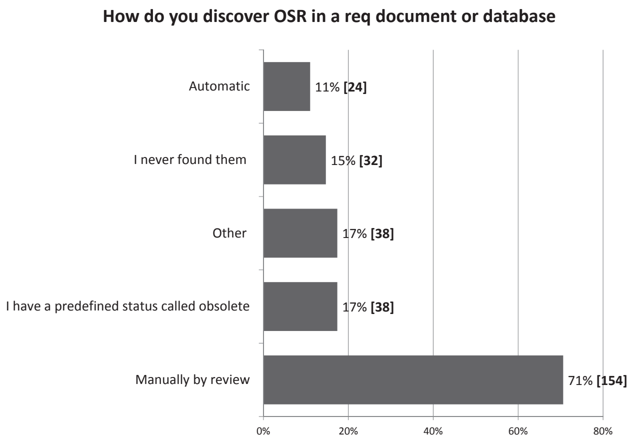


Figure 3.11: Methods used to identify OSRs

dents worked in did not significantly affect the top answer for this question. A total of 13.29% of all answers indicated the presence of a predefined "obsolete" status. Furthermore, 11.19% of all answers (32 answers) were given to the category *I never found them or I never thought of finding them*. Finally, less than 10% of all answers (24 answers) indicated the existence of any sort of automation to identify OSRs.

In the *Other* category, seven respondents mentioned that OSRs could be identified "by execution of test cases based on requirements" or "during regression testing cycles". Further, three answers suggested "using requirements traceability matrix while testing the software" while three answers suggested improved communication "by discussion of user stories with stakeholders". Finally, one respondent suggested that goal-oriented requirements engineering makes "finding OSRs trivial".

The answers from respondents who indicated using automated ways of discovering OSRs provided some names for the automated techniques, e.g., "customized system based on JIRA that takes OSRs into account by using special view filters", "traceability using DOORs to analyze for orphan and to track and status obsolete requirements", or "a tool called Aligned Elements to detect any inconsistencies including not implemented requirements". This would indicate that some tool support is present. However, tool efficiency and effectiveness was not part of this study.

Further analysis indicated that the majority of respondents using tools of some sort worked with companies with $>$ 501 employees (62%). This seems reasonable as large companies usually have more money for tool support (Quispe et al, 2010), and can even request especially tailored software from the requirements management tool vendors. The fact that automated methods to identify OSRs are rare among the smaller companies calls for further research into lightweight and inexpensive methods of OSR identification that can more easily be adapted in those companies. Furthermore, as both smaller and larger companies fall short in automation and assuming that larger companies can invest more money into education, this is probably not due to education either.

More than half (15) of the respondents from the automated group also indicated that they identify OSRs manually. One explanation could be that automated methods are used together with manual methods, e.g., after the respondents manually mark requirements as obsolete or perform other preliminary analysis that enables automated sorting. Searching, tagging or filtering capabilities in their requirements management tools are most likely dominant and seen as *automated* in relation to OSRs, but this task is done in an ad-hoc manner and not integrated with their requirements management process. Thus the "level of automation" needs further investigation.

The reasonably high number of answers given to the category *I never found them or I never thought of finding them* is intriguing and needs further investigation. Thirty respondents from this group (93.8%) also indicated having no process for managing OSRs. This seems logical as the inability to find OSRs could be related to the lack of processes for managing OSRs. Further, the majority of the respondents that indicated never finding OSRs worked with projects shorter than 12 months, and one fourth of them indicated having an ad-hoc process for managing requirements. The relatively short project times were not an indication of OSRs not being an issue as >80% of these same respondents indicated OSRs as being a *Serious* or *Very serious* issue. The absence of a defined and repeatable process might be a better indicator for not identifying OSRs in this case. In addition, waterfall was represented in more than 11% of the cases, while only about 6% worked in an agile manner.

Neither organizational size nor development methodology were statistically significant factors in terms of how OSRs were discovered or identified (Table A.5 in (Wnuk, 2011a)). However, a statistically significant relationship was identified in relation to the top five methodologies and how OSRs were identified (chi-square test $p<0.004$, Table A.5a in (Wnuk, 2011a)). This result could be explained by the following: (1) respondents who worked with waterfall methodology admitted more often to never finding OSRs (11%) than respondents who worked with agile methodologies (3.8%), (2) more respondents who worked with *RUP* methodology (34%) selected the option *I have a predefined status called obsolete* than respondents who worked with agile methodology (10%). Looking further, we could also see that the majority of the respondents who worked with *RUP* or *Prototyping* methodologies also worked with companies with >201 employees. This would seem to indicate that within the two mentioned methodologies it is possible to implement tool support for identification of OSRs. It is worth mentioning that a statistically significant relationship was also achieved between the top 5 methodologies and the results for choice of OSR definition (p-value 0.011, Table A.2a in (Wnuk, 2011a)) and Section 4.3. The results suggest that the respondents who worked with the RUP methodology may have a different opinion about the definition of OSRs and more frequently use a predefined status called obsolete to identify OSRs.

Looking at the types of requirements engineering used, the results showed that the respondents who work with *Bespoke or contract driven requirements engineering* didn't use predefined categories for OSRs; it was not part of their standard procedure to sort out OSRs. This seems to be logical as the majority of the respondent who admitted to never finding OSRs worked with bespoke or contract-driven projects. Finally, only one respondent mentioned automatic methods of finding OSRs.

For the context factor of project length, longer projects have more automated ways of identifying OSRs (the difference is about 5%) than shorter projects. This seems reasonable as longer projects usually invest more into project infrastructure and project management tools and processes. However, a large part of the longer projects respondents also indicated manual methods of identifying OSRs (about 60% for projects >1year). In comparison, subjects typically working in shorter projects used more tool supported automated methods (about 52% for projects <1 year). Thus the respondents working in longer projects did see the point of, and did try to, identify OSRs to a larger extent than the ones working in shorter duration projects, although manual methods dominated.

The analysis of the influence of the respondents' roles on the results revealed only minimal differences. Among the interesting differences, project and product managers respondents gave no answers in the I never found them category. This may indicate that they always find OSRs. Further, the management roles had the highest score for manual identification of OSRs. This result might indicate that management is, to some extent, more aware of the need for finding OSRs which may severely impede the project efforts. However, tool support is often lacking.

4.6 Handling of identified obsolete software requirements (RQ5)

More than 60% of the answers (results for multiple answer questions are calculated based on all the answers) indicated that the respondents kept the OSRs but assigned them a status called "obsolete" (see Figure 3.12). This might indicate that OSRs are a useful source of information about the history of the software product for both requirements analyst and software development roles. Moreover, 21.9% of all answers (66) suggested moving OSRs into a separated section in requirements documents. These views were the most popular among the respondents regardless of their role,

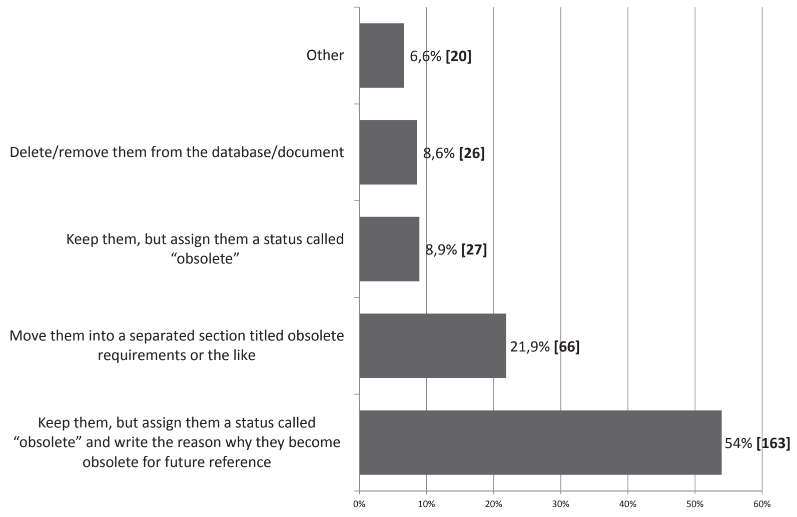


Figure 3.12: Methods used to manage identified OSRs

methodology, domain, size, project length and context. One could interpret this response as indicating that the most suitable way to manage identified OSRs is to classify them as obsolete, supplying rationale, and move them into a separated section or document or SRS. However, maintaining traceability links between OSRs and other requirements could prove work intensive, especially if end-to-end traceability is required (Berenbach et al, 2009). Regnell et al (2008) discuss scalable methods for managing requirements information where effective grouping of requirements e.g., placing semantically similar requirements in the same module, could enable more efficient maintenance of large structures of requirements (although OSRs were not mentioned specifically).

Looking at the answers given the *Other* category, two answers suggested informing the stakeholders about assigning a requirement an obsolete status. Furthermore, two respondents suggested to "hide and tag requirements that are obsolete using requirements management tools". Interestingly, one respondent questioned "why would you spend time in on dealing with not needed things". Since this person worked with a very small company with about 20 employees, we assume that the problem of overloaded database with legacy requirements is not known to this person. Finally, the other answers in this category mostly suggested keeping OSRs and optionally writing the justification.

Most of the answers in the *Other* category ($\sim$6%, 20 answers) suggested either removing OSRs, or keeping them, but moving them to a separated section or module in the database. Only ∼9% of answers (26) suggested deleting the OSRs from the requirements database or document. This suggests that most respondents think OSRs should be stored for reference and traceability reasons. However, keeping OSRs appears to be inconsistent with recommended practice for reducing the complexity of large and very large projects (Regnell et al, 2008; Buhne et al, 2004), and handling information overload as highlighted by Regnell et al (2008). The desired behavior in large and very large projects would seem to indicate the removal of unnecessary requirements to decrease the complexity of the requirements structure and traceability links. One possible avenue for further investigation is to evaluate the value of keeping OSRs.

Of the group who opted for OSRs deletion upon identification, the majority of the answers came from respondents who worked with large companies (>501 employees, 77%) and long projects (>12 months, 53.9%). Moreover, a majority of these respondents considered OSRs to be *Serious* or *Somehow serious* (Section 4.3). On the contrary, respondents that worked in smaller companies opted to keep OSRs.

Analysis revealed a lack of statistically significant relationships between the answers for this question (Figure 3.12) and the respondents' roles, domains, organizational size and, methodologies used (Table A.6 in (Wnuk, 2011a)). However, some indications could be observed. Respondents working in the engineering domain seemed to prefer the deletion of OSRs compared to respondents from other domains. One possible explanation could be that since the projects in the engineering domain are highly regulated, and often require end-to-end traceability (Berenbach et al, 2009), keeping OSRs in the scope could clutter the focus threatening to impede requirements and project management activities.

Type of requirements engineering factor turned out to have a minimal impact on the results regarding this question. However, one observation worth mentioning is that more support was given to the option of removing OSRs among the respondents who worked with *Bespoke or contract driven requirements engineering* (12.3%) than respondents who worked in *MDRE* (9.2% of answers). This appears to be logical as, in bespoke projects, obsolete requirements could be discarded after the contract is fulfilled. In market-driven projects they could be kept and later used during the requirements consolidation task, where new incoming requirements could be examined against already implemented or analyzed requirements which include OSRs (Natt och Dag et al, 2006) .

Table 3.3: OSRs effect on project size (215/219 respondents)

	(1) Not likely	(2) Somewhat likely	(3) Likely	(4) More than likely	(5) Very likely	Rating Average
Small-scale (∼10 of req.)	35.3% (76)	**35.8% (77)**	13.5% (29)	7.0% (15)	8.4% (18)	2.17
Medium-scale (∼100 of req.)	9% (19)	31.6% (67)	**41.5% (88)**	16.0% (34)	1.9% (4)	2.70
Large-scale (∼1000 of req.)	3.8% (8)	17.1% (36)	31.3% (66)	**32.7% (69)**	15.2% (32)	3.38
Very large-scale (>10000 of req.)	8.1% (17)	12.8% (27)	16.6% (35)	23.7% (50)	**38.9% (82)**	3.73

4.7 Context factors and obsolete software requirements (RQ6 and RQ7)

4.7.1 Obsolete software requirements and project size

The respondents were asked to indicate to what extent the phenomenon of OSRs would potentially (negatively) impact a project, and whether project size had anything to do with the likelihood of negative impact. The respondents used a Likert scale from 1 (*Not likely* impact) to 5 (a *Very likely* impact). The results are presented in Tables 3.3 and 3.4 below. The size classification is graded in relation to number of requirements and interdependencies, inspired by Regnell et al (2008).

Column 7 in Table 3.3 presents the average rating for each project size. We see that the larger the project, the more likely there will be a negative effect from OSRs. Looking at Table 3.3 for *Small-scale requirements* projects, most respondents deemed OSR impact as *Not likely* (35.3%) or *Somewhat likely* (35.8%). However, moving up just one category to *Medium-scale requirements* projects with hundreds of requirements, the respondents indicated the impact as being *Likely* (41.5%). The trend continues with *More than likely* (32.7) for *Large-scale requirements* projects, and *Very likely* for *Very large-scale requirements* projects (38.9%). The results confirm the viewpoint of Herald et al (2009) who listed OSRs as one of the risks in large integrated systems.

One interesting observation is that the results could be seen as potentially contradictory to the results from questionnaire question 2 (Section 4.3) where the respondents who worked in larger companies (over 100 employees) graded the overall impact of OSRs slightly lower than re-

Table 3.4: How likely OSRs affect various project types (215/219 respondents)

	(1) Not likely	**(2) Somewhat likely**	**(3) Likely**	**(4) More than likely**	**(5) Very likely**	**Rating Average**
Bespoke projects	14.4% (31)	**32.1% (69)**	26% (56)	16.3% (35)	11.2% (24)	2.78
Market-driven projects	6.5% (14)	20% (43)	**35.8% (77)**	23.3% (50)	14.4% (31)	3.19
Outsourced projects	2.3% (5)	16.4% (35)	**35.7% (76)**	27.2% (58)	18.3% (39)	3.43

spondents from smaller companies. However, since large companies often have large databases of requirements (Regnell et al, 2008) and often run projects with several thousands of requirements Konrad and Gall (2008), this would suggest that there are other factors that influence the impact of OSRs.

When it comes to the influence of methodology used by our respondents, we report that the respondents who used *Agile software development* methodology primarily graded OSRs as only *Likely* to affect *Large-scale requirements* projects, while respondents who used *Waterfall* methodology primarily graded the impact of OSRs as *More likely*. Interestingly, this result seems to contradict the results for RQ2 (Section 4.3), where the majority of respondents who considered OSRs *Very serious* worked in large companies and used agile or incremental methodologies. This might indicate that the size of the project is not more dominant than the size of the company, and the methodology used. This requires further investigation.

The respondents who worked with bespoke or contract driven requirements engineering primarily graded the effect of OSRs on *Large-scale requirements* projects as *Likely*. On the contrary, the respondents who worked with *Market-driven projects* primarily graded the impact of OSRs on *Large-scale requirements* projects as *Very Likely*. This result confirms the results for RQ2 (Section 4.3) where OSRs were also graded less serious by respondents who worked in bespoke contexts. Finally, for the *Very large-scale requirements* projects our respondents primarily graded the impact of OSRs as *Very likely* regardless of context factors.

4.7.2 Obsolete software requirements and project types

The respondents were also asked to rate how likely it was that OSRs affected various project types (on a scale from 1 to 5, where 1 is *Not likely*, and 5 is *Very likely*). The results for the average rating (column 7 in Table 3.4) indicate that *Outsourced projects* are the most likely to be affected by OSRs (average rating 3.43). One possible explanation for this result could be the inherited difficulties of frequent and direct communication with customers and end users in *Outsourced projects*. Moreover, as communication in *Outsourced projects* often needs to be done across time zones and large distances (Holmstrom et al, 2006; Šmite et al, 2010), the risk of requirements misunderstanding increases, and as we have seen (Section 4.4), inadequately specified requirements run a higher risk of becoming OSRs.

The high average rating for the *Market-driven projects* (average scope 3.19) can be explained by the inherited characteristics of the MDRE context where it is crucial to follow the market and customer needs and the direct communication with the customer may be limited (Regnell and Brinkkemper, 2005). This in turn can result in frequent scope changes (Wnuk et al, 2009) that may render requirements obsolete. Finally, it is worth mentioning that the gap between the *Market-driven projects* and *Bespoke projects* (average score 2.78) is wider than between *Outsourced* (average scope 3.43) and *Market-driven projects* (average score 3.19). One possible explanation could be that both *Market-driven projects* and *Outsourced projects* suffer similar difficulties in directly interacting with the end users or customers (Regnell and Brinkkemper, 2005; Holmstrom et al, 2006) and thus the risk of requirements becoming obsolete could be higher.

The results for all the categories and scales are presented in columns 2 to 6 in Table 3.4. Our respondents primarily graded the impact of OSRs on *Market-driven projects* and *Outsourced projects* as *Likely* and only *Somehow likely* for *Bespoke projects*. Interestingly, the answer *Very likely* did not receive top scores for any of the three types of projects. This would seem to indicate that the "project type" factor is less dominant in relation to OSRs than the "size" of the project discussed earlier in this section.

Since the statistical analysis between the results from the question and the context variables revealed no significant relationships, we performed descriptive analysis of the results. The respondents who indicated having a managerial role (32.7%) primarily graded the impact of OSRs on the *Market-driven projects* as *More than likely*, while the requirements analysts primarily graded this impact as only *Likely*. Similar to this result are the results for RQ2 (Section 4.3) where the managers primarily considered OSRs as *Serious* while requirements analysts predominantly considered it *Somehow serious*. The comparison is, however, not straight forward as in case of RQ2 where respondents were grading all types of requirements projects, not only *Bespoke projects*. Finally, the opinions of software development and management roles are aligned when grading the impact of OSRs on bespoke projects (the majority of the respondents from both roles graded the impact as *Somehow likely*).

In relation to project duration, interestingly, respondents who worked with smaller companies (<200 employees) more often graded the effect of OSRs on *Bespoke projects*, *Market-driven projects* or *Outsourced projects* as *Likely* or even *Very likely*. The majority of the respondents who worked for companies with > 201 employees selected the *Somehow likely* answer for the *Bespoke projects* and *Market-driven projects*. This result confirms the previous analysis (Section 4.7.1) by indicating that size is not the only factor that impacts the seriousness of OSRs. It can also be speculated that the phenomenon of OSRs might be clearer in smaller organizations where less specialization makes outdated requirements more "everybody's concern", while in larger organizations, with high specialization, the view of "not my job" might play a factor (Berenbach et al, 2009).

4.8 Where in the requirements life cycle should OSRs be handled (RQ7)

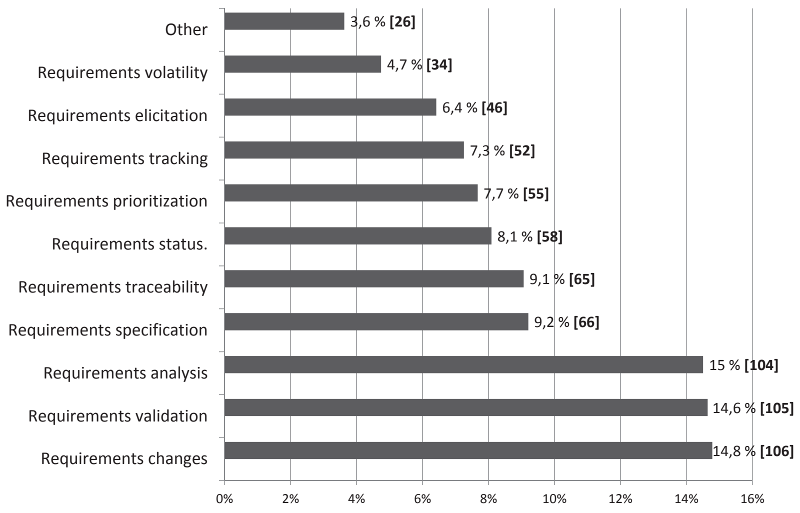


Figure 3.13: Requirements lifecycle stages for addressing OSRs

The results for this question are presented in Figure 13 as percentages of the total number of answers (717) since the question allowed multiple answers. The list of phases (or processes) in the requirements engineering lifecycle was inspired by Nurmuliani and Zowghi (2002). According to our respondents OSRs should first be handled during *Requirements analysis, Requirements validation* and *Requirements changes* phases (each with about 14% of the answers). This result is, to some extent in line with the study by Murphy and Rooney (2006), SWEBOK (IEEE Computer Society, 2004), and Nurmuliani and Zowghi (2002) who report that change leads to volatility, and volatility in its turn leads to obsolescence. However, less than 5% of the survey respondents indicate that OSRs should be managed as a part of chandling requirements volatility seems to support a distinction between volatility and the phenomenon of OSRs as such. That is, volatility may be related to OSRs; however, it needs to be handled continuously during analysis and validation as a part of change management in general.

The high numbers of answers given to *Requirements analysis* (14.5%) and *Requirements specification* (9.2%) phases confirm the suggestions made by Savolainen et al (2005) to manage OSRs in the requirements analysis phases. The low score in the *Requirements elicitation* phase answer (6.42% of all answers) contradicts the viewpoint of Merola (2006) who suggested managing obsolete software by continuous and timely market tracking and market trend change identification. This might seem to indicate that our respondents have difficulties understanding how OSRs could be managed, for example by finding and dismissing OSRs faster due to continuous elicitation depending on the accepted definition of OSRs.

Respondents working with *Agile software development* methodologies preferred to handle OSRs as a part of the *Requirements changes* phase, while respondents working in a *Waterfall* manner preferred the *Requirements validation* phase. This seems logical, as a part of agile methodology is to embrace change (Ramesh et al, 2010), while waterfall philosophy sees OSRs as something to be "handled" more formally in a development step (focusing on the specification and validation phases) (Kotonya and Sommerville, 1998).

Type of requirements engineering context (Figure 3.7) did not seem to significantly influence answers for this question. Requirements analysis, validation, and changes phases seemed to be dominant for *MDRE* and *Bespoke or contract driven requirements engineering* alike. However, looking at company size and project duration, respondents from larger companies with longer projects focused on handling OSRs in specific phases, i.e., analysis and validation. This result seems reasonable as large projects usually require more extensive requirements analysis due to, e.g., the larger number of stakeholders involved and possible higher complexity of the system to be developed (Regnell et al, 2008; Berenbach et al, 2009; Buhne et al, 2004).

Looking at the answers given in the *Other* category, four answers suggested that OSRs should be managed in all phases of software lifecycle: one answer suggested all requirements management phases and one suggested quality assurance. Further investigation is needed.

4.9 Existing processes and practices regarding managing OSRs (RQ5)

When queried about the existing processes and practices for managing OSRs, 73.6% of all respondents (159) indicated that their requirements engineering process does not take OSRs into consideration. This result can be interpreted as clear evidence of a lack of methods regarding OSRs in industry and confirms the need for developing methods for managing OSRs. At the same time, some processes for managing OSRs do exist, as indicated by 26.4% (57) of our respondents. The list of processes and methods used by our respondents include:

- Reviews of requirements and requirements specifications (19 respondents)
- Using tools and "marking requirements as obsolete" (6 respondents)
- Requirements traceability (6 respondents)
- Discussing and prioritizing requirements with customers in an agile context (4 respondents)
- "Mark obsolete requirements as obsolete" (4 respondents) — these respondents did not indicate if using a tool or not.
- During the requirements management process by identifying OSRs (3 respondents)
- Moving OSRs into a separated section in the SRS (3 respondents)
- Through a change management process (2 respondents)
- During the requirements analysis process (1 respondent)
- Having a proprietary process (1 respondent)

The identified "categories" of processes and methods above provide further support for previous results from the survey. For example, the process of managing OSRs by requirements reviews overlaps the most popular way to identify OSRs (Figure 3.11, Section 4.5), as indicated by our respondents. This would seem to indicate that manually reviewing requirements is dominant. Whether or not this is sufficient is another question which needs to be investigated further. The results confirm what was reported in Section 4.5, that automated methods for identification and management of OSRs are rare. Therefore, further research on scalable automatic methods for identification and management of OSRs is needed.

Some respondents provided names or descriptions of processes and methods used for managing OSRs. Those reported include:

- *Projective analysis through modeling* — Considered as a promising approach to study the complexity pertaining to systems of systems (Anderson et al, 2008), it requires a skilled "process modeler" to seamlessly use the modeling paradigm. If and how the method could be applied for smaller projects, and particularly for identification and management of OSRs remains an open question. Also, the technique is used during the requirements analysis phase which has been considered a good phase for management of OSRs by our respondents (Figure 3.13).

- *Hierarchical requirements' tables* — Specifying requirements on different abstraction levels is one of the fundamental techniques of requirements engineering that helps various stakeholders to understand requirements better (Lauesen, 2002). Considered as one of the requirements specification techniques, this could be promising according to our respondents (Figure 3.13). This method could be used to control OSRs to a certain degree as an overview of the requirements can be achieved, to some extent, through abstraction (Gorschek and Wohlin, 2005). However, given large numbers of requirements, scalability of the method could be a problem.

- *Project governance* — Support project control activities considering the environment in which project management is performed (Bekker and Steyn, 2008). By having greater time scope than ordinary project management, project governance could, according to our interpretation, be supportive in the task of continuous identification and management of OSRs.

- *Requirements tracking with risk management* — Although we consider tracking and risk management (Lauesen, 2002) as separated activities, combining them for the purpose of managing OSRs is an interesting alternative potential. In particular, the role of risk management in identification and management of OSRs should be further investigated, as the software risk management literature does not appear to mention OSRs as one of the software risks (Boehm, 1991).

- *Requirements-based test plans* — Aligning requirements with verification, although challenging, could be considered critical in assuring that the developed software fulfills customers' needs. Creating test plans based on requirements that are up-to-date and properly reflect changing customer needs is considered a best practice in software projects (Pohl, 2010). OSRs may create mismatches and problems with alignment between requirements and test cases. The discovery of a test result that was correct, however presently wrong, can indicate that a requirement has become obsolete. We are, however, uncertain to what degree the practice of writing test plans based on requirements could help in identification and management of OSRs. The fact that test plans are based on requirements is, to us, independent of the fact that these requirements may simply be obsolete.

- *Commenting out obsolete code and updating requirements documents accordingly* — This technique of managing OSRs could be considered promising and should help to keep the requirements aligned with the newest version of the code. However, the technique seems to only consider implemented requirements that could be directly traced to the code level. Given the fact that many requirements (especially quality requirements) are cross-cutting and require implementation in several places (Lauesen, 2002) in the source code, an OSR may become even more cross cutting than before. In our opinion, it could be challenging to correctly map changes in the code to changes in requirements. Thus, mapping change in the code to changes in requirements could be part of a solution; however, it lacks the possibility to identify and remove OSRs prior to implementation.

- *Using requirements validation techniques to identify if requirements are no longer needed* — Validating requirements is fundamental for assuring that the customer needs were properly and correctly understood by the development organization (Lauesen, 2002). In our opinion, this technique should be used together with customers who can confirm if the requirements are relevant. Our respondents also would like OSRs to be managed during requirements validation phase (Figure 3.13). However, if requirements reviews are conducted in isolation from "customers" by e.g., requirements analysts, they could have difficulties in identifying which requirements are, or are about to become, obsolete. This is further aggravated if the development organization operates in a MDRE context.

Looking at the context factors of organizational size, development methodology, and respondent role, although no statistically significant correlations could be observed, some interesting points warrant mentioning. Respondents from smaller companies (<50 employees) to a larger degree had explicit practices for handling OSRs as compared to respondents from larger companies. This seems reasonable when looking at the methods for managing OSRs provided, where manual review methods were most frequent. Quispire et al (2010) mentioned that processes used in small software enterprises are often manually based and less automated.

Respondents who worked with *MDRE* projects (Figure 3.7) reported having processes that take OSRs into consideration (34.3%), more often than respondents who worked with *Bespoke or contract driven requirements engineering* (26.5%) or *Outsourced projects* (15.8%) respectively (almost significant results with a p-value of 0.059, Table A.8a in (Wnuk, 2011a)). One possible explanation for this could be high and constant requirements influx in MDRE contexts (Regnell and Brinkkemper, 2005; Regnell et al, 2008), combined with frequent changes to requirements dictated by rapidly changing market situations. This in turn is resulting in more requirements becoming obsolete, forcing the use of methods to manage OSRs.

Further statistical tests (Table A.8 in (Wnuk, 2011a)) indicated a statistical significance between the roles of respondents and the existence of processes to manage OSRs (p = 0.0012). There was also a moderate association (Cramer's V = 0.345) between the respondents' roles and the existence of requirements engineering processes that take OSRs into account. From the cross-tabulation table between the respondents' roles and the existence of OSRs handling process (Table A.9 in (Wnuk, 2011a)) we can see that the respondents who worked in management positions (project and product managers) were more likely to utilize OSRs handling method compared to respondents who worked in software development roles, as developers.

Further, the presence of a method or process that considers OSRs seems to decrease the negative impact of OSRs among our respondents, as 50% of the respondents who deemed OSRs *Trivial* confirmed having a process of managing OSRs (Section 4.3). Moreover, as requirements engineers as well as product and project managers usually work more with requirements engineering related tasks than software development roles, it appears to be logical that more methods of managing OSRs are reported among the management roles.

4.10 Summary of results

The results from the study are summarized in the following points:

- Our respondents defined an OSR (RQ1) as: "a software requirement (implemented or not) that is no longer required for the current release or future releases, and it has no or little business value for the potential customers or users of a software artifact." This definition seems to be homogeneous among our respondents (with a small exception for the respondents who used *RUP* methodologies).

- OSRs constitute a significant challenge for companies developing software intensive products, with the possible exception of companies involved in the service domain. The phenomenon of OSRs is considered serious by 84.3% of our respondents (RQ2). At the same time 73.6% of our respondents reported having no process for handling obsolete software requirements (RQ5).

- Requirements related to standards and laws are the least likely to become obsolete, while inconsistent and ambiguous requirements are the most likely to become obsolete (RQ3). Moreover, requirements originating from domain experts were less likely to become obsolete than requirements originating from customers or (internal) developers.

- OSRs identification is predominantly a manual activity, and less than 10% of the respondents reported having any automated functionality. They also confirmed that automatic identification of OSR is difficult which suggests research opportunities in creating automated methods for OSR identification and management (RQ4).

- The identified OSRs should, according to more than 60% of the survey answers, be kept in the requirements document or the database, but tagged as obsolete. Deleting OSRs is not a desired behavior (RQ5). Most respondents opted for keeping the OSRs for purposes of reference and traceability, which seems to indicate that the identification of OSRs is not the only action, but a wish to potentially use the OSRs to minimize repeated work (e.g. specifying new requirements that are the same or similar to already identified OSRs). This is especially relevant in the MDRE context where "good ideas" can resurface as proposed by, for example internal developers.

- Although there exist some methods and tool support for the identification and handling of OSRs, a clear majority of the respondents indicated no use of methods or tools to support them. Rather, adhoc and manual process seemed to dominate (RQ5). Moreover, even when the identification of OSRs was deemed central (e.g., for respondents working in longer duration projects), only some tool support and automation was present (mostly for bespoke projects), but even here manual processes and routines dominated (Section 4.5).

- Project managers and product managers indicate that they always find OSRs in their work (Section 4.5), even if many of the respondents don't actively look for them.

- OSRs are more likely to affect *Large-scale requirements* and *Very large-scale requirements* projects (RQ6). Larger projects (hundreds of requirements) tend to have larger issues related to the presence of OSRs, and there seems to be a correlation between impact severity and project size (amount of requirements). OSRs seem to have a somewhat larger impact on projects in a MDRE context as compared to bespoke or contract driven development (Section 4.7.2). However, for very-large projects (over 10 000 requirements) all respondents, independent of context factors, agree that the potential impact of OSRs was substantial.

- According to the respondents, OSRs should first of all be handled during the *Requirements analysis* and *Requirements validation* phases (RQ7). At the same time, less than 5% of the answers indicate that

OSRs should be managed as a part of requirements volatility handling which supports the distinction between volatility and the phenomenon of OSRs as such. Finally, our respondents suggested that *Requirements elicitation* is not the best phase to manage OSRs.

- Latency may not be the main determinant of OSRs becoming a problem. Rather, the results point to the lack of methods and routines for actively handling OSRs as a central determinant. This would imply that claimed low latency development models, like agile, has and can have problems with OSRs.

5 Conclusions and Further Work

Although the phenomenon of obsolete software requirements and its negative effects on project timelines and the outcomes have been reported in a number of publications (Hood et al, 2008b; Murphy and Rooney, 2006; Stephen et al, 2011; Merola, 2006; Cao and Ramesh, 2008), little empirical evidence exists that explicitly and exhaustively investigates the phenomenon of OSRs.

In this paper, we report results from a survey conducted among 219 respondents from 45 countries exploring the phenomenon of OSRs by: (1) eliciting a definition of OSRs as seen by practitioners in industry, (2) exploring ways to identify and manage OSRs in requirements documents and databases, (3) investigating the potential impact of OSRs, (4) exploring effects of project context factors on OSRs, and (5) defining what types of requirements are most likely to become obsolete.

Our results clearly indicate that OSRs are a significant challenge for companies developing software systems — OSRs were considered serious by 84.3% of our respondents. Moreover, a clear majority of the respondents indicated no use of methods or tools to support identification and handling OSRs, and only 10% of our respondents reported having automated support. This indicates that there is a need for developing automated methods or tools to support practitioners in the identification and management of OSRs. These proposed methods need to have effective mechanisms for storing requirements tagged as OSRs, enabling the use of the body of OSRs as decision support for future requirements and their analysis. This could potentially enable automated regression analysis of active requirements, continuously identifying candidates for OSRs, and flagging them for analysis.

Although manually managing OSRs is currently the dominant procedure, which could be sufficient in small projects, research effort should be directed towards developing scalable methods for managing OSRs — methods that scale to a reality that is often characterized by large numbers of requirements and a continuous and substantial influx of new requirements. The reality facing many product development organizations developing software intensive systems today is that OSRs are a problem, and as the amount and complexity of software increases so will the impact of OSRs.

Bibliography

Anderson W, Boxer PJ, Brownsword L (2008) An examination of a structural modeling risk probe technique. Tech. Rep. CMU/SEI-2006-SR-017, Software Engineering Institute, Carnegie Mellon University

Anthes G (1994) No more creeps! are you a victim of creeping user requirements? Computerworld 28(18):107 – 110

Arcuri A, Briand L (2011) A practical guide for using statistical tests to assess randomized algorithms in software engineering. In: Proceeding of the 33rd International Conference on Software Engineering, ACM, New York, NY, USA, ICSE '11, pp 1–10

Aurum A, Wohlin C (2005a) Engineering and Managing Software Requirements. Springer-Verlag New York, Inc., Secaucus, NJ, USA

Aurum A, Wohlin C (2005b) Requirements engineering: Setting the context. In: Aurum A, Wohlin C (eds) Engineering and Managing Software Requirements, Springer Berlin Heidelberg, pp 1–15

Bano M, Ikram N (2010) Issues and challenges of requirement engineering in service oriented software development. In: Proceedings of the Fifth International Conference on Software Engineering Advances (ICSEA), 2010, pp 64 –69

Bekker M, Steyn H (2008) The impact of project governance principles on project performance. In: Proceedings of the Portland International Conference on Management of Engineering Technology, 2008. PICMET 2008., pp 1324 –1330

Berenbach B, Paulish DJ, Kazmeier J, Rudorfer A (2009) Software & Systems Requirements Engineering: In Practice. Pearson Education Inc.

Boehm B (1991) Software risk management: principles and practices. Software, IEEE 8(1):32 –41

Boehm B (2000) Requirements that handle ikiwisi, cots, and rapid change. Computer 33(7):99–102

Bourque P, Dupuis R (2004) Guide to the software engineering body of knowledge 2004 version. SWEBOK

Buhne S, Halmans G, Pohl K, Weber M, Kleinwechter H, Wierczoch T (2004) Defining requirements at different levels of abstraction. In: Proceedings of the 12th IEEE International Requirements Engineering Conference, pp 346 – 347

Cao L, Ramesh B (2008) Agile requirements engineering practices: An empirical study. Software, IEEE 25(1):60 –67

Chen J, Reilly R, Lynn G (2005) The impacts of speed-to-market on new product success: the moderating effects of uncertainty. IEEE Transactions on Engineering Management 52(2):199 – 212

Curtis W, Krasner H, Shen V, Iscoe N (1987) On building software process models under the lamppost. In: Proceedings of the 9th international conference on Software Engineering (ICSE 1987), pp 96–103

Dawson C (2005) Projects in Computing and Information Systems: A Student's Guide. Addison Wesley

DeBellis M, Haapala C (1995) User-centric software engineering. IEEE Expert 10(1):34 –41

DeMarco T, Lister T (2003) Risk management during requirements. IEEE Software 20(5):99–101

Easterbrook S (2004) What is requirements engineering? `http://www.cs.toronto.edu/~sme/papers/2004/FoRE-chapter01-v7.pdf`

Easterbrook S, Singer J, Storey MA, Damian D (2008) Selecting empirical methods for software engineering research. In: Shull F, Singer J, Sjøberg D (eds) Guide to Advanced Empirical Software Engineering, Springer London, pp 285–311

Finkelstein A, Gabbay D, Hunter A, Kramer J, Nuseibeh B (1994) Inconsistency handling in multiperspective specifications. IEEE Trans Softw Eng 20(8):569–578

Gorschek T, Davis A (2008) Requirements engineering: In search of the dependent variables. Inf Softw Technol 50:67–75

Gorschek T, Wohlin C (2005) Requirements abstraction model. Requir Eng 11:79–101

Gorschek T, Garre P, Larsson S, Wohlin C (2007a) Industry evaluation of the requirements abstraction model. Requir Eng 12:163–190

Gorschek T, Svahnberg M, Borg A, Loconsole A, Börstler J, Sandahl K, Eriksson M (2007b) A controlled empirical evaluation of a requirements abstraction model. Inf Softw Technol 49:790–805

Gorschek T, Fricker S, Palm K, Kunsman S (2010) A lightweight innovation process for software-intensive product development. Software, IEEE 27(1):37 –45

Gulk G, Verhoef C (2008) Quantifying requirements volatility effects. Sci Comput Program 72(3):136–175

Harker S, Eason K, Dobson J (1993) The change and evolution of requirements as a challenge to the practice of software engineering. In: Proceedings of IEEE International Symposium on Requirements Engineering, pp 266 –272

Herald T, Verma D, Lubert C, Cloutier R (2009) An obsolescence management framework for system baseline evolution perspectives through the system life cycle. Syst Eng 12:1–20

Holmstrom H, Conchuir E, Agerfalk PJ, Fitzgerald B (2006) Global software development challenges: A case study on temporal, geographical and socio-cultural distance. In: Proceedings of the International Conference on Global Software Engineering, ICGSE ’06., pp 3 –11

Hood C, Wiedemann S, Fichtinger S, Pautz U (2008a) Change management interface. In: Requirements Management, Springer Berlin Heidelberg, pp 175–191

Hood C, Wiedemann S, Fichtinger S, Pautz U (2008b) Requirements Management The Interface Between Requirements Development and All Other Systems Engineering Processes. Springer, Berlin

Hood C, Wiedemann S, Fichtinger S, Pautz U (2008c) Requirements Management: The Interface Between Requirements Development and All Other Systems Engineering Processes. Springer-Verlag Berin

Houston DX, Mackulak GT, Collofello JS (2001) Stochastic simulation of risk factor potential effects for software development risk management. Journal of Systems and Software 59(3):247 – 257

Iacovou C, Dexter A (2004) Turning around runaway information technology projects. Engineering Management Review, IEEE 32(4):97 –112

IBM (2011) The description of the method can be found at. `http://www-01.ibm.com/software/awdtools/rup/`

IEEE (1997) IEEE recommended practice for software requirements specifications, 830-1998. `http://standards.ieee.org/findstds/standard/830-1998.html`

IEEE Computer Society (2004) Software Engineering Body of Knowledge (SWEBOK). Angela Burgess, EUA, URL `http://www.swebok.org/`

Institute SE (2011) Capability maturity model integration (cmmi), version 1.3. `http://www.sei.cmu.edu/cmmi/solutions/info-center.cfm`

Kabbedijk J, K Wnuk BR, Brinkkemper S (2010) What decision characteristics influence decision making in market-driven large-scale software product line development? In: Proceedings of the Product Line Requirements Engineering and Quality workshop 2010, pp 42 –53

Kohl R (2001) Changes in the requirements engineering processes for cots-based systems. IEEE Computer Society, Los Alamitos, CA, USA, vol 0, p 0271

Konrad S, Gall M (2008) Requirements engineering in the development of large-scale systems. In: Proceedings of the 2008 16th IEEE International Requirements Engineering Conference, IEEE Computer Society, Washington, DC, USA, RE '08, pp 217–222

Kossmann M, Gillies A, Odeh M, Watts S (2009) Ontology-driven requirements engineering with reference to the aerospace industry. In: Proceedings of the Second International Conference on the Applications of Digital Information and Web Technologies, ICADIWT '09, pp 95 –103

Kotonya G, Sommerville I (1998) Requirements Engineering. John Wiley & Sons

L M Rea RP (1005) Designing and Conducting Survey Research: A Comprehensive Guide. Jossey-Bass, San Francisco, CA, 94103-1741

Lamsweerde A (2009) Requirements Engineering: From System Goals to UML Models to Software Specifications. John Wiley

Lauesen S (2002) Software Requirements – Styles and Techniques. Addison–Wesley

Legodi I, Barry M (2010) The current challenges and status of risk management in enterprise data warehouse projects in south africa. In: Proceedings of the Technology Management for Global Economic Growth (PICMET), pp 1 –5

Lethbridge T, Sim E, Janice J (2005) Studying software engineers: Data collection techniques for software field studies. Empirical Software Engineering 10:311–341

Linkedin (2011) The linkedin website. `http://www.linkedin.com/`

Loconsole A, Borstler J (2005) An industrial case study on requirements volatility measures. In: Asia-Pacific Software Engineering Conference, pp 1–8

Loesch F, Ploederoeder E (2007) Restructuring variability in software product lines using concept analysis of product configurations. In: Proceedings of the 11th European Conference on Software Maintenance and Reengineering, CSMR '07., pp 159 –170

Mannion M, Lewis O, Kaindl H, Montroni G, Wheadon J (2000) Representing requirements on generic software in an application family model. In:

Proceedings of the 6th International Conerence on Software Reuse: Advances in Software Reusability, Springer-Verlag, London, UK, pp 153–169

McGee S, Greer D (2009) A software requirements change source taxonomy. In: Proceedings of the Fourth International Conference on Software Engineering Advances, ICSEA '09., pp 51 –58

Merola L (2006) The cots software obsolescence threat. In: Proceedings of the Fifth International Conference on Commercial-off-the-Shelf (COTS)-Based Software Systems, p 7 pp.

Monkey S (2011) Survey monkey webpage. `http://www.surveymonkey.net`

Murphy D, Rooney D (2006) Investing in agile: Aligning agile initiatives with enterprise goals. Cutter IT Journal 19(2):6 –13

Nakagawa S (2004) A farewell to Bonferroni: the problems of low statistical power and publication bias. Behavioral Ecology 15(6):1044–1045

Natt och Dag J, Thelin T, Regnell B (2006) An experiment on linguistic tool support for consolidation of requirements from multiple sources in market-driven product development. Empirical Softw Engg 11:303–329

Nurmuliani N, Zowghi D, Powell S (2004) Analysis of requirements volatility during software development life cycle. In: Proceedings of the 2004 Australian Software Engineering Conference, pp 28 – 37

Pohl K (2010) Requirements Engineering: Fundamentals, Principles, and Techniques, 1st edn. Springer Publishing Company, Incorporated

Quispe A, Marques M, Silvestre L, Ochoa S, Robbes R (2010) Requirements engineering practices in very small software enterprises: A diagnostic study. In: XXIX International Conference of the Chilean Computer Science Society (SCCC), 2010, pp 81 –87

Ramesh B, Cao L, Baskerville R (2010) Agile requirements engineering practices and challenges: an empirical study. Inf Syst J 20(5):449–480

Regnell B, Brinkkemper S (2005) Market-driven requirements engineering for software products. In: Aurum A, Wohlin C (eds) Engineering and Managing Software Requirements, Springer Berlin Heidelberg, pp 287–308

Regnell B, Svensson RB, Wnuk K (2008) Can we beat the complexity of very large-scale requirements engineering? In: Proceedings of the 14th international conference on Requirements Engineering: Foundation for Software Quality, Springer-Verlag, Berlin, Heidelberg, REFSQ '08, pp 123–128

Robertson S, Robertson J (1999) Mastering the requirements process. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA

Robinson W, Pawlowski D (1999) Managing requirements inconsistency with development goal monitors. IEEE Transactions on Software Engineering 25(6):816–835

Ruel H, Bondarouk T, Smink S (2010) The waterfall approach and requirement uncertainty: An in-depth case study of an enterprise systems implementation at a major airline company. Int J Technol Proj Manage (USA) 1(2):43 – 60

Russo A, Nuseibeh B, Kramer J (1998) Restructuring requirements specifications for inconsistency analysis: A case study. In: Third IEEE International Conference on Requirements Engineering, IEEE Computer Society Press, pp 51–60

Savolainen J, Oliver I, Mannion M, Zuo H (2005) Transitioning from product line requirements to product line architecture. In: Proceedings of the 29th Annual International Computer Software and Applications Conference, COMPSAC 2005., vol 1, pp 186 – 195 Vol. 2

Sawyer P (2000) Packaged software: Challenges for re. In: Proceedings of the Sixth International Workshop on Requirements Engineering: Foundations of Software Quality (REFSQ 2000), pp 137–142

Shan X, Jiang G, Huang T (2010) The study on knowledge transfer of software project requirements. In: 2010 International Conference on Biomedical Engineering and Computer Science (ICBECS), pp 1 –4

Siegel S, Castellan N (1998) Nonparametric statistics for the behavioral sciences, 2nd edn. McGraw-Hill

Singer J, Sim SE, Lethbridge TC (2008) Software engineering data collection for field studies. In: Shull F, Singer J, Sjøberg DIK (eds) Guide to Advanced Empirical Software Engineering, Springer London, pp 9–34

Sommerville I (2007) Software Engineering. Addison–Wesley

Sommerville I, Sawyer P (1997) Requirements Engineering: A Good Practice Guide. John Wiley & Sons

Stephen J, Page J, Myers J, Brown A, Watson D, Magee I (2011) System error fixing the flaws in government it. Tech. Rep. 6480524, Institute for Government, London

Strauss A, Corbin J (1990) Basics of Qualitative Research: Grounded Theory Procedures and Techniques. Sage Publications, Newbury Park, California

Takahashi M, Kamayachi Y (1989) An empirical study of a model for program error prediction. IEEE Transactions on Software Engineering 15:82–86

Šmite D, Wohlin C, Gorschek T, Feldt R (2010) Empirical evidence in global software engineering: a systematic review. Empirical Softw Eng 15:91–118

Wiegers KE (2003) Software Requirements, Second Edition. Microsoft Press, Redmond, WA, USA

Wikipedia (2011) Likert scale. `http://en.wikipedia.org/wiki/Likert\_scale`

Wnuk K (2011a) The appendix with analysis can be accessed at. `http://fileadmin.cs.lth.se/serg/ExperimentPackages/Obsolete/AppendixA\_Analysis.pdf`

Wnuk K (2011b) The full list of countries can be obtained at. `http://fileadmin.cs.lth.se/serg/ExperimentPackages/Obsolete/COUNTRIES.pdf`

Wnuk K (2011c) The full list of mailing lists can be accessed at. `http://fileadmin.cs.lth.se/serg/ExperimentPackages/Obsolete/ListOfDiscussionGroups.pdf`

Wnuk K (2011d) The full list of tool vendors can be accessed at. `http://fileadmin.cs.lth.se/serg/ExperimentPackages/Obsolete/ListOfVendors.pdf`

Wnuk K (2011e) The survey questionnaire. `http://fileadmin.cs.lth.se/serg/ExperimentPackages/Obsolete/AppendixB\_SurveyQuestions.pdf`

Wnuk K, Regnell B, Karlsson L (2009) What happened to our features? visualization and understanding of scope change dynamics in a large-scale industrial setting. In: Proceedings of the 17th IEEE International Requirements Engineering Conference, RE '09, pp 89 –98

Wohlin C, Xie M, Ahlgren M (1995) Reducing time to market through optimization with respect to soft factors. In: Proceedings of the 1995 IEEE Annual Engineering Management Conference, 'Global Engineering Management: Emerging Trends in the Asia Pacific'., International, pp 116 –121

Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslén A (2000) Experimentation in software engineering: an introduction. Kluwer Academic Publishers, Norwell, MA, USA

Zowghi D, Nurmuliani N (2002) A study of the impact of requirements volatility on software project performance. Asia-Pacific Software Engineering Conference 0:3

Paper IV

Are You Biting Off More Than You Can Chew? A Case Study on Causes and Effects of Overscoping in Large–Scale Software Engineering

Krzysztof Wnuk, Elizabeth Bjarnason, ,Björn Regnell
Department of Computer Science,
Lund University, Sweden
{Krzysztof.Wnuk,Elizabeth.Bjarnason,Bjorn.Regnell}@cs.lth.se

Information and Software Technology,
Volume 54 Issue 10, October, 2012, Pages 1107-1124

Abstract

Context: Scope management is a core part of software release management and often a key factor in releasing successful software products to the market. In a market-driven case, when only a few requirements are known a priori, the risk of overscoping may increase. Objective: This paper reports on findings from a case study aimed at understanding overscoping in large-scale, market-driven software development projects, and how agile requirements engineering practices may affect this situation. Method: Based on a hypothesis of which factors that may be involved in an overscoping situation, semi-structured interviews were performed with nine practitioners at a large, market-driven software company. The results from the interviews were validated by six (other) practitioners at the case company via a questionnaire. Results: The results provide a detailed picture of overscoping as a phenomenon including a number of causes, root causes and effects, and indicate that overscoping is mainly caused by operating in a fast-moving market-driven domain and how this ever-changing inflow of requirements is managed. Weak awareness of overall goals, in combination with low development involvement in early phases, may contribute to 'biting off' more than a project can 'chew'. Furthermore, overscoping may lead to a number of potentially serious and expensive consequences, including quality issues, delays and failure to meet customer expectations. Finally, the study indicates that overscoping occurs also when applying agile requirements engineering practices, though the overload is more manageable and perceived to result in less wasted effort when applying a continuous scope prioritization, in combination with gradual requirements detailing and a close cooperation within cross-functional teams. Conclusion: The results provide an increased understanding of scoping as a complex and continuous activity, including an analysis of the causes, effects, and a discussion on possible impact of agile requirements engineering practices to the issue of overscoping. The results presented in this paper can be used to identify potential factors to address in order to achieve a more realistic project scope

1 Introduction

Maximizing the business value for a product and a set of available resources may sound like a simple task of selecting features according to the highest return of investment. However, in market-driven requirements engineering (MDRE) (Karlsson et al, 2007a; Regnell and Brinkkemper, 2005) software product managers face the challenge of managing continuously shifting market needs (Abramovici and Sieg, 2002) with a large number of new and changing requirements (Gorschek and Wohlin, 2006) caused both by a capricious market situation (DeBaud and Schmid, 1999) and by evolving technologies. In this situation, selecting which requirements to include into the next release of a software product, also called *scoping* (Schmid, 2002) or project scoping (Project Management Institute, 2000), is a complex and continuous task of assessing and re-assessing how these scoping changes impact the common code base of the software product line (Pohl et al, 2005) on which those products are built (Wnuk et al, 2009). This domain scoping is considered part of the product line scoping (Schmid, 2002), which derives value from the opportunities to reuse functionality of the product line. These factors, combined with increased market com- petition and unpredictable market response to new products, force decision makers to continuously face the task of making and re-evaluating decisions in an ever evolving world (Aurum and Wohlin, 2003).

Defining the scope of a product to fit a required schedule is a known risk in project management (Boehm, 1989) and in our previous work (Wnuk et al, 2009) we found that the project scope at a large software company changed significantly throughout the entire project life cycle. These changes were partly due to overscoping, i.e. setting a scope that requires more resources than are available. Several researchers have focused on scope creep where the scope is increased by the developers, and highlighted this as a serious project risk (Carter et al, 2001; Crockford, 1980; Iacovou and Dexter, 2004). Others have investigated scoping as a part of release planning (Schmid, 2002; Svahnberg et al, 2010; Wnuk et al, 2009). However, no study has yet attempted to investigate the causes and effects of overscoping even though requirements engineering (RE) decision making is an acknowledged challenge (Alenljung and Persson, 2008; Aurum and Wohlin, 2003; Ngo-The and Ruhe, 2005). In this study, we have investigated this phenomenon of overscoping a project, or biting off more that you can chew, in particular in a market-driven and very-large scale RE (VLSRE) context (Regnell et al, 2008).

Agile development processes claim to address several of the challenges involved in scoping frequently changing requirements. For example, in eXtreme programming (XP) (Beck, 1999) and Scrum (Schwaber and Beedle, 2002) the balance between scope and available resources is managed by extreme prioritization and constant (re)planning of the scope in combination with time boxing of the individual development iterations. However, agile requirements engineering (RE) practices have also been found to pose new challenges, e.g., in achieving consensus on priorities among multiple stakeholders and in creating accurate project plans (cost and timeline) for an entire project (Ramesh et al, 2010).

The main goal of the case study reported on in this paper was to increase the understanding of factors involved in overscoping and thereby highlight this risk and take a step towards addressing and avoiding overscoping of projects. To achieve this, the study was designed to answer the following questions:

- (RQ1) What causes over- scoping?
- (RQ2) What are the resulting effects of overscoping?
- (RQ3) How may agile RE practices impact the causes and effects of overscoping?

The case study has been conducted at a large market-driven software development company that has started to shift towards a more agile way of working. The study includes interviews with nine practitioners working with requirements engineering, software development and product testing. The interview results were then validated via a questionnaire with another six practitioners from the case company. The contribution of the presented work includes eight main causes of overscoping complemented by a number of root causes, and nine main effects of overscoping. In addition, the results indicate that three of the agile RE practices adopted by the case company may impact some of these causes and root causes and, thus, may also reduce the effects of overscoping.

Partial results from this study have previously been published as workshop publications in (Bjarnason et al, 2010) where overscoping was preliminarily investigated and in (Bjarnason et al, 2011a) where preliminary results on the benefits and side effects of agile RE practices were published. For this article, the results are extended with (1) additional causes, root causes and effects of overscoping; (2) additional empirical results on overscoping from 6 (other) practitioners; and (3) details on the impact of agile RE practices specifically on overscoping. These extensions were achieved by further analysis of the full interview material and further validation of the results through a questionnaire.

The remainder of this paper is structured as follows: Section 2 describes related work. Section 3 describes the case company, while the research method is outlined in Section 4. The results are reported in Section 5 for the interviews and in Section 6 for the validation questionnaire. In Section 7, the results are interpreted and related to other work, and limitations and validity threats are discussed. Finally, Section 8 contains conclusions and further work.

2 Related work

Unrealistic schedules and budgets are among the top ten risks in software engineering (Boehm, 1989) and some reasons for overloading projects with scope have been suggested. For example, DeMarco and Lister (2003) mentioned that a failure among stakeholders to concur on project goals (also one of the challenges of agile RE (Ramesh et al, 2010)) can result in an excessive scope burden on a project. Project overload may also result from sales staff agreeing to deliver unrealistically large features without considering scheduling implications (Hall et al, 2002). Furthermore, scope that is extended beyond the formal requirements by the developers, i.e. scope creep, is stated by Iaconou and Dexter (Iacovou and Dexter, 2004) as a factors leading to project failures. Scope creep is also mentioned as having a big impact on risk and risk management in enterprise data warehouse projects (Legodi and Barry, 2010). In addition, it is listed as one of five core risks during the requirements phase, and is a direct consequence of how requirements are gathered (DeMarco and Lister, 2003). On the other hand,Gemmer (1997) argues that people's perceptions of risk and their subsequent behavior is overlooked within risk management and that an increased awareness of causes and effects of risks may lead to an improved discussion and management of these risks (Gemmer, 1997). Some methods and tools to mitigate and manage risks related to scoping have been presented (Crockford, 1980). For example, Carter et al (2001) suggested combining evolutionary prototyping and risk-mitigation strategy to mitigate the negative effects of scope creep. However, the full issue of overscoping is not explicitly named as a risk in the related work, nor empirically investigated for their causes and consequences.

Requirements engineering (RE) is a decision intense part of the software engineering process (Aurum and Wohlin, 2003), which can support and increase the probability of success in the development process (Aurum and Wohlin, 2005a). However, the efficiency and effectiveness of RE decision making has cognitive limitations (Aurum and Wohlin, 2003), due to being a knowledge intensive activity. Furthermore, research into the field of RE decision making is still in its infancy (Alenljung and Persson, 2008; Ngo-The and Ruhe, 2005). A major challenge in this research (according to Alenljung and Persson) lies in understanding the nature of RE decision making and identifying its obstacles (Alenljung and Persson, 2008) and several authors (Alenljung and Persson, 2008; Aurum and Wohlin, 2003, 2005a; Ngo-The and Ruhe, 2005) mention the need to: (1) identify decision problems in the RE process; (2) perform empirical studies of RE decision making; and (3) examine how non-technical issues affect or influence decision making. Communication gaps are an example of such non-technical issues which have been reported to negatively affect the decision making and contribute to defining an unrealistic scope (Bjarnason et al, 2011b).

There are two characteristics of MDRE (Regnell and Brinkkemper, 2005) which further aggravates RE decision making, namely a lack of actual customers with which to negotiate requirements (Karlsson and Ryan, 1997; Potts, 1995) and a continuous inflow of requirements from multiple channels (Karlsson et al, 2007a; Gorschek and Wohlin, 2006). As a result, rather than negotiate with specific customers, the demands and requirements of an anonymous consumer market have to be 'invented' (Potts, 1995) through market research. Moreover, the success of the final product is primarily validated by the market with the uncertainty (Regnell and Brinkkemper, 2005) of how well the 'invented' requirements compare to the market needs. Commonly, market research continuously issues more requirements (Regnell and Brinkkemper, 2005) than can be handled with available resources. A state of congestion then occurs in the MDRE process (Karlsson et al, 2007a) and the set of requirements to implement in the next project has to be selected using prioritization techniques based on market predictions and effort estimates (Carlshamre, 2002; Karlsson and Ryan, 1997; Jørgensen and Shepperd, 2007).

Scope management is considered as one of the core functions of software release planning and a key activity for achieving economic benefits in product line development (Schmid, 2002). Accurate release planning is vital for launching products within the optimal market window. And, this is a critical success factor for products in the MDRE domain (Sawyer, 2000). Missing this window might cause both losses in sales and, additional cost for prolonged development and delayed promotion campaigns. However, making reliable release plans based on uncertain estimates (Karlsson et al, 2007a) and frequently with features with dependencies to other features (Carlshamre et al, 2001) is a challenge in itself. In addition, a rapidly changing market situation may force a project to consider new market requirements at a late project stage. Release planning is then a compromise where already committed features may need to be sacrificed at the expense of wasted effort (Wnuk et al, 2009) of work already performed. The area of release planning is well researched and Svahnberg et al reported on 24 strategic release planning models presented in academic papers intended for market-driven software development (Svahnberg et al, 2010). Furthermore, Wohlin and Aurum investigated what is important when deciding to include a software requirement in a project or release (Wohlin and Aurum, 2005). Despite this, the understanding of the challenges related to scope management and their causes and effects is still low.

Scoping in agile development projects mainly involves three of the agile RE practices identified by Ramesh et al (2010), namely *extreme prioritization*, *constant planning* and *iterative RE*. High-level requirements are prioritized and the features with the highest market value are developed first. This approach ensures that if the project is delayed launch may still be possible since the most business-critical requirements will already be developed. Ramesh et al (2010) identified a number of benefits for companies applying these agile RE practices, but also challenges and varying impact on project risks. The identified benefits include an ability to adapt to changing prioritization of requirements, as well as, a clearer understanding of what the customers want, thus reducing the need for major changes. On the other hand, agile RE practices were found to include challenges in (1) correctly estimating and scheduling for the full project scope (which continuously changes), (2) a tendency to omit quality requirements and architectural issues (with the risk of serious and costly problems over time), and (3) constant re-prioritization of the requirements (with subsequent instability and waste) (Ramesh et al, 2010).

3 The case company

The case company has around 5000 employees and develops embedded systems for a global market using a product line approach (Pohl et al, 2005). The projects in focus for this case study are technology investments into an evolving common code base of a product line (a.k.a. platform) on which multiple products are based. There are several consecutive releases of this platform where each release is the basis for one or more products. The products mainly reuse the platform's functionality and qualities, and contain very little product-specific software. A major platform release has a lead time of approximately 2 years from start to launch, and is focused on functionality growth and quality enhancements for a product portfolio. For such projects typically around 60-80 new features are added, for which approximately 700-1000 system requirements are produced. These requirements are then implemented by 20-25 different development teams with, in total, around 40-80 developers per team assigned to different projects. The requirements legacy database amounts to a very complex and large set of requirements, at various abstraction levels, in the order of magnitude of 20,000 entities. This makes it an example of the VLSRE (very-large scale RE) context (Regnell et al, 2008). Both the flow of new requirements (added to and removed from the scope of platform projects) and the scoping decisions associated with this ?ow may change frequently and rapidly. This exposes the project management to a series of unplanned, and often difficult, decisions where previous commitments have to be changed or canceled.

3.1 Organisational set-up

Several organizational units within the company are involved in the development. For this case study, the relevant units are: *the requirements unit* that manages the scope and the requirements; *the software unit* that develops the software for the platform; and the product unit that develops products based on the platform releases. In addition, there is a usability design unit responsible for designing the user interface. Within each unit there are several groups of specialists divided by technical area. These specialists are responsible for the work in various stages of the development process. For this study, the most essential groups are the requirements teams (RTs) (part of the requirements unit) that, for a specific technical area, define the scope, and elicit and specify system requirements, and the development teams (DTs) (part of the software unit) that design, develop and maintain software for the (previously) defined requirements. Each RT has a team leader who manages the team. Another role belonging to the requirements unit is the requirements architect who is responsible for managing the overall scope, which includes coordinating the RTs. In the DTs there are several roles, namely

- Development team leader who leads and plans the team's work for the implementation and maintenance phases;
- Development team requirements coordinator who leads the team's work during the requirements management and design phase, and coordinates the requirements with the RTs;
- Developer who designs, develops and maintains the software;
- Tester who verifies the software.

The software unit also has a project management team consisting of (among others): quality managers who set the target quality levels and follow up on these, and software project managers who monitor and coordinate the DTs and interact with the requirements architects. For the product development unit in this study, we focus on the system testing task from the viewpoint of the functionality and quality of the platform produced by the software unit.

3.2 Phase-based process

The company used a stage-gate model. There were milestones (MS) for controlling and monitoring the project progress. In particular, there were four milestones for the requirements management and design (MS1-MS4) and three milestones for the implementation and maintenance (MS5-MS7). For each of these milestones, the project scope was updated and baselined. The milestone criteria were as follows:

- *MS1:* At the beginning of each project, RT roadmap documents were extracted to formulate a set of features for an upcoming platform project. A feature in this case is a concept of grouping requirements that constitute a new functional enhancement to the platform. At this stage, features contained a description sufficient for enabling estimation of its market value and implementation effort, both of which were obtained using a cost-value approach (Karlsson and Ryan, 1997).

These values were the basis for initial scoping inclusion for each technical area when the features were reviewed, prioritized and approved. The initial scope was decided and baselined per RT, guided by a project directive and based on initial resource estimates given by the primary receiving (main) DT. The scope was then maintained in a document called feature list that was updated each week after a meeting of the change control board (CCB). The role of the CCB was to decide upon adding or removing features according to changes that occur.

- *MS2:* The features were refined to requirements and specified by the RTs, and assigned to their main DTs, responsible for designing and implementing the feature. The requirements were reviewed with the main DTs and were then approved. Other (secondary) DTs that were also affected by the features were identified. The DTs make an effort estimate per feature for both main and secondary DT.
- *MS3:* The DTs had refined the system requirements and started designing the system. The set of secondary DTs were refined along with the effort estimates, and the scope was updated and baselined.
- *MS4:* The requirements refinement work and the system design were finished, and implementation plans were produced. The final scope was decided and agreed with the development resources, i.e. the software unit.
- *MS5:* All requirements had been developed and delivered to the platform.
- *MS6:* The software in the platform had been stabilized and prepared for customer testing.
- *MS7:* Customer-reported issues had been handled and the software updated. The software was ready to be released.

According to the company's process guidelines, the majority of the scoping work should have been done by MS2. The requirements were expressed using a domain-specific, natural language, and contained many special terms that required contextual knowledge to be understood. In the early phases, requirements contained a customer-oriented description while later being refined to detailed implementation requirements.

3.3 Agile development process

In order to meet the challenges of managing high requirements volatility, the case company was introducing a new development process at the time of this study. The size and complexity of the software development, including the usage of product lines, remained the same irrespective of the process used. The new process has been influenced by ideas and principles from the agile development processes eXtreme programming (XP) (Beck, 1999) and Scrum (Schwaber and Beedle, 2002). The phase-based process was replaced by a continuous development model with a toll-gate structure for the software releases of the software product line (to allow for coordination with hardware and product projects, see P1 below). The responsibility for requirements management was transferred from the (previous) requirements unit, partly into the business unit and partly into the software unit. The following agile RE practices were being introduced:

- *One continuous scope and release-planning flow (P1).* The scope for all software releases is continuously planned and managed via one priority-based list (comparable to a product backlog). The business unit gathers and prioritizes features from a business perspective. All new high-level requirements are continuously placed into this list and prioritized by the business unit. The software unit estimates the effort and potential delivery date for each feature based on priority and available software resource capacity. Development is planned and executed according to priority order. Planning and resource allocation is handled via one overall plan which contains all the resources of the software unit. The scope of the platform releases are synchronized with the product releases by gradual commitment to different parts of the scope. Critical scope is requested to be committed for specific product releases, while non-critical features are assigned to product releases when they are implemented and ready to be integrated into the platform.

- *Cross-functional development teams (P2)* that include a customer representative assigned by the business unit (comparable to the agile practice of customer proxy) have the full responsibility for defining detailed requirements, implementing and testing a feature (from the common priority-based list). Each new feature is developed by one cross-functional team specifically composed for that feature. The different software engineering disciplines and phases (e.g. requirements, design and test) are performed in an integrated fashion and within the same team. The team has the mandate to decide on changes within the value, effort and time frames assigned for the feature.

- *Gradual and iterative detailing of requirements* (P3). The requirements are first defined at the high level (as features in the priority-based list) and then iteratively refined in the development teams into more detailed requirements as the design and implementation work progresses.

- *Integrated requirements engineering* (P4). The requirements engineering tasks are integrated with the other development activities. The requirements are detailed, agreed and documented during design and development within the same cross-functional development team through close interaction between the customer representative and other team members, e.g. designers, developers and testers.

- *User stories and acceptance criteria* (P5) are used to formally document the requirements agreed for development. User stories define user goals and the acceptance criteria define detailed requirements to fulfill for a user story to be accepted as fully implemented. The acceptance criteria are to be covered by test cases.

This study mainly focuses on the situation prior to introducing the new agile way of working, i.e. for projects working as described in Section 3.2. The agile RE practices covered in this paper were defined in the company's internal development process at the time of the study. Practices P1 and P2 were being used in the projects, while P3 was partly implemented, and P4 and P5 were in the process of being implemented. Thus, it was not possible to investigate the full impact of the new agile RE practices at the time of this study. Nevertheless, the study investigates how these (new) practices are believed to affect the overscoping situation, i.e. which causes and root causes may be impacted by the agile RE practices and, thus, lead to reducing overscoping and its effects.

4 Research method

The study was initiated due to a larger transition taking place within the case company and with the aim of understanding the differences between the scoping processes of the phase-based process and the new agile development process. Our previous research into scoping (Wnuk et al, 2009) served as the basis for identifying research questions aimed at seeking a deeper understanding of overscoping as a phenomenon. In order to obtain detailed insight, an explanatory approach (Robson, 2002) was taken and the study design was based on the specific company context and the authors' pre-understanding. These investigations can then be broadened in future studies. Existing knowledge from literature was taken into account in interpretation and validation of the results.

A single-company explanatory case study (Robson, 2002) was performed using mainly a qualitative research approach complemented by a quantitative method for some of the data gathering. Qualitative research is suitable for investigating a complex phenomenon (such as overscoping) in a real-life context where it exists (Myers and Avison, 2002) (such as our case company). In this study, practitioners' perceptions of overscoping were studied through interviews where the verbalized thoughts of individuals with a range of different roles at the case company were captured(Myers and Avison, 2002; Robson, 2002). An overview of the research method is shown in Figure 4.1.

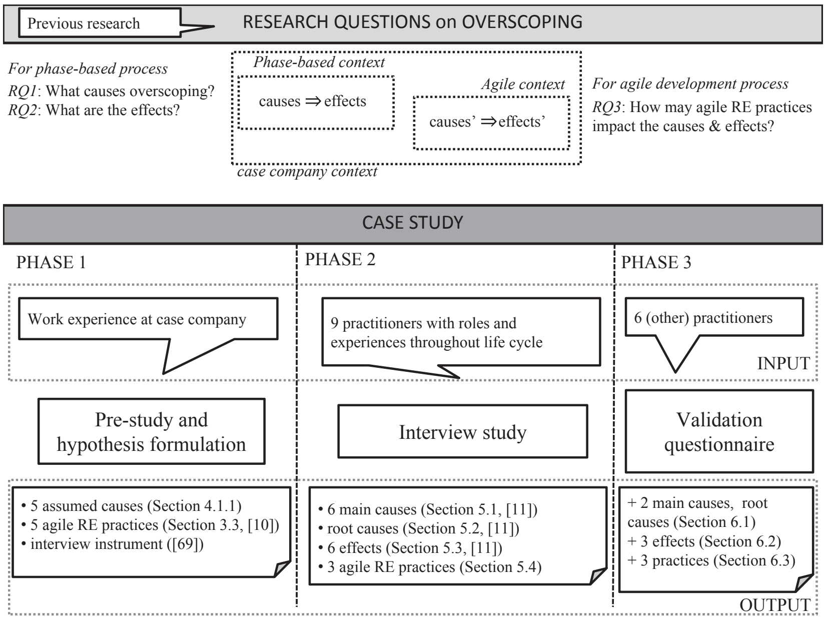

Previous research
RESEARCH QUESTIONS on OVERSCOPING
For phase-based process
RQ1: What causes overscoping?
RQ2: What are the effects?
Phase-based context
causes ⇒ effects
Agile context
causes' ⇒ effects'
case company context
For agile development process
RQ3: How may agile RE practices impact the causes & effects?
CASE STUDY
PHASE 1
PHASE 2
PHASE 3
Work experience at case company
9 practitioners with roles and experiences throughout life cycle
6 (other) practitioners
INPUT
Pre-study and hypothesis formulation
Interview study
Validation questionnaire
• 5 assumed causes (Section 4.1.1)
• 5 agile RE practices (Section 3.3, [10])
• interview instrument ([69])
• 6 main causes (Section 5.1, [11])
• root causes (Section 5.2, [11])
• 6 effects (Section 5.3, [11])
• 3 agile RE practices (Section 5.4)
+ 2 main causes, root causes (Section 6.1)
+ 3 effects (Section 6.2)
+ 3 practices (Section 6.3)
OUTPUT

Figure 4.1: Overview of research method for case study.

The case study was performed in three phases, see Figure 4.1. In the first phase, the industrial experience of one of the authors was used to formulate a hypothesis concerning possible (assumed) causes of overscoping and (assumed) effects which may result from overscoping. This hypothesis was used as a starting point in creating the interview instrument (Bjarnason, 2012) for the interviews, which took place in the second phase of the study. In the third phase, the interview results were presented to (another) six practitioners from the same company and validated by using a questionnaire (see Section 6 for more details and (Bjarnason, 2012) for the validation questionnaire). This was done to reduce the risk of inappropriate (false) certainty of the correctness of the results (Robson, 2002).

4.1 Phase one: pre-study and hypothesis generation

The purpose of the first phase of the study was to formulate a hypothesis on overscoping and prepare for the interviews. The experience of one of the authors (who has worked at the case company, with experience in several areas including coding, design, requirements engineering and process development) was used to identify possible (assumed) causes and effect of overscoping. In addition to these assumptions for the phase-based way of working, this author also identified the agile RE practices being introduced at the case company. These practices were assumed to impact one or more of the issues believed to cause overscoping in the phase-based process. If these assumptions were correct, applying the new practices should then result in reducing (or eliminating) the effects connected to those causes, and thus reduce (or eliminate) overscoping. In order to avoid selecting a set of assumptions biased by only one person, a series of brainstorming sessions around the hypothesis were conducted within the group of researchers involved in this study (i.e. the authors). The resulting (updated) hypothesis was then used as the main input in creating an interview study instrument (accessible online (Bjarnason, 2012)).

4.1.1 Formulated hypothesis

The hypothesis formulated for this study is that overscoping is caused by a number of factors, and that by addressing one or more of these factors, e.g. through agile RE practices, the phenomenon of overscoping may be alleviated, or even eliminated. The following five factors were identified as assumed causes for overscoping in phase one:

- *continuous requirements inflow via multiple channels (C1)* was assumed to cause overscoping by the many inflows increasing the difficulty of defining a realistic scope for multiple parallel projects. Requirements continuously arrive from the market, as well as, from internal stakeholders. This inflow was managed by batching those requests into one or two projects per year. It was a challenge to manage the execution of multiple parallel projects, while handling requests for new features and requirements, as well as, requests for changes to the agreed project scope.

- *no overview of software resource availability (C2)* was assumed to cause overscoping due to the challenge of balancing the size of the total scope for several (parallel) development projects against the (same) set of development resources. The resource allocation for the software development unit was handled at the DT level, i.e. there was no total overview of the load and available capacity of all the development resources of the software unit.

- *low DT involvement in early phases (C3)* was assumed to contribute to defining unrealistic and unclear requirements in the early phases, that are later deemed too costly or even impossible to implement, thus causing overscoping. The development teams were not very involved in the early project phases (MS1-MS4) with providing cost estimates and feedback during requirements writing.

- *requirements not agreed with DT (C4)* was assumed to cause overscoping due to not ensuring that the suggested scope was feasible and understandable. The requirements specification was not always agreed with the development teams at the handover point (MS2). Even if there was a formal review by DTs, we assumed that there was a low level of commitment from DTs. Furthermore, this low level of agreement was assumed to lead to low DT motivation to fulfil the requirements defined by the RTs.

- *detailed requirements specification is produced upfront (C5)* by the requirements teams by MS2 before the design starts was assumed to cause overscoping by limiting the room for negotiating requirements that could enable a more efficient design and realistic development plan. Furthermore, time and cost overhead for managing such changes was also assumed to contribute to overscoping.

4.2 Phase two: an interview study at the case company

In phase two, semi-structured interviews with a high degree of open discussion between the interviewer and the interviewee were held. The hypothesis provided a framework that helped to discuss, explore and enrich the understanding of this complex phenomenon. To avoid imposing this hypothesis on the interviewees, the discussion both on overscoping in general and on the detailed causes and effect was always started with an open ended question. In addition, the interviewees were asked to talk freely about the roles and phases she had experience from at the beginning of the interviews. In order to separate between the situation with the phase-based process and with the new agile RE practices, the impact of the new practices was discussed specifically in a (separate) section at the end of the interviews.

Our aim was to cover the whole process from requirements definition through development (design, implementation and testing) to the resulting end product (quality assurance, product projects), mainly for the phase-based process. This was achieved by selecting people with experience from all the relevant organizational units (see Section 3) to be interviewed and thereby catch a range of different perspectives on the phenomenon of overscoping. Nine people in total were selected to be interviewed. Two of the interviewees with identical roles requested to have their interviews together. The roles, organizational belongings, and relevant experience of the interviewed persons within the case company for the phase-based process can be found in Table 4.1. We have used a coding for the interviewees that also includes their organizational belonging. For example, interviewees belonging to the requirements unit are tagged with a letter *R*, belonging to product unit with a letter *P* and belonging to software unit with a letter *S*.

The interviews were scheduled for 90 min each with the possibility to reduce time or prolong it. All interviews were recorded and transcribed, and the transcripts sent back to the interviewees for validation. For each interview, the transcript was 7-10 pages long and contained in average 3900

Table 4.1: Interviewees roles (for phase-based process), organizational belonging and length of experience for each role within the company, see Section 3.1.

Code	Organizational unit	Role (s) within company	Years within role
Ra	Requirements	RT leader	5
Rb	Requirements	RT leader	2
Rc	Requirements	Requirements architect	3
Pd	Product	System test manager	7
Se	Software	Tester	3
Sf	Software	Software project manager	2
		DT leader	2
		Developer	2
Sg	Software	Quality manager	3
Sh	Software	DT requirements coordinator	1
		Developer	2
		DT leader	1
Si	Software	DT requirements coordinator	7

words. Transcription speed varied from 3 to 7 times of recorded interview time. The coding and analysis was done in MS Excel. The underlying section structure of interview instrument, i.e. causes, effects and agile RE practices, were numbered and used to categorize the views of the interviewees. For each interview, the transcribed chunks of text were placed within the relevant sections and, if so needed, copied to multiple sections. Relationships between different categories, as well as, the level of agreement on causes, effects and agile RE practices were noted in specific columns. The viewpoints of the two practitioners interviewed together (interviewees Ra and Rb) were separated into different columns in order to allow reporting their individual responses.

4.3 Phase three: validation of results via questionnaire

To further strengthen the validity of the results from the interviews a set of six (additional) practitioners at the case company was selected in phase three, see Table 4.2. To ensure that these six practitioners understood the results correctly and in a uniform way, the interview results were presented to them. During the meeting the participants could ask for clarifications and comment on the results, especially when they disagreed or had other different or additional viewpoints. In order to gather their views on the results in a uniform and non-public way, the participants were asked to fill out a questionnaire (available online at (Bjarnason, 2012)) stating to which

Table 4.2: Questionnaire respondents: roles and organizational belonging (for phase-based process), and length of experience within company (see Section 3.1) for descriptions of organizational units and roles).

Organizational unit	Role(s)	Years within company
Software	Software project manager, DT leader	4
Software	Tester	7
Software	DT requirements coordinator, DT leader	5
Requirements	Requirements architect	5
Requirements	RT leader	13
Product	System test manager	15

degree they agreed to the results and if additional relevant items had been missed. Due to limited availability of the participants a total of three such sessions were held. Each session was scheduled for 90 min with the possibility to extend or decrease the time as needed. The results from the questionnaire can be found in Section 6.

5 Interview results

The causes and effects of overscoping derived from the interviews performed in phase two of the study, see Figure 4.1, are outlined in Figure 4.2 and described in the following sections. Section 5.1 covers the main causes for overscoping, while the root causes are reported in Section 5.2 and the effects in Section 5.3. The findings from the interviews concerning how the agile RE practices may address overscoping are described in Section 5.4. The outcome of the validation questionnaire (phase 3) on these results is reported in Section 6.

5.1 Causes of overscoping (RQ1)

The viewpoint of each interviewee concerning the causes of overscoping was categorized and matched against the hypothesis regarding the assumed causes of overscoping (*C1-C5*, see Section 4.1.1). In addition, five of the eight interviewees were found to describe a sixth main cause for overscoping, namely *C6 unclear vision of overall goal*. A lack of (clearly communicated) strategy and overall goals and business directions for software development led to unclarities concerning the intended direction of both software roadmaps and product strategies, as well as, unclear business priority of project scope. The interviewees described how this cause (*C6*) led to scope being proposed primarily from a technology aspect, rather than from a business perspective, and without an (agreed) unified priority. Instead, most of the scope of a project was claimed to be critical and non-negotiable.

The interviewee results around the main causes of overscoping are shown in Table 4.3. The opinions of the interviewees have been classified in the following way:

- *Experienced:* the cause (both its occurrence and its impact on overscoping) is experienced
- *Agreed:* either the cause (occurrence and impact) was not directly mentioned, but derived or agreed after direct question, or when interviewee has no direct experience, but had observed it or heard about it from others.
- *Partly agreed:* Experienced or partly Agreed.
- *Disagreed:* does not agree to the cause, either its occurrence, or that it caused overscoping.
- *Not mentioned:* even though within expected experience for role.
- *NA:* not within the expected experience for the role (according to the process).

All interviewees had *Experienced* or *Agreed* to overscoping as a challenge, and a majority had *Experienced* or *Agreed* to causes 1-3. No interviewees *Disagreed* to any of the causes, though causes 4 and 5 both had less than a majority of *Experienced* or *Agreed* interviewees. Causes 4-5 were *Not mentioned* by all, while cause 6, which was derived from 5 of the interviewees, was *Not mentioned* by the others.

The entries marked *NA* (Not Applicable) indicate that the interviewee in her role was not expected to have experience of that cause. The system test manager (Pd) and the quality assurance manager (Sg) were classified as *NA* for C2, C3 and C4 since they merely observed the requirements flow from their management-level positions and were not directly involved in the early phases of the projects. In addition, Sg was also classified as *NA* for C5 due to lack of contact with the SRS. Furthermore, the software tester (Se), who had no insight into project planning, was categorized as *NA* for the causes C1 and C2. For all assumed causes there were some counts of *Partly agreed*, namely:

- *continuous requirements inflow via multiple channels* (C1). The quality manager (Sg) mentioned the continuous inflow of requirement changes after setting the scope as causing overscoping, but no root causes prior to this milestone, and is therefore classified as *Partly agreed*.

	Overscoping as a challenge			C1 Continuous req. inflow			C2 No overview of softw. resources			C3 Low DT Involvm. in early phases			C4 Reqs. not agreed with DTs			C5 Detailed reqs. specification produced upfront			C6 Unclear vision of overall goal		
	Requirements	Software	Product	Requirements	Software	Product	Requirements	Software	Product	Requirements	Software	Product	Requirements	Software	Product	Requirements	Software	Product	Requirements	Software	Product
Experienced	2	5	1	1	3	1	1	2		3	1		1	1			2		3	2	
Agreed	1			2			2				1			1		1	1				
Partly agreed					1			1			2			2		1					
Disagreed																					
Not mentioned													2			1	2			3	1
NA					1			2	1		1	1		1	1			1			

Table 4.3: For each identified main causes of overscoping, the number of interviewees per response category (see Section 5.1) and organizational unit (Reqs, etc, see Section 3.1).

- *no overview of software resource availability* (C2). One of the DT requirements coordinators (Si) is noted as *Partly agreed* to this cause, due to believing that a better overview of available resources would not alleviate the overscoping to any greater extent. In contrast, another interviewee (Sf) saw this as a strong cause for overscoping; 'There was no control of what people were working with. There were different DT leaders who just sent out [tasks]'.

- *low DT involvement in early phases* (C3). Both DT requirements coordinators (Sh, Si) were categorized as *Partly agree* since the involvement from the rest of the DT including producing cost estimates was low, even though they personally had experienced good cooperation with the RT leaders during MS1-MS2. This lack of involvement was seen by the DT tester (Se) as leading to an unrealistically large scope being specified, "'The full view of requirements would be improved by including input from more roles, and a more realistic scope could be identified earlier on."'

- *requirements not agreed with DT* (C4). The DT requirements coordinators (Sh, Si) believed that the requirements were understood and agreed with the DT at MS2, though the DT did not commit to implementing them at that point. One of them (Sh) mentioned that the system requirements specification was primarily agreed with the DT requirements coordinators and not with developers and testers in the DT.

- *detailed requirements specification produced upfront* (C5). One of the RT leaders (Rb) had an agile way of working and did not produce a detailed requirements specification upfront, but instead regularly and directly interacted with the DT. This increased insight into the DT enabled a more flexible discussion around scope and detailed requirements, led to overscoping being experienced as a more manageable challenge by Rb. The other RT leader (Ra, interviewed together with Rb) did not mention C5 as causing overscoping, but agreed to Rb's conclusions and was noted as *Partly agreed*. Ra had the opposite experience, i.e. of producing and relying on a requirements specification, and then not staying in touch with the DT during the later phases of development (after MS2) who then developed software that was usually different from what was specified in the SRS. One of the DT interviewees (Sf) believed that the (wasted) effort of producing and agreeing to detailed requirements upfront (for features that were later descoped) increased the overscoping since it hindered those resources from working on viable features. Another interviewee (Sh) said: 'At this point [MS2] we [DT] approved a lot of things, because we liked what they [RT] wrote here and we really wanted that functionality then we [DT] started to over commit.'

5.2 Root cause analysis (RQ1)

To provide a deeper understanding the interviewees were asked to describe what may be triggering overscoping, i.e. the root causes of overscoping. These root causes have been grouped according to the main cause (C1-C6, outlined in Sections 4.1 and 5.1) that they affect. A full picture of the cause-effects relationships for overscoping identified through this study is depicted in Figure 4.2. The results around root causes from both the interviews and from the questionnaire are also summarized in Table 4.4.

- *root causes of C1 (continuous requirements inflow via multiple channels)*. A number of requirement sources besides the regular requirement flow (defined by the RTs) were mentioned as contributing to the continuous inflow. These include: requirements produced by defining many different product variants in the product line (RC1a); and, many late new market requirements and changes incurred by the long project lead times (RC1b) compared to rate of requirements change. Furthermore, communication gaps (RC1c) were mentioned as causing additional requirements inflow throughout the project life cycle. These consist of communication gaps between the requirements unit and the software unit (RC1ci) which resulted in the software unit preferring to work according to their own software-internal roadmap containing a large amount of requirements not agreed with the requirements unit. Communication gaps between technical areas, both for RTs and for DTs, (RC1cii) led to indirect requirements between DTs being discovered after the initial scope selection at MS2, which greatly increased the amount of implementation required. The impact of these indirect requirements was especially large for DTs responsible for service-layer functionality like display frameworks and communication protocols. Furthermore, communication gaps between usability design and the RTs (RC1ciii) resulted in additional functional requirements appearing in usability design specification, sometimes in conflict with RT requirements. And, finally, due to lack of communication between the software quality managers and the requirements unit (RC1civ), requirements on quality aspects were not defined and prioritized together with the RT requirements, but managed separately in a later phase.

- root causes of C2 (no overview of software resource availability). The lack of overview of available software development resources was believed to be a consequence of communication gaps within the software unit and between the DTs (RC2a). The organizational structures and the high scope pressure were seen to result in each DT focusing on their own areas rather than striving for cooperation and good communication with other DTs. One interviewee described that en-

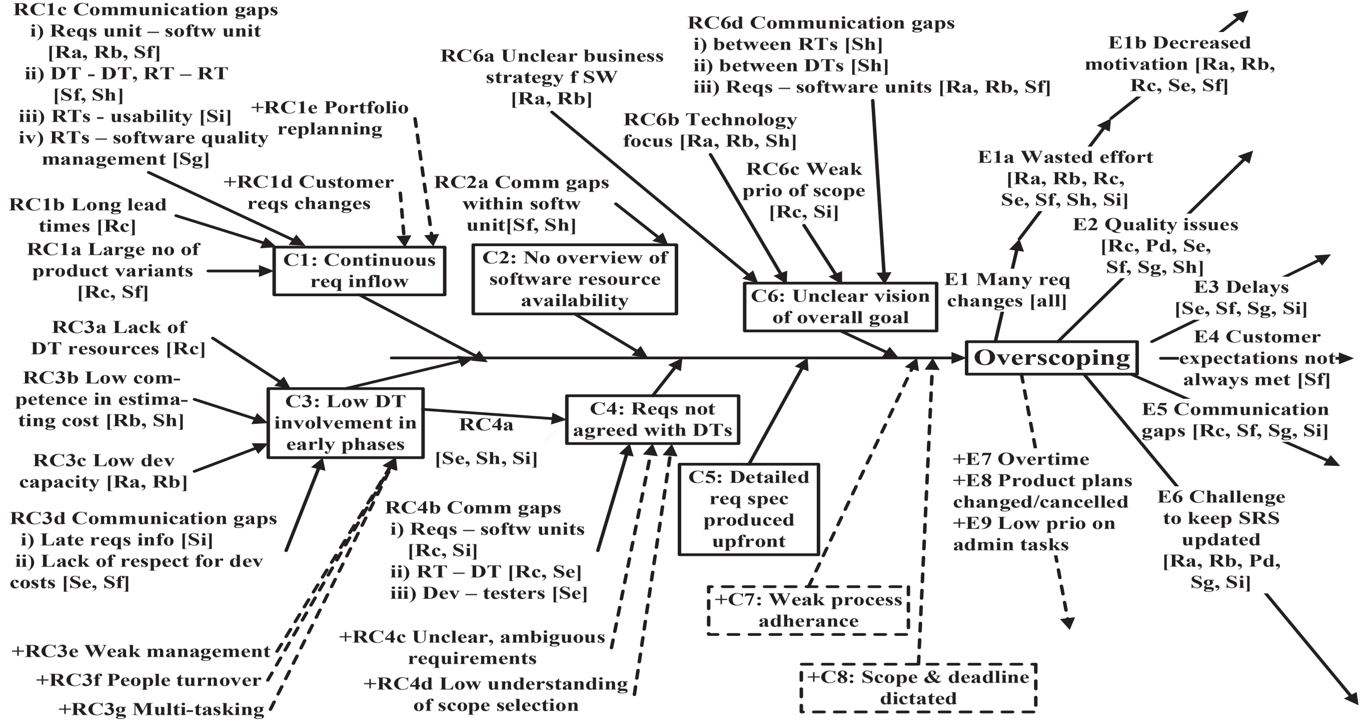

RC1c Communication gaps
i) Reqs unit – softw unit [Ra, Rb, Sf]
ii) DT - DT, RT – RT [Sf, Sh]
iii) RTs - usability [Si]
iv) RTs – software quality management [Sg]
+RC1e Portfolio replanning
+RC1d Customer reqs changes
RC1b Long lead times [Rc]
RC1a Large no of product variants [Rc, Sf]
C1: Continuous req inflow
RC2a Comm gaps within softw unit[Sf, Sh]
C2: No overview of software resource availability
RC6a Unclear business strategy f SW [Ra, Rb]
RC6d Communication gaps
i) between RTs [Sh]
ii) between DTs [Sh]
iii) Reqs – software units [Ra, Rb, Sf]
RC6b Technology focus [Ra, Rb, Sh]
RC6c Weak prio of scope [Rc, Si]
C6: Unclear vision of overall goal
E1b Decreased motivation [Ra, Rb, Rc, Se, Sf]
E1a Wasted effort [Ra, Rb, Rc, Se, Sf, Sh, Si]
E1 Many req changes [all]
E2 Quality issues [Rc, Pd, Se, Sf, Sg, Sh]
E3 Delays [Se, Sf, Sg, Si]
E4 Customer expectations not always met [Sf]
E5 Communication gaps [Rc, Sf, Sg, Si]
E6 Challenge to keep SRS updated [Ra, Rb, Pd, Sg, Si]
Overscoping
RC3a Lack of DT resources [Rc]
RC3b Low competence in estimating cost [Rb, Sh]
RC3c Low dev capacity [Ra, Rb]
RC3d Communication gaps
i) Late reqs info [Si]
ii) Lack of respect for dev costs [Se, Sf]
C3: Low DT involvement in early phases
RC4a
[Se, Sh, Si]
C4: Reqs not agreed with DTs
RC4b Comm gaps
i) Reqs – softw units [Rc, Si]
ii) RT – DT [Rc, Se]
iii) Dev – testers [Se]
C5: Detailed req spec produced upfront
+E7 Overtime
+E8 Product plans changed/cancelled
+E9 Low prio on admin tasks
+RC3e Weak management
+RC3f People turnover
+RC3g Multi-tasking
+RC4c Unclear, ambiguous requirements
+RC4d Low understanding of scope selection
+C7: Weak process adherance
+C8: Scope & deadline dictated

Figure 4.2: Overview of all found causes (C), root causes (RC) and effects (E) of overscoping. Items derived from questionnaire noted with + and dashed lines. Interviewee code (see Section 4.2) noted within brackets.

abling DTs to coordinate their plans had the effect of improving the scoping situation by increasing the delivery rate and efficiency, 'We tried to solve the overscoping by enabling the DTs to co-plan and deliver incrementally. This resulted in more deliveries and increased efficiency.' (Sf)

- *root causes of C3 (Low DT involvement in early phases).* Several interviewees described that software resources were rarely available in early project phases (RC3a) due to development and maintenance work for previous projects. Rc said: 'by MS2, but it was hard to get [DT] resources. That probably was the problem.' In addition, weak and incorrect cost estimations (RC3b) were mentioned as leading to including too much into the project scope. In contrast, low development capacity of the software unit (RC3c) caused by bad architecture was believed by the two RT leaders to be the main reason for over-scoping. Furthermore, gaps in the communication (RC3d) between the requirements unit and the software unit were mentioned as causing low DT involvement. For example, interviewees mentioned that early DT involvement was often postponed due to a lack of understanding within the software unit for the importance of this work. However, the opposite was also mentioned, namely that the DTs received requirements information too late (RC3di) which then resulted in extending the project scope without realistic plans. Similarly, the cost estimates for both development and testing were not always respected (RC3dii). In contrast, close cooperation between the RTs and the DTs were experienced (by Rc) to lead to an early uncovering of problems, thereby enabling definition of more stable requirements that were then successfully implemented.
- *root causes of C4 (requirements not agreed with DTs).* Low DT involvement in the early phases (C3, RC4a) was seen as leading to weak agreement and commitment to the requirements, by all three interviewees with experience from planning DT work (Se, Sh, Si). The interviewees connected the level of requirements agreement with the level of communication around requirements (RC4b), i.e. RTs and DTs that communicated well also tended to have a mutual understanding and agreement of the requirements. Due to variations in communication between teams, the view on C4 varied between interviewees (see Section 5.1). Even so, one interviewee (Sh) who had experienced good cooperation with the RT mentioned that the different organizational belongings (RC4bi) caused timing issues due to different priorities for different units. In addition, communication gaps between RTs and DTs (RC4bii) including no contact between testers and RT leaders were caused by physical and organizational distances and resulted in weak DT agreement on the requirements. Weak communication on requirements and design between develop-

ers and testers (RC4biii) was also mentioned (by Se) as causing weak requirements agreement.

- *root causes of C5 (detailed requirements specification produced upfront).* The phase-based process defined that a requirements specification should be produced by MS2, therefore no further root causes have been identified for this cause.

- *root causes of C6 (unclear vision of overall goal).* The RT leaders (Ra and Rb) described that the lack of clear business strategy (RC6a) and vision that could guide them in defining a roadmap resulted in proposing a project scope from a pure technology standpoint (RC6b). A weak and unified business priority (RC6c) of the proposed scope (almost everything was 'critical') was described (by Si) as pushing the DTs to commit to unrealistic project plans. In addition, Rc mentioned that the lack of unified priority hindered the project management from effectively addressing the overscoping. Furthermore, several communication gaps (RC6d) were seen to contribute to this cause. Weak communication both between RTs (RC6di) and between DTs (RC6dii) were described by Rc as resulting in weak scope coordination between functional areas, as well as, conflicts and lack of clarity concerning the overall goal. Finally, both RT leaders described that communication gaps and low common understanding between the requirements unit and the software unit (RC6diii) of the overall goal resulted in the project scope being decided to a large extent by the DTs, and not (as the process stated) by the RTs.

5.3 Effects of overscoping (RQ2)

The interviews uncovered the following six main effects of over-scoping (marked as E1 to E6, see Figure 4.2):

- *many requirement changes after the project scope is set (E1).* All interviewees had experienced that overscoping caused requirement changes to take place after the project scope was set (at MS2). As the projects proceeded and the overload was uncovered large amounts of features were removed from scope (descoped). The phenomena was so common that the phrases 'overscoping' and 'descoping' have become part of company vocabulary. This descoping of already started features was a waste (E1a) of both RT and DT effort and led to frustration and decreased motivation (E1b) to work with new requirements. As interviewee Sh said: 'There are many things that you as a tester or developer have spent time on that never resulted in anything. And that isn't very fun. There is a lot of overtime that has been wasted.' However, the many requirement changes were experienced by Pd as

	Mentioned as causes or root causes by nbr. of interviewees (9 in total)	Number of questionnaire responses (6 in total)				
		Experienced	Agree	Party agree	Disagree	Do not know
Overscoping (as a challenge)	9	6				
C1: Continuous reqs. inflow via multiple channels	8	4	2			
(a) Large number of product variants	2	3	1	2		
(b) Long lead times	1	4	2			
(c) Communication gaps	6	3	1	1		1
(i) Between reqs and software unit	3	3	2		1	
(ii) Between RT-RT and DT-DT	2	3	2			1
(iii) Between RT and usability design	1	2	1	1		2
(iv) Between RT and software quality managers	1		2			4
(+d) Customer requirements changes (many and late)		3				
(+e) Product portfolio re-planning		1				
C2: No overview of software resource availability	6	2	3		1	
(a) Communication gaps within software unit	2	1	2	1	1	1
C3: Low development team involvement in early phases	7	1	2	2	1	
(a) Lack of DT resources for pre-development work	1	2	2	2		
(b) Low competence in estimating cost	2	2	1	3		
(c) Low development capacity	2	1	1	2	1	1
(d) Communication gaps	3					
(i) Late requirements information to DT	1	2		2	1	1
(ii) Lack of respect or understanding of development cost	2	2	3		1	
(+e) Weak leadership including ineffective communication		1				
(+f) Change of people during the project		1				
(+g) Multi-tasking		1				
C4: Requirements not agreed with development teams	5	2	2	2		
(a) Low DT involvement in early phases (C3)	3	2	2	1	1	
(b) Communication gaps	3	1	2	2	1	
(i) Between requirements and software units	2	1	2		1	2
(ii) Between RT and DT	2	1	1	2	1	1
(iii) Between developers and testers	1	1	1	1	2	1
(+c) Unclear and ambiguous requirements		3				
(+d) Low understanding of why particular scope is selected		1				
C5: Detailed reqs specification produced upfront	5	1	3	1	1	
C6: Unclear vision of overall goal	5	4	1		1	
(a) Unclear business strategy for software development	2	3	2	1		
(b) Technology focus when scope set	3	3	2		1	
(c) Weak priority of scope	2	3	2	1		
(d) Communication gaps		2	3			1
(i) Between RTs	1	1	3			2
(ii) Between DTs	1	2	2			2
(iii) Between requirements and software units	3	1	3		1	1
+C7 Weak process adherence		1				
+C8 Overall scope and deadline dictated from management		1				

Table 4.4: Summary of all identified causes and root causes of overscoping: Number of responses for interviewees (see Section 5 for details) and for Questionnaire responses per level of agreement (see Section 6 for details). Additional items from questionnaire responses are marked with +.

having only minor impact on the system testing. They merely adjusted the test plans, and rarely wasted any effort due to this effect.

- *quality issues (E2).* All interviewed practitioners involved after MS4 (when development started, Rc, Pd, Se, Sf, Sg, Sh) mentioned that software quality was negatively affected by over-scoping both due to the high workload and due to the many requirement changes. The software quality manager Sg expressed, 'If you get too much scope, you get quality problems later on and you haven't got the energy to deal with them.' Similarly, interviewee Pd said: 'When you have a lot going on at the same time, everything isn't finished at the same time and you get a product with lower quality.' Furthermore, the lack of respect for development costs (C3dii) in the earlier phases was mentioned by the software tester (Se) to contribute to insufficient testing and subsequent quality issues.

- *delays (E3).* The overscoping and subsequent overloading of the DTs was described by several practitioners as resulting in delayed deliveries being the norm rather than the exception. In addition, overscoped DTs were often forced to commit to customer-critical requests and changes which in turn resulted in even more delays and quality issues (E2). One DT interviewee (Sf) stated that 'our team was always loaded to 100% at MS4, which was too much since there were always customer requests later on that we had to handle. That meant that we were forced to deliver functionality with lower quality or late.' The same situation was described by the quality manager (Sg) who said: 'Even if we decided on a scope for MS4, there were extremely many changes underway, so we were never ready by MS5, as we had said, but were delayed.'

- *customer expectations are not always met (E4).* Overscoping was mentioned by a few interviewees as resulting in sometimes failing to meet customer expectations. For example, customers sometimes file change requests for features that had previously been removed due to overscoping. In addition, overscoping caused by requiring a large number of products (RC1a) with different display sizes and formats was experienced by interviewee Sf as resulting in releasing products with faulty software, e.g. misplaced icons.

- *communication gaps (E5).* Overscoping and overloading an organization was described as leading to several communication gaps; between the requirements and software units; within the software unit itself, between DTs (Sg, Si) and between DTs and software project managers (Sf); and between the software and the product unit. For example, the many descoped features (E1) and wasted effort (E1a) resulted in distrust between the requirements unit and the software unit, so much so that the software unit defined their own internal roadmap without coordinating this with the requirements unit. Furthermore, invalid error reports filed by the system testers based on an unreliable SRS (caused by E1-E6) caused an increase both in work load and in frustration at the software unit and, consequently friction and widened communication gaps between these units.

- *challenge to keep the SRS updated (E6):* The situation caused by overscoping, with a high workload and many late requirement changes (E1), increased the challenge of keeping the SRS updated. The practitioners mentioned that in an overscoping situation the task of updating the SRS was given low priority (partly caused by E1b). Furthermore, the amount of required updates both for changed and descoped requirements was increased (Ra, Rb, Pd, Sg, Si) by producing the requirements upfront (C5) with a low level of DT agreement (C4). The RT leaders (Ra, Rb) had also experienced that many requirement-related changes were made during development without informing the RTs (or the system testers), many of which might have been a result of insufficient DT involvement in the early phases (C3).

5.4 Impact of agile RE practices (RQ3)

The general opinion of the interviewees on the situation after introducing the agile RE practices (see Section 3.3) is that even though some overscoping is still experienced, it is a more manageable challenge than with the previous phase-based process. For example, there is less descoping and most of the features worked on by the software unit now continue until completion (Si). Interviewee Sg said: "We still have overscoping in all projects. But, it is more controlled now and easier to remove things without having done too much work." Many of the interviewees stated that in theory the agile RE practices address overscoping, but that these practices also incur a number of new challenges. The following practices were mentioned by the interviewees as impacting some of the causes and/or root causes of overscoping:

- *one continuous scope and release-planning flow (P1).* This practice (which was implemented at the time of the interviews) was seen to address the root cause weak prioritization of scope (RC6c, mentioned by Rc, Pd, Sg, Sh) and the causes continuous requirements inflow via multiple channels (C1, mentioned by Se, Sf) and no overview of software resource availability (C2, mentioned by Sf, Sg), by enforcing that all scope and development resources are managed through a uniformly prioritised list.

- *cross-functional development teams (P2).* This practice (which was implemented at the time of the interviews) was seen to address several communication gaps, and, thus, impact causes C1-C4 by closing the gaps (identified as root causes) between RTs and DTs and between different functional areas. This practice was also believed to impact C5 (detailed requirements specification produced upfront) since detailing of requirements is now handled within the development teams together with the customer representative. Interviewee Sf said: 'It is an advantage that they [the team] sit together and can work undisturbed, and there is no us-and-them, but it is us. And when they work with requirements the whole group is involved and handshakes them.'

- *gradual and iterative detailing of requirements (P3).* This practice (which was partly implemented at the time of the interviews) was mentioned as directly impacting the cause C5 (detailed SRS produced upfront). Furthermore, this practice was also seen by Sf and Sg to reduce both the lead time for each high-level requirement (RC1b) and the amount of changes after project scope is set (E1) and, thus also reduce the amount of wasted effort (E1a, also mentioned by Ra, Rb).

6 Validation questionnaire on interview results

Overscoping was further investigated through the validation questionnaires (Bjarnason, 2012), see Table 4.4. Each of the six respondents noted her level of agreement by using the following notation:

- *Experienced:* I have experienced this (item and connection) to be valid.
- *Agree:* I agree to this, but have not experienced it personally.
- *Partly agree:* I agree to part, but not all, of this.
- *Disagree:* I do not agree.
- *Do not know:* I have no knowledge of this item or its impact.

6.1 Causes and root causes (RQ1)

A majority of the questionnaire respondents confirmed (i.e. Experienced or Agreed to) all main causes as contributing to overscoping, except C3 (low DT involvement) for which there was also one Disagree response. Causes C2, C3, C5 and C6 each had one count of Disagree from respondents with experience from the requirements unit. Two additional main causes were given by two respondents, namely weak processes adherence (+C7) and dictation of scope and deadlines from management (+C9). Furthermore, some additional root causes were given for C1, C3 and C4. For C3, two alternative root causes were given, namely turn-over of DT members as the project progressed (RC3f) and assigning the same resources to

	Total impact	Softw	Softw	Softw	Reqs	Reqs	Product
C1: Continuous reqs inflow via multiple channels	275	20	20	15	50	100	70
C2: No overview of software resource availability	60	10	20		20		10
C3: Low DT involvement in early phases	80		10	50	20		
C4: Requirements not agreed with DTs	10	5	5				
C5: Detailed reqs specification produced upfront	15	5	5	5			
C6: Unclear vision of overall goal	140	60	40	30	10		
+C7: Weak process adherence							
+C8: Overall scope and deadline dictated from top	20						20

Table 4.5: The total number of points reflecting the impact of each cause on overscoping. Each questionnaire respondent distributed 100 points. Effects of overscoping (RQ2). The columns show the number of points per responder (organisational belonging given in header, see Section 3.1.

multiple parallel projects (RC3g). For C4 (requirements not agreed with DT) three respondents stated that this was caused by unclear and ambiguous requirements (RC4c), while one respondents suggested that DTs often lacked insight into why certain features and requirements were important, which is related to C6 (unclear vision of overall goal). All responses from the validation questionnaire on causes and root causes can be found in Table 4.4.

The impact of each main cause on overscoping was gauged by asking the questionnaire respondents to distribute 100 points over all causes according to the extent of their impact (see Table 4.5) C1 got the highest score in total and all six respondents, thereby indicating that the continuous requirements inflow was a main cause of overscoping. The second highest total score was given to C6 (unclear vision of overall goal), which all the participants from the software unit graded with 30-60, while the other participants graded this with 0 or 30. Causes C4, C5, +C6 and +C7 were seen as having a minor or no impact on the overscoping situation.

6.2 Effect of overscoping (RQ2)

In large, the questionnaire respondents had experienced or agreed to all the effects of overscoping identified from the interviews. The respondent from the product unit had no view on E5 or E6, while the requirements architect partly agreed E5. In addition, the respondents mentioned the following effects of overscoping: overtime (+E7); changed and sometimes canceled product plans (+E8); low prioritization of administrative tasks (+E9). The full questionnaire response on effects is shown in Table 4.6.

In addition to stating the level of agreement to the identified effects of overscoping, the respondents were asked to grade their impact. The following notation was used:

- *Critical:* Company or customer level.
- *Major:* Project or unit level.
- *Medium:* Team level.
- *Minor:* Individual level.
- *None:* No impact.

All the effects identified from the interviews were seen as having an impact. All effects except E5 (communication gaps) were seen as having major or critical impact by a majority of the participants. There were two counts of minor impact: one for E6 (keeping SRS updated) and one for +E7 (overtime).

6.3 Impact of agile RE practices (RQ3)

The questionnaire respondents mostly agreed to the three identified agile RE practices as impacting the challenge of overscoping, either through their own experience or by believing the practice should work in theory. Furthermore, some additional practices were mentioned as impacting overscoping: (+P4) clearer company vision (i.e. directly addressing C6), (+P5) open source development (limiting C1 by restricting what the customer can reasonably expect when large parts of the software are outside of company control) and (+P6) incremental deliveries (shorter cycles facilitate scope size control for each cycle). Table 4.7 contains the questionnaire responses on the impact of the agile RE practices on overscoping.

Finally, the respondents had all experienced, agreed or partly agreed that overscoping was still a challenge for the case company. The new agile process and practices are seen to, at least partly, address the situation and provided ways to better manage and control the extent of overscoping and its effects. The practitioners' responses concerning the current situation are shown in Table 4.8.

	Mentioned by nbr. of interviewees (9 in total)	Questionnaire responses (6 in total)									
		Agreement					Impact				
		Experienced	Agree	Partly agree	Disagree	Don't know	Critical	Major	Medium	Minor	None
E1: Many req. changes after scope is set	9	5	1				4	2			
(a) Wasted effort	7	5	1				3	3			
(b) Decreased motivation	5	4	2				3	2	1		
E2: Quality issues	6	6					5	1			
E3: Delays	4	6					5	1			
E4: Customer expectations not always met	1	4	2				5	1			
E5: Communication gaps	4	2	1	1		1	2	1	3		
E6: Keep SRS updated	5	1	4			1		5		1	
+E7: Overtime		3					1		1	1	
+E8: Changed and canceled product plans		1					1				
+E9: Administrative tasks not always performed		1					1				

Table 4.6: Number of questionnaire responses on the effects of overscoping per level of agreement (notation described in Section 6) and per impact category (notation described i Section 6.2). Additional items derived from questionnaire marked with +.

	Experienced	Agree (in theory)	Partly agree	Disagree	Do not know
P1: One continuous scope and release-planning flow	2	4			
P2: Cross-functional development teams	3	2			1
P3: Gradual and iterative detailing of requirements	2	2	2		
+P4: Company vision	1				
+P5: Open source development	1				
+P6: Incremental deliveries	1				

Table 4.7: Number of questionnaire responses on the impact of agile RE practices on overscoping per level of agreement notation described in Section 6). Additional practices identified through questionnaire responses are marked with +.

	Experienced	Agree (in theory)	Partly agree	Disagree	Do not know
Overscoping is still a challenge	3	1	2		
There is less overscoping now	1	1	3	1	
Overscoping is more manageable now	1	3	1		1

Table 4.8: Number of questionnaire responses per agreement category (described in Section 6) on the current situation at the case company with agile RE practices, as compared to when using phase-based process.

7 Interpretation and discussion

The results of this study corroborate that overscoping is a complex and serious risk for software project management (Boehm, 1989; DeMarco and Lister, 2003; Legodi and Barry, 2010) both for phase-based and for agile development processes. In addition, the results show that communication issues have a major impact on overscoping. This complements the work by Sangwan et al and Konrad et al (2008) who mentioned that weak communication can cause project failures in large-scale development and global software engineering (Sangwan et al, 2006). Moreover, our results extend the lists of effects of weak coordination proposed by Sangwan et al (2006) (long delays, leave teams idle and cause quality issues) by adding overscoping. Further research is needed to fully identify and address the factors involved. The results are discussed and related to other research in further detail, per research question, in Sections 7.1 (RQ1), 7.2 (RQ2) and 7.3 (RQ3). Finally, the limitations of this study and threats to validity of the results are discussed in Section 7.4.

7.1 Causes of overscoping (RQ1)

Our results indicate that overscoping is caused by a number of causes and root causes. These causes mainly originate from the nature of the MDRE context in which the company operates, but are also due to issues concerning organizational culture and structures, and communication. This was further highlighted by interviewees describing the additional cause C6 (unclear vision of overall goal) and two questionnaire respondents mentioning additional causes connected to lack of respect for the decision- and development process, i.e. C7 and C8. In contrast, practitioners with experience of good cooperation and well-communicating teams described overscoping as a less serious and more manageable challenge. This may explain all the *Disagree* questionnaire responses but one (i.e. C5).

We interpret the results around the six causes of overscoping identified through the interviews (see Section 5.1 and Figure 4.2) as follows:

- *continuous requirements inflow from multiple channels (C1).* We interpret the homogeneity of the interview and questionnaire results (see Tables 4.3, 4.4 and 4.5) to mean that a large and uncontrollable inflow of requirements has the potential to cause over-scoping when not managed and balanced against the amount of available capacity. This cause was also been identified by Regnell and Brinkkemper (2005) and Karlsson et al (2007a) as one of the challenges of MDRE. In addition to corroborating this challenge, our work also identifies that this continuous inflow of requirements can cause overscoping. The importance and seriousness of this factor are indicated by this cause scoring the highest total impact factor in the questionnaire, see Ta-

ble 4.5. The extent to which this cause affects companies that operate in the bespoke requirements engineering context (Regnell and Brinkkemper, 2005) requires further research.
Our study also reveals that the inlow of requirements can be further increased by scope creep at the software management level through a software-internal roadmap (RC1ci, see Section 5.2). In effect, this hindered resources from being available for managing new customer requirements. Similar results have been reported by Konrad and Gall (2008) who found that scope creep can result in problems with meeting customer expectations, i.e. effect E4 (see Section 5.3). Konrad et al (2008) propose addressing scope creep by increased understanding and traceability of customer requirements, and by creating an effective hierarchical CCB structure. The impact of these methods on overscoping remains to be evaluated.

- *no overview of software resource availability (C2).* The majority of our responders (six of nine interviewees and five of six questionnaire respondents) had experienced or agreed to the lack of overview of available resources being a cause of overscoping. However, the questionnaire results suggest that the impact of this cause is not as critical as cause C1. This result is surprising, when considering the importance of management of the daily workload including coordination of tasks and activities reported by, e.g. Philips et al (2012). The contrasting opinions of low development capacity (RC3c, held by RT leaders) and low respect for development costs (RCdii, held by DT roles) is interesting. This difference can be interpreted as a low understanding of each other's viewpoint around cost and an indication that this view-point is dependent on role (related to Jorgensen and Shepperd (2007)). If the development capacity really is low is a different issue. Finally, this cause specifically includes the lack of overview, or awareness of the total load on the resources. To the best of our knowledge, this issue has not been empirically investigated. Rather software cost estimation research (Jørgensen and Shepperd, 2007) mainly focuses on effort estimation and on optimizing resource assignment (Lixin, 2008).

- *low development team involvement in early phases (C3).* The results indicate that low development involvement in the requirements phase can cause overscoping (mentioned by 6 out of 9 interviewees and 5 out of 6 questionnaire respondents did not disagree to this). This confirms previous work that points out the need of early development involvement in requirements engineering, e.g. required by interdependencies between product management and software development (Nuseibeh, 2001). Glinz et al also mentioned that lack of communication between project management and development at requirements hand-off may lead to unsatisfactory results (Glinz et al,

2002). Similarly, Karlsson et al (2007a) reported that communication gaps between marketing (requirements unit for our case company) and development, can result in insufficient effort estimates (i.e. RC3b) and in committing to unrealistically large features without considering the technical and scheduling implications (Karlsson et al, 2007a).
Our results corroborate these results in that low involvement and weak communication in early phases may lead to problems later on, including overscoping. These communication issues may also exacerbate the problem of getting accurate and reliable effort estimates (RC3b). Furthermore, the fact that one questionnaire respondent expressed experiencing good communication and cooperation between requirements and development teams may also explain the one Disagree response for this cause. On the other hand, a surprising result from the validation questionnaire is that this cause (C3) was seen to influence overscoping less than cause C6 (unclear vision of overall goal) both in total (among all respondents) and by 2 of the 3 software respondents. These results indicate that there may be additional (uncovered) factors that influence the impact this cause has on overscoping.
Finally, several methods have been proposed for addressing cause C3, e.g. negotiation of implementation proposals (Fricker et al, 2007), model connectors for transforming requirements to architecture (Medvidovic et al, 2003), cooperative requirements capturing (Macaulay, 1993) and involving customers in the requirements management process (Kabbedijk et al, 2009). Goal-oriented reasoning can also provide constructive guidelines for architects in their design tasks (van Lamsweerde, 2003). If and to which degree the mentioned methods can alleviate overscoping by impacting this cause remains a topic for further research.

- *requirements not agreed with development team (C4).* The results provide empirical evidence that weak agreement on requirements between requirements and software units can cause overscoping (all 6 questionnaire responders agreed to cause C4 and five interviewees mentioned C4 as a cause of overscoping). A significant root cause for this cause was found to be communication gaps, mainly between the requirements-related roles and the development and testing roles. This confirms the viewpoint of Hall et al (2002) that most requirement problems are actually organizational issues. In addition, this confirms the importance of seamless integration of different processes in collaborative work (Ebert and De Man, 2002). The impact of insufficient communication on software engineering has been reported as a general issue within requirements engineering and product management (Bjarnason et al, 2011b; Fricker et al, 2007; Hall et al, 2002;

Kabbedijk et al, 2009; Karlsson et al, 2007a). Surprisingly, C4 scored the lowest impact among all the causes and only two questionnaire responders (both from the software unit) rated this cause as having any (low) impact factor on overscoping. In contrast, cause C6 (weak vision of overall goal) was rated as having the largest impact on overscoping.

- *detailed requirements specification produced upfront (C5).* Our results indicate that too much detailed documentation produced upfront may cause overscoping (mentioned by five interviewees and experienced, agreed or partly agreed to by five questionnaire respondents, see section 5.1). This complements other studies into documentation in software engineering projects. For example, Emam and Madhavji (1995) mentioned that in organizations which require more control the pressure to produce much detail is also greater. Lethbridge reported that, for software engineers, there is often too much documentation for software systems, frequently poorly written and out of date (Lethbridge et al, 2003). Furthermore, Sawyer et al (1999) mention that premature freezing of requirements may cause scope creep and communication problems (both of which are identified as root causes of overscoping in our study) and suggest evolutionary prototyping as a remedy. Other remedies suggested for addressing excessive documentation include reuse of requirements specifications (Faulk, 2001), as well as, simply creating less documentation (Aurum and Martin, 1999). The effectiveness of these methods for the risk of overscoping remains to be investigated.
  The differing views on this cause between respondents may be explained by their roles and relationship to RE. All the disagreeing questionnaire respondents for this cause worked with requirements related roles. These roles are more likely to consider detailed requirements specifications as positive and good, rather than an issue. However, these roles have less insight into the later phases when development takes place and the effects of overscoping are experienced. Three of the respondents with experience from later development phases had experienced C5 as causing overscoping. Furthermore, Berry et al mentioned that when time for elicitation is short, i.e. there is a lack of upfront documentation (or lack of C5), the requirements usually end up as an enhancement or become descoped since all of the client's requests can not be delivered (Berry et al, 2010). Considering this, we conclude that both under specifying, as in Berry et al (2010), and over specifying, as in our study, can cause overscoping and later descoping, and that it remains to be investigated how to strike a good balance.

- *unclear vision of overall goal (C6).* Our study identifies that a lack of clearly communicated goals and strategy for software development

may cause defining the project scope primarily from a technology perspective, rather than with a business focus, thereby contributing to overscoping. Overall this cause was graded as having the second largest impact on overscoping, despite one questionnaire respondent (an RT leader) disagreeing to this cause. Our results support the findings from related papers (Aurum and Wohlin, 2005a; Selby and Cusumano, 1998; DeMarco and Lister, 2003; Khurum et al, 2007; Neumann-Alkier, 1997; Rosca et al, 1997) that stress the importance of selecting requirements aligned with the overall business goals and discarding others as early as possible. In addition, failure of stakeholders to concur on project goals was found by DeMarco and Lister to pose the biggest risk for a project (DeMarco and Lister, 2003).
A method for early requirements triage based on management strategies was proposed by Khurum et al (2007). Aurum and Wohlin (2005a) have proposed a framework for aligning requirements with business objectives. Rosca et al (1997) mention that the most demanding characteristic of business is the likelihood of change which can not be fully controlled. This can be managed when business objectives are clear to the software developers, thus enabling them to manage a system requiring modifications while meeting the business objectives (Selby and Cusumano, 1998). Finally, Karlsson et al (2007a) mentioned the lack of common goals and visions as a challenge in achieving good cooperation, quoting their responders: 'If everyone has the same goal and vision, then everyone works in the right direction.'

- *weak process adherence (+C7) and scope and deadline dictated by management (+C8).* These two causes were mentioned in the questionnaires, though none of them were seen as having any major impact on overscoping. Karlsson et al (2007a) found that weak process adherence may be caused both by high process complexity, as well as, lack of time for process implementation. The latter could be a consequence of overscoping. The direction of causal relationship between overscoping and process adherence remains to be investigated.

7.2 The effects of overscoping (RQ2)

The results indicate that overscoping may lead to a number of effects (or consequences), many of which are judged to be serious and potentially very costly for the company. Several of the identified effects may be in line with held beliefs about what overloading a project with too much work may lead to. The aim of this study is to investigate if such beliefs can be supported by empirical evidence or not, and if more surprising phenomena arise in relation to a specific, real-world overscoping situation.

- *many changes after the project scope is set (E1).* The results show that overscoping leads to a large number of scope changes (experienced by all responders and impact graded as critical or major by all six questionnaire responders). This confirms evidence provided by Harker et al (1993) that requirements are not static and, thus, are hard to capture or classify. In addition, requirements volatility is mentioned as one of the challenges in MDRE by Karlsson et al (2007a) and identified by Ramesh et al (2010) as one of the 14 assumptions underlying agile software development. Furthermore, origins of requirements volatility have been listed (Harker et al, 1993). Despite this awareness, causes for requirements volatility have not been empirically explored. Our results highlight overscoping as one possible cause of late requirement changes. Furthermore, our results confirm that it is challenging to manage requirement changes.

- *quality issues (E2).* The results indicate this as an important effect of overscoping (experienced and agreed for both interviews and questionnaires, and graded as having critical or major impact). This confirms that the quality of requirements engineering determines the software quality, as reported, e.g. by Aurum and Wohlin (2005b). In addition, our results highlight overscoping as a potential reason for quality issues.

- *delays (E3).* This study shows (with a high degree of alignment between interviewees and questionnaire responses) that delays can be an effect of overscoping. Within MDRE, delays in launching products can be very costly and result in loss of market shares (Sawyer et al, 1999; Sawyer, 2000; Regnell and Brinkkemper, 2005; Karlsson et al, 2007a). Therefore, the insight that overscoping may have this effect is important evidence that indicates that overscoping is a (potentially) serious risk.

- *customer expectations are not always met (E4).* Our results indicate that overscoping can have the effect of failing to meet customer expectations. This could be explained by an overloaded project having no time or capacity neither to analyse or implement new requirements, nor to validate if market or customer needs could have changed. Furthermore, Karlsson et al (2007a) reported failure to meet customer needs as one of the risks of developing products based on a technology focus (root cause RC6b). Another crucial part of producing software products that will satisfy the customers, as pointed out by Aurum and Wohlin (2005b), is working with RE throughout the project life cycle (as opposed to upfront requirements detailing, C5). The results of this study highlight the importance of selecting a feasible scope as one factor to consider when attempting to better understand and capture the customers' needs.

- *communication gaps (E5).* Our results indicate that overscoping may cause increased communication gaps. (Roughly half of our interviewees and questionnaire respondents mentioned and agreed to this effect.) This may be explained by the tendency to deflect by blaming others when under pressure, rather than cooperate to solve problems together. Furthermore, interviewees described that the many changes resulting from overscoping (E1) were badly communicated to the product unit and resulted in false error reports being filed on changed, but not updated requirements. This in turn, caused irritation among the development teams and further increased the communica tion gaps. Similarly, Karlsson et al (2007a) reported that constant inflow of requirements (cause C1) caused decision conflicts between marketing and development roles.

- *challenge to keep SRS updated (E6).* The majority of the respondents confirmed that overscoping increases the challenge to keep the SRS updated. When the SRS is detailed upfront (C5), the combination of the two (overscoping) effects E1 (many scope changes) and E1b (decreased motivation) lead to an increased need, but a lower motivation to update the SRS. This complements previous work, which reports requirements volatility as a common challenge for software projects (Harker et al, 1993; Hood et al, 2008; Jönsson and Lindvall, 2005; Wiegers, 2003) and that the view of RE as concerning a static set of requirements is inappropriate (Hall et al, 2002; Harker et al, 1993). In addition, Berry et al report that time and resources are never sufficient to keep the documentation updated and that scope creep occurs when programmers code while the documentation keeps changing (Berry et al, 2010). Furthermore, our study highlights that the challenge of keeping the SRS updated is increased as an effect of overscoping. Harker et al (1993) proposed to address this challenge by defining a minimum critical specification combined with incremental deliveries (i.e. +P6) and thereby gradually providing more value. Further research is needed to investigate if the methods proposed to address the challenge of updating the requirements documentation could also minimize this effect for overscoping.

- *overtime (+E7), changed/cancelled product plans (+E8), low priority for administrative tasks (+E9).* These effects were mentioned in the validation questionnaires and each got one count of critical impact. Further investigations are needed to validate their relationship to overscoping.

7.3 How agile RE practices may impact overscoping (RQ3)

Our study identifies that three of the agile RE practices being introduced at the case company may impact several of the causes and root causes of overscoping. In addition, three more practices were suggested by questionnaire respondents as addressing overscoping. The details of how the identified agile RE practices may impact overscoping (mentioned root causes can be seen in Figure 4.2) are discussed below. We interpret the results as an indication that overscoping is still a challenge for the case company, though more manageable with the (partly implemented) agile RE practices. Further investigations are needed to fully understand the situation in the agile context.

- *one continuous scope and release planning flow (P1)* is experienced by the responders to directly impact cause C2 (no overview of software resource availability) by enabling transparency and insight into the full project scope and into the current workload of the software unit. The increased visibility of the load and available resource capacity to both business and software unit may bridge several communication gaps identified as root cause of overscoping, i.e. RC1c, RC3d and RC4b. This practice covers the agile RE practices of requirements prioritization and constant re-planning for the high-level requirements (Ramesh et al, 2010). Our results confirm the findings of Dybå and Dingsøyr that managers of agile companies are more satisfied with the way they plan their projects than are plan-based companies (Dybå and Dingsøyr, 2008). Furthermore, our study also corroborates the findings that agile prioritization of the scope in combination with a stage-gate model at the feature level can avoid delaying critical features and also provides early feedback on features (Karlström and Runeson, 2005). However, achieving correct high-level cost and schedule estimation has been identified as a challenge also for agile project (Ramesh et al, 2010), which may be one reason why overscoping remains an issue for the case company.

- *Cross-functional development teams (P2)* are indicated by our results as improving several of the communication gaps identified by our study as important root causes to overscoping (i.e. RC1c, RC2a, RC3d, RC4b, RC6d). This case company practice is equivalent to the agile RE practice of preferring face-to-face requirements communication over written documentation (Beck et al, 2012) in combination with agile prioritization and constant re-planning at the detailed requirements level (Ramesh et al, 2010). At this detailed requirements level, cost and schedule estimations in an agile fashion (by only allowing additions when simultaneously removing something less prioritized) have been found to be efficient (Ramesh et al, 2010; Karlström and Runeson, 2005) and eliminate the 'requirements cramming' problem (Karlström and Runeson, 2005), which is equivalent to overscoping. Other studies have found that communication within development teams is improved by agile practices, but that communication towards other (dependent) teams remains a challenge (Karlström and Runeson, 2005; Pikkarainen et al, 2008). This challenge is addressed with P2 by including competence covering all the involved functional areas within the same team (thus, impacting root causes RCicii, RC2a, RC4b and RC6dii). Furthermore, the agile RE practice of including a customer representative in the development teams is summarized by Dybå and Dingsøyr (2008) as improving the communication been customer and engineers, while filling this role can be stressful and challenging (Karlström and Runeson, 2005; Ramesh et al, 2010).

- *Gradual and iterative requirements detailing (P3)* is seen (by our interviewees) to decrease the total lead time for development of a feature (root cause RC1b) by delaying the detailing of requirements until they are actually needed for design and development. This in turn reduces the amount of requirement changes within the (shorter) time frame for the feature development, which in a market with high requirements volatility is a significant improvement. It may also reduce the communication gaps that occur due to the timing aspect of detailing requirements before design and implementation starts, i.e. root causes RC3d, RC4a, RC4b. The case company practice P3 is equivalent to the agile practice of iterative RE (Ramesh et al, 2010).

7.4 Threats to validity and limitations

As for every study there are limitations that should be discussed and addressed. These threats to validity and steps taken to mitigate them are reported here based on guidelines provided by Robson (2002) for flexible design studies. Another important aspect for the quality of a flexible design research is the investigator (Robson, 2002), and for this study all researchers involved have previous experience in conducting empirical research, both interview studies and surveys.

7.4.1 Description validity

Misinterpretation (Robson, 2002) of the interviewees poses the main threat to description validity. This threat was addressed in several ways. The interviews were recorded and transcribed. To enhance reliability of the transcriptions, the person taking notes during the interviews also transcribed them. In addition, this person has worked for the case company for several years and is well versed in company culture and language. Also, data triangulation was applied to the transcriptions by another researcher performing an independent transcription and coding of two randomly selected interviews. Furthermore, the interviewees checked both the transcriptions and the results of the study for errors and misinterpretations. Finally, data triangulation was applied to the interview results by collecting additional viewpoints from six (other) practitioners through a questionnaire (Robson, 2002).

7.4.2 Interpretation validity

For this study, the main threat to valid interpretation has been the risk of imposing the hypothesis (formulated in phase one) onto the interviewees. To address this threat, open interview questions were always posed before asking specific questions based on the hypothesis. Furthermore, spontaneous descriptions of causes (without prompting) have been reported (as Experienced) separately from responses to follow-up questions on specific causes (as Agreed), see Section 5.1 and Table 4.3.

For phase three, the threat to valid description was addressed by the researchers jointly designing the questionnaire and the session held in connection to it. To ensure that all questionnaire responders correctly and uniformly understood the interview results, the results were presented to the participants. They could then ask for clarifications before filling out the questionnaire. The fact that questionnaire responders were confronted with a frame- work of results remains an open threat to interpretation validity. On the other hand, both interviewees and questionnaire respondents were explicitly encouraged to disagree and mention additional causes, effects and practices, which they also did. One of the main limitations of the study is the limited number of respondents. Although representatives from each of the covered units of the case company were involved in both interviews and validation questionnaire, the number of persons is relatively small and more factors may be identified by including additional viewpoints.

7.4.3 Theory validity

The main threat to theory validity for this study is the risk of missing additional or alternative factors. One source of this threat is the limited set of practitioners from which data has been gathered. Another potential source is the risk of observer biases limiting the study to the researcher's pre-knowledge of the company. This was a risk mainly in phase one and was addressed by involving the other researchers in discussing and reviewing the study design and the hypothesis which shaped the interview instrument. The fact that an additional main cause (i.e. C6) was identified as a result of the interviews shows that this bias was successfully addressed. However, identification of additional results in phase 3 may indicate that saturation and the full exploration of the problem under investigation is not yet reached. As the goal of this work is exploratory our aim is not to present or achieve a complete coverage of the problem under investigation.

The involvement of the researcher with work experience from the case company has played a vital role in the study. This has en- sured that the investigated problem is authentic and that the re- sults are derived though an interpretation of the data based on a deep understanding of the case and its context. However, the results are limited to the case company and there is a risk that other possible causes of overscoping experienced at other companies were not identified. This also applies to the set of agile RE practices, which are limited to the ones that were currently known and partly implemented at the case company at the time of the study.

Internal generalisability was addressed by sampling interviewees and questionnaire respondents from different parts of the company thereby selecting roles and responsibilities involved throughout the development life cycle. Even so, it was not possible to include representatives from sales and marketing (they were unavailable at the time of the study). However, the requirements team leaders provided some insight into these aspects based on their experience from contacts with customers and with sales and marketing roles.

Considering external generalisability, the results should be interpreted with the case company context in mind. External validity is addressed by using analytical generalization which enables drawing conclusions without statistical analysis and, under certain conditions, relating them also to other cases (Robson, 2002; Runeson et al, 2012). Within the scope of this paper, analytical generalization is argued by applying the making a case strategy ((Robson, 2002), p. 107) by analysing related work and reporting similarities, differences and disagreements to our results (see Section 7). This analysis builds a supporting argument towards external validity of our study by seeking data which is not confirming a pre-assumed theory. In addition, follow-up studies in other domains can be conducted to utilize the direct demonstration strategy to further address the threat to external validity (Robson, 2002).

8 Conclusions and further work

Decision making is at the heart of requirements engineering (RE) (Aurum and Wohlin, 2003) and within market-driven requirements engineering (MDRE) release planning is one of the most important and challenging tasks (Karlsson et al, 2007a,b; Regnell and Brinkkemper, 2005; Sawyer et al, 1999). Decisions concerning what to develop, and when, are inherently related to achieving customer satisfaction. Even though release planning (Karlsson and Ryan, 1997; Karlsson et al, 2007b; Regnell and Brinkkemper, 2005) is well researched, RE decision making is acknowledged as challenging (Alenljung and Persson, 2008; Aurum and Wohlin, 2003; Ngo-The and Ruhe, 2005) and scope creep is ranked as a serious project risk (Carter et al, 2001; Crockford, 1980; Iacovou and Dexter, 2004), other aspects of scope management have been less explored (van de Weerd et al, 2006). Furthermore, techniques for prioritizing requirements (Karlsson and Ryan, 1997; Karlsson et al, 2007b) often focus on planning the scope of a project as a discrete activity, or one in a series of releases (Ngo-The and Ruhe, 2005). Our previous work reported that scoping in an MDRE context is a continuous activity that may last throughout the entire project lifecycle (Wnuk et al, 2009). If not successfully managed, and more requirements are included into the project scope than can be handled with available resources the result is overscoping, i.e. the project 'bites off more than it can chew'.

Our study provides a detailed picture of factors involved in overscoping and confirms that scoping is a challenging part of requirements engineering and one of the risks in project management (Boehm, 1989; DeMarco and Lister, 2003; Legodi and Barry, 2010). Our results indicate that overscoping is mainly caused by the fast-moving market-driven domain in which the case company operates, and how this inflow of requirements is managed. In the early project phases, low involvement from the development-near roles in combination with weak awareness of overall goals may result in an unrealistically large project scope. Our study indicates that overscoping can lead to a number of negative effects, including quality issues, delays and failure to meet customer expectations. Delays and quality problems are expensive, not just considering the cost of fixing the quality issues, but also in loss of market shares and brand value (Regnell and Brinkkemper, 2005). Furthermore, we found indications that a situation of overscoping may cause even more overscoping, i.e. an organization may end up in a vicious cycle when overscoping ties up development resources which are then not available for participating in early project phases. Furthermore, overscoping leads to increased communication gaps, which in turn are root causes of overscoping.

Companies, such as our case company, that develop embedded software for a business domain with a high market pressure need an organizational set-up and process suited to efficiently managing frequent changes in a cost effective way. Development projects need to respond quickly to changes, while at the same time handling the complexity of developing software in a large-scale setting. Agile processes are claimed to be better adapted to managing change than phase-based ones. As one interviewee stated: 'The waterfall approach is good from a preparation perspective, if you can then stick to what is planned. But, since we live in a world that changes a lot it doesn't work after all.' Our study indicates, that despite introducing agile RE practices, overscoping is still an issue for the case company, although more manageable. We conclude that the improvements may be explained by the agile RE practices of continuous prioritization of the project scope, in combination with performing cost and schedule estimation, and gradual requirements detailing, in close collaboration within cross-functional teams, thereby closing a number of communication gaps. However, agile RE practices also pose challenges (Ramesh et al, 2010), e.g. communication between teams (Karlström and Runeson, 2005; Pikkarainen et al, 2008), difficulty in cost estimation (Ramesh et al, 2010).

This, in combination with a fast-moving, market-driven domain may explain why overscoping remains a challenge also with the agile development process.

The causes and effects unveiled through this study (summarized in Figure 4.2) can be used as a basis for identifying potential issues to address in order to avoid or alleviate an overscoping situation. For example, the root cause of low competence in cost estimations may be addressed by introducing techniques for improving cost estimation, which should lead to more realistic plans. Finally, supported by our findings of potentially serious effects of overscoping, we conclude that this phenomenon can be a major risk of requirements engineering and project management, complementary to the risk of scope creep mentioned by De Marco and Lister (2003).

Acknowledgments

We would like to thank all anonymous interviewees and questionnaire respondents for their invaluable contribution to this project. We would also like to thank Dr. Dietmar Pfahl for reviewing an early version of this paper. The project is partly funded by the Swedish Foundation for Strategic Research and VINNOVA (The Swedish Governmental Agency for Innovation Systems) within the EASE and UPITER Projects.

Bibliography

Abramovici M, Sieg O (2002) Status development trends of product lifecycle management systems. In: Proceedings of International Conference Integrated Product and Process Development, pp 55–70

Alenljung B, Persson A (2008) Portraying the practice of decision-making in requirements engineering: a case of large scale bespoke development. Requirements Eng 13(4):257–279

Aurum A, Martin E (1999) Managing both individual and collective participation in software requirements elicitation process. In: Proceedings of the14th International Symposium on Computer and Information Sciences, Turkey, (ISCIS'99), pp 124–131

Aurum A, Wohlin C (2003) The fundamental nature of requirements engineering activities as a decision-making process. Information and Software Technology 45(14):945–954

Aurum A, Wohlin C (2005a) Aligning requirements with business objectives: a framework for requirements engineering decisions. In: Proceedings of the Workshop on Requirements Engineering Decision Support, REDECS'05

Aurum A, Wohlin C (2005b) Requirements engineering: Setting the context. In: Aurum A, Wohlin C (eds) Engineering and Managing Software Requirements, Springer Berlin Heidelberg, pp 1–15

Beck K (1999) Extreme Programming Explained. Addison-Wesley

Beck K, Beedle M, Bennekum A, Cockburn A, Cunningham W, Fowler M, Grenning J, Highsmith J, Hunt A, Jeffries R, Kern J, Marick B, Martin R, Mellor S, Schwaber K, Sutherland J, Thomas D (2012) The agile manifesto. `http://agilemanifesto.org/`

Berry D, Czarnecki K, Antkiewicz M, AbdElRazik M (2010) Requirements determination is unstoppable: An experience report. In: Proceedings of the 18th IEEE International Requirements Engineering Conference (RE'10), pp 311 –316

Bjarnason E (2012) Case study material (interview instrument, questionnaire, etc) for the before and after (bna) study. `http://serg.cs.lth.se/research/experiment\_packages/bna/`

Bjarnason E, Wnuk K, Regnell B (2010) Overscoping: Reasons and consequences; a case study on decision making in software product management. In: Proceedings of the Fourth International Workshop on Software Product Management (IWSPM'2010), pp 30 –39

Bjarnason E, Wnuk K, Regnell B (2011a) A case study on benefits and side-effects of agile practices in large-scale requirements engineering. In: Proceedings of the 1st Workshop on Agile Requirements Engineering, ACM, New York, NY, USA, AREW '11, pp 3:1–3:5

Bjarnason E, Wnuk K, Regnell B (2011b) Requirements are slipping through the gaps; a case study on causes amp; effects of communication gaps in large-scale software development. In: Proceedings of the 19th IEEE International Requirements Engineering Conference (RE'2011), pp 37 –46

Boehm B (1989) Software risk management. Tutorial notes, IEEE Computer Society Press

Carlshamre P (2002) A usability perspective on requirements engineering – from methodology to product development. PhD thesis, Linköping University, Sweden

Carlshamre P, Sandahl K, Lindvall M, Regnell B, Natt och Dag J (2001) An industrial survey of requirements interdependencies in software product release planning. In: Proceedings of the Fifth IEEE International Symposium on Requirements Engineering (RE 2001), pp 84–91

Carter R, Antón AI, Dagnino A, Williams L (2001) Evolving beyond requirements creep: A risk-based evolutionary prototyping model. In: Proceedings of the 5th IEEE International Symposium on Requirements Engineering (RE'01, pp 94–101

Crockford N (1980) An introduction to risk management. Woodhead-Faulkner

DeBaud J, Schmid K (1999) A systematic approach to derive the scope of software product lines. In: Proceedings of the 21st International Conference on Software Engineering (ICSE 1999), pp 34–43

DeMarco T, Lister T (2003) Risk management during requirements. Software, IEEE 20(5):99 – 101

Dybå T, Dingsøyr T (2008) Empirical studies of agile software development: A systematic review. Inf Softw Technol 50(9-10):833–859

Ebert C, De Man J (2002) e-r & d effectively managing process diversity. Ann Softw Eng 14(1-4):73–91

El Emam K, Madhavji N (1995) A field study of requirements engineering practices in information systems development. In: Proceedings of the Second IEEE International Symposium on Requirements Engineering RE'95, pp 68 – 80

Faulk S (2001) Product-line requirements specification (prs): An approach and case study. In: Proceedings of the Fifth IEEE International Symposium on Requirements Engineering, IEEE Computer Society, Washington, DC, USA, RE '01, pp 48–

Fricker S, Gorschek T, Myllyperkiö P (2007) Handshaking between software projects and stakeholders using implementation proposals. In: Proceedings of the 13th international working conference on Requirements engineering: foundation for software quality, Springer-Verlag, Berlin, Heidelberg, REFSQ'07, pp 144–159

Gemmer A (1997) Risk management: moving beyond process. Computer 30(5):33 –43

Glinz M, Berner S, Joos S (2002) Object-oriented modeling with adora. Information Systems 27:425–444

Gorschek T, Wohlin C (2006) Requirements abstraction model. Requirements Engineering Journal 11:79–101

Hall T, Beecham S, Rainer A (2002) Requirements problems in twelve software companies: an empirical analysis. IEEE Software 149(5):153 – 160

Harker S, Eason K, Dobson J (1993) The change and evolution of requirements as a challenge to the practice of software engineering. In: Proceedings of IEEE International Symposium on Requirements Engineering RE'93, pp 266 –272

Hood C, Wiedemann S, Fichtinger S, Pautz U (2008) Change management interface. In: Requirements Management, Springer Berlin Heidelberg, pp 175–191

Iacovou C, Dexter A (2004) Turning around runaway information technology projects. Engineering Management Review, IEEE 32(4):97 –112

Jönsson P, Lindvall M (2005) Impact analysis. In: Aurum A, Wohlin C (eds) Engineering and Managing Software Requirements, Springer Berlin Heidelberg, pp 117–142

Jørgensen M, Shepperd M (2007) A systematic review of software development cost estimation studies. IEEE Trans Softw Eng 33(1):33–53

Kabbedijk J, Brinkkemper S, Jansen S, van der Veldt B (2009) Customer involvement in requirements management: Lessons from mass market software development. In: Proceedings of the 17th IEEE International Requirements Engineering Conference, (RE'09), pp 281 –286

Karlsson J, Ryan K (1997) A cost-value approach for prioritizing requirements. IEEE Software 14(5):67–74

Karlsson L, Dahlstedt sG, Regnell B, Natt och Dag J, Persson A (2007a) Requirements engineering challenges in market-driven software development - an interview study with practitioners. Inf Softw Technol 49(6):588–604

Karlsson L, Thelin T, Regnell B, Berander P, Wohlin C (2007b) Pair-wise comparisons versus planning game partitioning–experiments on requirements prioritisation techniques. Empirical Softw Engg 12(1):3–33

Karlström D, Runeson P (2005) Combining agile methods with stage-gate project management. Software, IEEE 22(3):43 – 49

Khurum M, Aslam K, Gorschek T (2007) A method for early requirements triage and selection utilizing product strategies. In: Proceedings of the 14th Asia-Pacific Software Engineering Conference, IEEE Computer Society, Washington, DC, USA, APSEC '07, pp 97–104

Konrad S, Gall M (2008) Requirements engineering in the development of large-scale systems. In: Proceedings of the 16th International Requirements Engineering Conference (RE 2008), pp 217–222

Legodi I, Barry M (2010) The current challenges and status of risk management in enterprise data warehouse projects in south africa. In: Technology Management for Global Economic Growth (PICMET), 2010 Proceedings of PICMET '10:, pp 1 –5

Lethbridge T, Singer J, Forward A (2003) How software engineers use documentation: the state of the practice. Software, IEEE 20(6):35 –39

Lixin Z (2008) A project human resource allocation method based on software architecture and social network. In: Proceedings of the 4th International Conference on Wireless Communications, Networking and Mobile Computing, 2008. WiCOM '08., pp 1 –6

Macaulay L (1993) Requirements capture as a cooperative activity. In: Requirements Engineering, 1993., Proceedings of the IEEE International Symposium on, pp 174 –181

Medvidovic N, Grünbacher P, Egyed A, Boehm BW (2003) Bridging models across the software lifecycle. J Syst Softw 68(3):199–215

Myers M, Avison D (2002) Qualitative research in information systems

Neumann-Alkier L (1997) Think globally, act locally - does it follow the rule in multinational corporations? In: ECIS'97, pp 541–552

Ngo-The A, Ruhe G (2005) Engineering and Managing Software Requirements, Springer, chap Decision Support in Requirements Engineering, pp 267–286

Nuseibeh B (2001) Weaving together requirements and architectures. Computer 34(3):115 –119

Phillips J, Bothell T, Snead G (2012) The Project Management Scorecard. Improving Human Performance Series, Taylor and Francis

Pikkarainen M, Haikara J, Salo O, Abrahamsson P, Still J (2008) The impact of agile practices on communication in software development. Empirical Softw Engg 13(3):303–337

Pohl K, Bockle G, van der Linden F (2005) Software Product Line Engineering: Foundations, Principles and Techniques. SpringerVerlag

Potts C (1995) Invented requirements and imagined customers: requirements engineering for off-the-shelf software. In: Proceedings of the Second IEEE International Symposium on Requirements Engineering (RE 95), pp 128–130

Project Management Institute (2000) A Guide to the Project Management Body of Knowledge (PMBOK Guide), 2000, Project Management Institute, Four Campus Boulevard, Newtown Square, PA 19073-3299, USA, chap Chapter 5: Project Scope Management, pp 47–59

Ramesh B, Cao L, Baskerville R (2010) Agile requirements engineering practices and challenges: an empirical study. Information Systems Journal 20(5):449–480

Regnell B, Brinkkemper S (2005) Engineering and Managing Software Requirements, Springer, chap Market–Driven Requirements Engineering for Software Products, pp 287–308

Regnell B, Berntsson Svensson R, Wnuk K (2008) Can we beat the complexity of very large-scale requirements engineering? In: Lecture Notes in Computer Science, vol 5025, pp 123-128

Robson C (2002) Real World Research. Blackwell Publishing

Rosca D, Greenspan S, Feblowitz M, Wild C (1997) A decision making methodology in support of the business rules lifecycle. In: Proceedings of the Third IEEE International Symposium on Requirements Engineering (RE'97), pp 236 –246

Runeson P, Host M, Rainer A, Regnell B (2012) Case Study Research in Software Engineering: Guidelines and Examples. John Wiley & Sons

Sangwan R, Bass M, Mullick N, Paulish D, Kazmeier J (2006) Global Software Development Handbook. Auerbach series on applied software engineering, Taylor and Francis

Sawyer P (2000) Packaged software: Challenges for re. In: Proceedings of the Sixth International Workshop on Requirements Engineering: Foundations of Software Quality (REFSQ 2000), pp 137–142

Sawyer P, Sommerville I, Kotonya G (1999) Improving market-driven re processes. In: Proceedings of the International Conference on Product-Focused Software Process Improvement (Profes '99)

Schmid K (2002) A comprehensive product line scoping approach and its validation. In: Proceedings of the 24th International Conference on Software Engineering (ICSE 2002), pp 593–603

Schwaber K, Beedle M (2002) Agile software development with scrum. Series in agile software development, Prentice Hall

Selby RW, Cusumano MA (1998) Microsoft secrets. Simon and Schuster

Svahnberg M, Gorschek T, Feldt R, Torkar R, Saleem SB, Shafique MU (2010) A systematic review on strategic release planning models. Inf Softw Technol 52(3):237–248

van de Weerd I, Brinkkemper S, Nieuwenhuis R, Versendaal J, Bijlsma L (2006) Towards a reference framework for software product management. In: Proceedings of the 14th IEEE International Requirements Engineering Conference (RE 2006), pp 319–322

van Lamsweerde A (2003) From System Goals to Software Architecture. In: Formal Methods for Software Architectures, pp 25–43

Wiegers K (2003) Software Requirements: Practical Techniques for Gathering and Managing Requirements Throughout the Product Development Cycle. Addison-Wesley

Wnuk K, Regnell B, Karlsson L (2009) What happened to our features? visualization and understanding of scope change dynamics in a large-scale industrial setting. In: Proceedings of the 17th IEEE International Requirements Engineering Conference (RE 2009), pp 89–98

Wohlin C, Aurum A (2005) What is important when deciding to include a software requirements in a project or release? In: Proceedings of the International Symposium on Empirical Software Engineering (ISESE 2005), pp 246–255

Paper V

Factors Affecting Decision Outcome and Lead-time in Large-Scale Requirements Engineering

Krzysztof Wnuk[1], Jaap Kabbedijk[2], Sjaak Brinkkemper[2], Björn Regnell[1]
[1]Department of Computer Science,
Lund University, Sweden
{Krzysztof.Wnuk,Bjorn.Regnell}@cs.lth.se
[2]Department of Information and Computing Sciences
Utrecht University
{J.Kabbedijk,S.Brinkkemper}@uu.nl

under revision for
Software Quality Journal 2012

Abstract

Lead-time is crucial for decision making in market-driven requirements engineering. In order to identify what factors influence the decision lead-time and outcome, we conducted a retrospective case study at a large product software manufacturer and statistically analyzed seven possible relationships among decision characteristics. Next, we further investigated relationships among decision characteristics in a survey among industry participants. The results show that the number of products affected by a decision could increase the time needed to take a decision. Results also show that when a change request originates from an important customer, the request is faster accepted.

1 Introduction

Requirements Engineering (RE) addresses the critical problem of designing the right software for the customer (Aurum and Wohlin, 2005). In Market–Driven Requirements Engineering (MDRE), the content of the product has to be aligned with the targeted market needs to create a profitable software product (Regnell and Brinkkemper, 2005). For large MDRE projects, with thousands of continuously arriving (Karlsson et al, 2007a) potential requirements, deciding which requirements should be implemented is far from trivial.

Large companies often use the software platform concept, also known as Software Product Lines (SPL) (Pohl et al, 2005). SPL helps to decrease the cost and to increase the ability to provide an individualized software product. Moreover, SPL allow software development organizations to reuse a common technology base to meet customers' needs. However, the cost for this greater degree of reuse and increased productivity is increased complexity of coexisting product variants and a more complex decision making process.

The requirements selection process is a complex decision problem, bringing up several challenges, e.g. shifting goals, time stress (Alenljung and Persson, 2008) and uncertain estimates (Karlsson and Ryan, 1997) just to name a few. To effectively improve RE decision–making, more effort should be dedicated towards decision–making aspects identification (Alenljung and Persson, 2008; Natt och Dag et al, 2005). In particular, it is important to explore additional factors influencing both the time needed to make the decision (also called the decision lead–time) as well as the outcome of the decision process.

In this paper, a retrospective analysis of the decision making process in a large–scale MDRE and SPL project is performed with the aim for identifying which characteristics of change requests, i.e. number of products, release number, type of customer, may influence the decision lead–time and the decision outcome. The decision lead–time is in this context defined as time required to analyze the impact of a decision. The decision outcome is in this context defined as a specific outcome of the decision process, namely acceptance or rejection. For brevity, we use decision outcome throughout the paper. The results from analyzing the decision–log are further investigated in a survey among 50 industry respondents.

The main goals for the paper are threefold: (1) to explore possible factors that may influence decision lead–times, (2) to investigate the possible factors that may influence decision outcomes and (3) to investigate if the decision lead–time affects the decision outcome.

Partial results from this study have previously been published as workshop publications in (Kabbedijk et al, 2010). This paper extends our previous work by: (1) validating the results the decision log analysis in a survey, (2) extending the analysis of the results regarding factors that affect the decision lead-time and the relationship between the decision lead-time and the decision outcome (3) extending the analysis of related work, (4) extending the interpretation of the results in the light of the related work.

The paper is structured as follows. Related work is discussed in Section 2, followed by a description of the case company in Section 3. Our research design and research questions are outlined in Section 4. Next, we present the results of the statistical analysis of the decision logs and the survey in Section 5. We conclude the paper and present the future work in Section 6.

2 Related Work

Decision making is an important aspect of requirements engineering (Alenljung and Persson, 2008; Aurum and Wohlin, 2003; Evans et al, 1997) and significantly impacts requirements management. As stated by DeGregorio (1999), requirements management is not possible without decision management. Therefore, understanding of the nature of the decisions made in the RE process is necessary for improving it (Aurum and Wohlin, 2003). Despite an increasing awareness for supporting RE decision making, research in this area is still "in its infancy" (Ngo-The and Ruhe, 2005).

The requirements engineering process is a decision rich activity for which decisions can range from the organization level to the project level (Aurum and Wohlin, 2003; Ngo-The and Ruhe, 2005). Moreover, since RE decision making is a knowledge–intensive activity that is performed in natural settings, it has to deal with the difficulties such as shifting, ill–defined or competing goals and values (Klein et al, 1995). As a result, the risk of making inappropriate decisions is high and the consequences of made decisions can be serious. Furthermore, RE decisions are often semi–structured or unstructured and made only once, which make the evaluations of the decision outcomes difficult (Ngo-The and Ruhe, 2005). Moreover, Strigini (1996) stressed a lack of objective criteria for guiding making decisions, e.g. based on statistics about past experience which results in important decisions often depending on subjective judgments. Thus, empirical investigation of the factors that affect decision outcomes is important as it can contribute to more continuous, controllable and structured requirements engineering decision making.

Several researchers have looked into modeling decision–making in software and requirements engineering. Ashrafi (1998) proposed a decision–making model that addresses various software quality aspects. Rolland et al (1995) proposed a decision making meta–model for requirements engineering process that captures both how and why the requirements engineering activities are performed. Wild et al (1994) modeled the software development process as a set of problem solving activities (decisions). Ruhe (2005) modeled release planning decisions by combining computational knowledge intelligence and experience of decision makers or by using linear programming (Ruhe, 2009). van den Akker et al used integral linear programming to find an optimal set of requirements within the given resource constraints that can maximize the revenue (van den Akker et al, 2008). Li et al focused on time schedling aspect of release planning (Li et al, 2010). Regnell and Kuchcinski used constraint programming (Regnell and Kuchcinski, 2011) to model release planning decision making while Egyed et al (2006) proposed using constraints programming for reducing the number of possible software design decisions. Karlsson (1997) promoted a cost–value approach to support requirements prioritization which was later experimentally compared to other prioritization techniques (Karlsson et al, 2007b). Ruhe (2009) covered supporting product release decisions on various levels by modeling the release planning criteria and constraints. However, the mentioned methods mainly focus on the task of reducing the number of possible decision or assigning features to releases according to given criteria, while this study focuses on understanding the factors that may affect both decision lead–times and outcomes.

Among the challenges in RE decision making Alenljung et al (2008) listed: ill–structured problems, uncertain environments, shifting goals, action and feedback loops, time stress, high stakes, multiple player situations and organizational goals and norms. Ngo-The and Ruhe (2005) argued that requirements decisions are hard because of the incompleteness of the available information and any notion of strict optimality is not appropriate in this context. Karlsson et al (2007a) listed release planning based on uncertain estimates as one of the challenges in MDRE that is related to RE decision making. Another challenging aspect of decision making; mentioned by Fogelstrom et al (2009); is finding the right balance between the commercial requirements selected over internal quality requirements, also mentioned by Karlsson et al (2007a). Furthermore, requirements prioritization (Karlsson and Ryan, 1997) was recognized as challenging because of, e.g. conflicting priorities between stakeholders (Berander and Andrews, 2005) or the complex dependencies between requirements (Cleland-Huang et al, 2005). Finally, several researchers stressed the need for empirical studies in RE decision making process to create a coherent body of knowledge in RE decision making and to improve requirements engineering (Alenljung and Persson, 2008; Aurum and Wohlin, 2003; Berander and Andrews, 2005).

Despite the above mentioned need for more empirical studies and several reported studies that outline challenges in requirements engineering decision making, the number of publications that empirically investigate factors affecting decision making in requirements engineering is still low. Among the reported studies, Wohlin and Aurum (2005) investigated the criteria used in the requirements selection process for a project or release and reported that business–oriented and management–oriented criteria are more important than technical concerns. Wnuk et al (2009) investigated the reasons for excluding features from the project's scope reporting that the stakeholder business decision is the dominant reasons for feature exclusions. Barney et al (2008) reported that the client and market base of the software product are the dominant factors that affect the decision to implement specific requirements. Moreover, Barney et al (2008) stressed that factors such as maturity of the product, the marketplace in which it exists and the available development tools and methods also influence the decision of whether or not include requirements in a software product. To the best of our knowledge, no study had yet attempted to investigate factors that affect both decision lead–times and decision outcomes.

While looking more generally at related work in decision making, Khatri (2000) discussed the intuition's role in decision making whereas Messerschmitt and Szyperski (2004) discussed the "marketplace issues" that may affect software project planning and decision making. Hogarth (1975) proposed a relationship function between the decision time and the task complexity. Saliu and Ruhe (2005) suggested that there is a relationship between decision outcomes and release planning. A similar relationship was suggested by Bagnall (2001) but, as in Ruhe and Saliu (2005), the relationship hasn't been named. Zur and Breznitz (1981) suggested a relationship between the time pressure, people's experience and the risks of their choice behaviors. Hallowell (1996) suggested a relationship among customer satisfaction, loyalty and profitability. However, the mentioned relationships haven't been empirically investigated in a large-scale MDRE context and especially in relation to decision lead–times and decision outcomes.

3 Case Company Description

The paper reports results from a content analysis (Lethbridge et al, 2005) of the record of decisions made in an industrial project at the large company using the SPL approach (Pohl et al, 2005). The company operates globally selling embedded systems and has more than 4000 employees. The core of the software part of embedded systems is called a *platform* and corresponds to the common code base of the SPL (Pohl et al, 2005). There are several consecutive releases of the *platform* in which each of them is a basis for one or more products that reuse the platform's functionality and qualities. A major platform release has approximately a two year lead–time from start to launch, and is focused on functionality growth and quality enhancements for a product portfolio. Minor platform releases are usually focused on the platform's adaptations to the different products that will be later launched. The stage–gate model with several increments (Cooper, 1990) is used by the company. The scope of the core release project is constantly changing during this process, from the initial roadmap extraction which is a basics for creating high level features to the final milestone of the requirements management process after which the development phase starts.

The case company utilizes the concept of a *feature* as an entity for making scoping decision. A *feature* is defined as a group of requirements that constitute new functionality enhancements to the platform upon which market value and implementation cost can be estimated. The project decision makers consider both internally issued features and features from external customers. Change requests to these features are performed constantly by stakeholders from inside and outside the company. The change control system is used in order to capture, track and assess the impact of changes (Leffingwell and Widrig, 2003; Kitchenham et al, 1999). The scope of each project is maintained in a document called the feature list, that is updated each week after a meeting of the change control board (CCB). The CCB exists of product and platform managers, complemented with other project stakeholders to a total of 20 members. The role of the CCB is to decide upon adding or removing features according to issued change requests. The decision process of the CCB is illustrated in Figure 5.1.

The CCB decision process is depicted in Figure 5.1. The process is similar to the processes described in the related literature (Leffingwell and Widrig, 2003; Kitchenham et al, 1999; Jonsson and Lindvall, 2005). The change requests are high level requests on feature level. After a change request is filed, its ambiguity and completeness are analyzed. This analysis is based on the quality gateway model (Natt och Dag et al, 2001), also called the "firewall" by Leffingwell and Widrig (2003). If the request is ambiguous or incomplete, it is sent back to the submitter to ask for a clarification, otherwise the request is put on the CCB agenda for performing the impact analysis. The impact analysis is performed by the appropriate Technical Groups that elicit and specify high–level requirements for a special technical area and Focus Groups that design and develop previously defined functionality. In this way, the impact of a change on the cost and functionality of the system as well as on customers and other external stakeholders is assessed (Leffingwell and Widrig, 2003). After the impact analysis, the request is presented at the CCB meeting and the change request is decided upon. When the analysis performed by a certain group is not clear enough, extra information can be requested before the final decision is made. If the request is accepted, the change is implemented, else the submitter gets a rejection notification.

All change requests and decisions made about them (including their rationale) are stored in the scope decision log of a project. In this sense, the company follows the advice of Aurum and Wohlin that the rationale and effects of RE decisions on software product should be tracked in order to support and improve RE activities (Aurum and Wohlin, 2003)

An example of an entry in the decision log is shown in Table 5.1. For reasons of confidentiality, we used fictive data. This decision log comprises a number of attributes like the change submitter and justification,

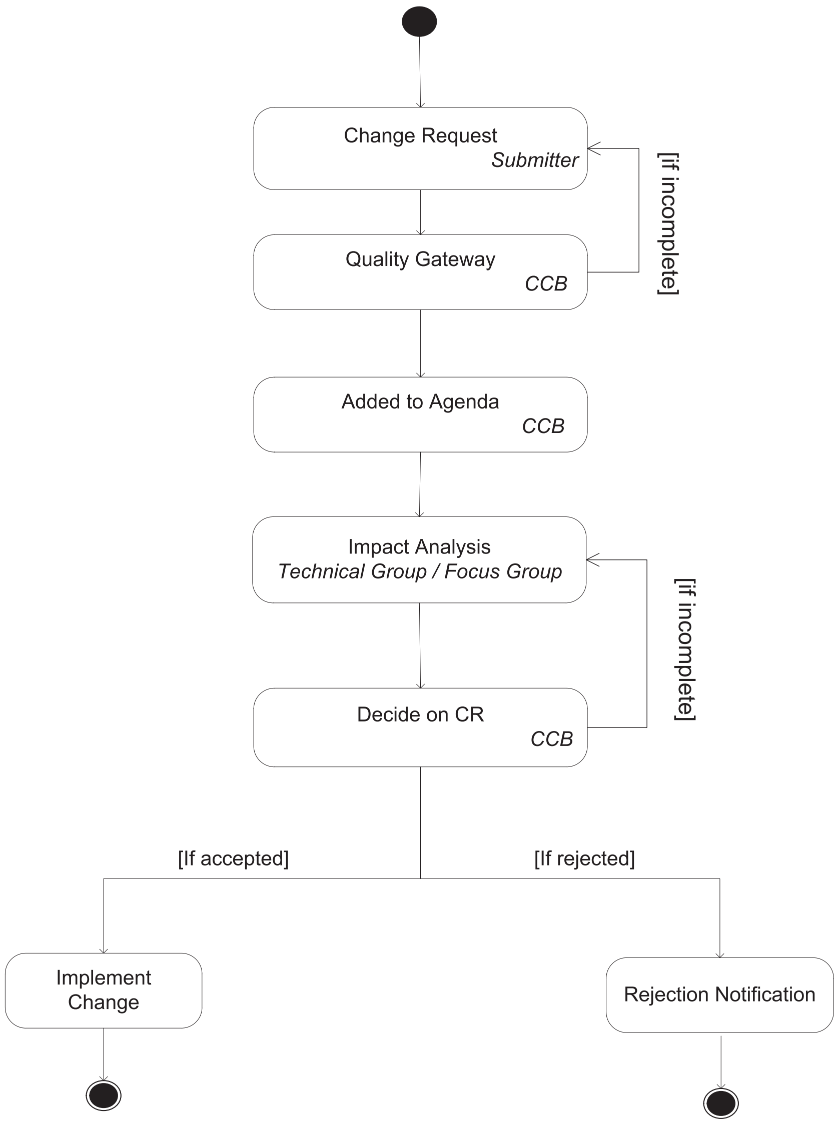


Figure 5.1: Change control board decision process

Table 5.1: Decision log entry example

ID	54
Change Request	HD resolution for video
Decision Outcome	Accepted
Comments	This will enlarge our market share in this sector
Description of proposed change	Add HD resolution for recording
Justification	Requested by a large provider
Proposition Area	Video
Main affected Technical Group	Video Group
Affected product	All products with a camera
Affected key customer	Customer X
Affected Functional Group	HD Group
Submittal Date	09-02-09
RM tool ID	10F1
Decision Date	18-02-09

the date that the request has been submitted and decided upon, the products impacted by a change, the release of the platform project impacted by a change, and the markets impacted by a change. The release of the platform project impacted by a change attribute is used to request a certain feature in an earlier release (the release number is low) or in a later release (the release number i high). For brevity, we will call this attribute release number throughout the paper. For this paper, we were granted access to an extensive decision log. This log contained 1439 change requests for all products planned to be released in 2008.

4 Research Design

Since the number of papers that investigate factors influencing RE decision making is low, see Section 2, our research was mainly exploratory and conducted in order to: (1) identify the main decision characteristics and (2) analyze the relationships between the identified characteristics. After identifying the characteristics, we formulated research questions, see Section 4.1 about relations within requirements engineering decision making and hypotheses based on these questions, see Section 5. We run statistical tests on empirical data to either accept or reject hypotheses and draw conclusions based on the test results. The results of the statistical analysis were further validated in a survey and interpreted in relation to related studies.

4.1 Research questions

Three research questions are investigated in this paper and are outlined in Table 5.2, complemented with aim and example answers for each question. The questions were shaped and inspired by the related literature outlined in Section 2. All three research questions are relationship questions (Easterbrook et al, 2008). The questions are further decomposed into hypotheses that were investigated using statistical tests, see Section 5.

Table 5.2: Research questions

Research question	**Aim**	**Example answers**
RQ1: Which decision characteristics affect the decision lead–time?	To understand which decision characteristics i.e, number of products, release number, type of customer have a significant impact on the decision lead–time	The number of products involved in the decision increases the decision lead–time. Decisions that are related to the current release or consider our largest customers have shorter lead–times.
RQ2: Which decision characteristics affect the decision outcome?	To understand the relation between the decision characteristics and the acceptance or rejection of a decision	The number of products involved in the decision decreases the probability of accepting this decision. Decisions that are related to the current release or issued by important customers are usually accepted.
RQ3: Is the decision outcome related to the decision lead–time	To understand the relation between the acceptance rate and the decision lead–time	Decisions with longer lead–time are more often rejected

The first research question (RQ1) is inspired by Hogarth (1975), who created a function on the relationship between the decision time and the task complexity. Hogarth stated that the amount of time needed to make a decision is an increasing function of the task complexity. After some point the costs of errors due to the task complexity becomes lower than the cost of time. This is the tilting point at which the amount of time becomes a decreasing function of the task complexity. In this paper, we empirically investigate the viewpoint of Hogart (1975) as well as we investigated further factors that may influence the decision lead–time, e.g. the type of the customer and the release number.

The second research question (RQ2) investigates the relationships between the decision characteristics and the decision outcome. This question is partly based on the work of Saliu and Ruhe (2005) and the work of Bagnall et al (2001) suggesting a relationship between decision outcomes and release planning. However, their work (Saliu and Ruhe, 2005; Bagnall et al, 2001) didn't suggest any explicit relationship between setting the requirements release time and the decision outcome. Therefore, RQ2 focuses on investigating if such relationships could be found.

Among other related studies, the paper by Hallowell (1996), suggested a relationship among customer satisfaction, loyalty and profitability. Thus, it is reasonable to assume a possible relationship between the fact that a request is filed by an important customer and its decision outcome. Software companies should keep their customers satisfied and thus they could accept requests of these customers faster than internal requests.

Research question RQ2 is also based on the work of Hogarth (1975). Since there is a tilting point in the relationship curve, there is a certain complexity level after which the decision maker decides the errors costs due to a wrong decision are lower than the costs of spending any more time on making the decision. From this point it is logical to state a hypothesis that negative decisions could be made and a relationship between the decision outcome and the number of products affected by a decision could exist.

The last research question (RQ3) is based on the work of Zur (1981) and our previous work (Wnuk et al, 2009). In our previous work (Wnuk et al, 2009), we reported that project management is more eager to accept features in the beginning of a large project and exclude features towards the end of the project due to time pressures and other unexpected difficulties. In a related paper, Zur (1981) claims a relationship between the time pressure people's experience and the risks of their choice behavior. Thus, we investigated in this study if longer lead–times impact decision outcomes.

4.2 Research methods

Case study and survey methods were selected for conducting this study. Both methods are considered as relevant for software engineering research (Easterbrook et al, 2008; Runeson and Höst, 2009). The details of the methods are outlined in the subsections that follow.

4.2.1 Case study

Case studies have been recognized as an appropriate method to understand complex social phenomena (Yin, 2008) and highly recommended for software engineering research (Runeson and Höst, 2009). We have used the analysis of electronic databases of work performed technique (Lethbridge et al, 2005) for data collection as it is a suitable technique for analyzing large amounts of data. The researchers were granted access to an extensive decision log of all products planned to be released in 2008 containing 1439 change requests. To address the risk of low control over the gathered information quality, the data was validated with one practitioner from the case company and analyzed by two authors of this paper to perform observer triangulation. Based on the decision characteristics (see Section 3),

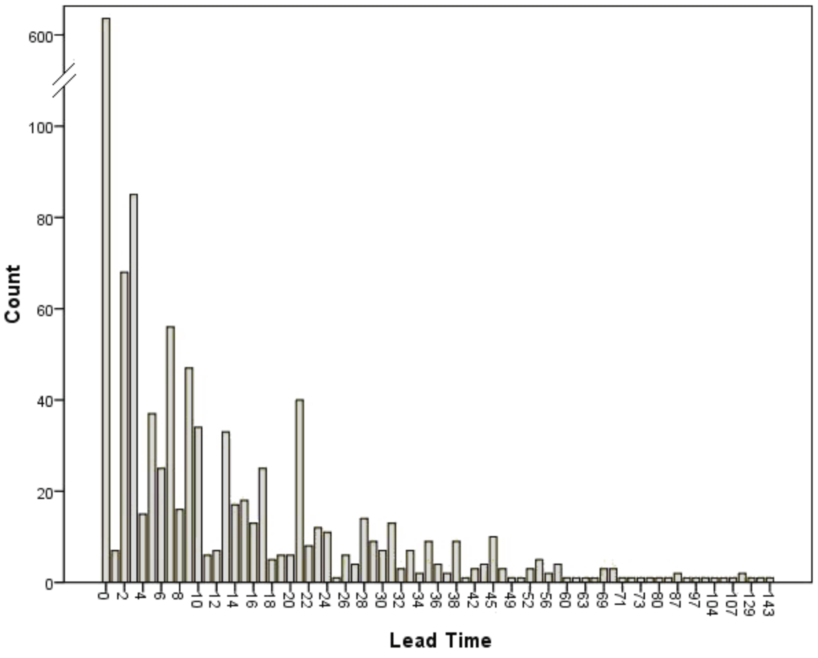


Figure 5.2: Number of decisions taking a certain lead-time

five variables were created for each decision.
1. *Lead-Time:* the duration between the moment a request was filed to the moment the decision was made by the CCB. The lead–time is measured in week days and not working days, so there could be a small difference in days between two decisions who took the same number of working days to be taken, due to weekends. Figure 5.2 gives an indication of how the lead–time is distributed. About half of the decisions are made the same day they are requested (686 decisions, 48%), but the 753 requests that are left can take up to 143 days before a decision is made.
2. *Number of Products Affected:* a number between one and fourteen (the total number of product for this software product line) indicating the number of different products for which the requirements would change if the request was accepted. We consider this attribute as a proxy for decision

complexity.
3. Release Number: a variable strongly related to the release method used within the case company. As described in Section 3, the product line platform of the case company is released in a heartbeat rhythm of one base release and four sequential releases. The release number variable indicates the specific number of the release affected by the change request. The higher the variable, the later the release is in the release heartbeat rhythm of the case company.
4. Type of Customer: a nominal variable used to indicate whether a request is filed by an important external customer or is a request coming from inside the company. External customers in this case are large partners of the case company who also help to bring the developed products to the market. Thus, we will refer to them as *important external customers.*
5. Decision Outcome: a variable of nominal level of measurement indicating whether or not a change request is accepted by the CCB.

4.2.2 Survey

We conducted a survey among 50 respondents from industry to validate the results from the case study as well as to strengthen the external validity of the study. The survey respondents were mainly working for companies producing product software using the SPL approach.

The questionnaire was created based on principles described by Kitchenham and Pfleeger (2002). The questionnaire contained a part dedicated to identify the context and background of the respondents, followed by a part focusing on their experiences considering possible relations in requirements engineering decision making. The questions identifying the respondents' context and background are based on the facets identified by Paech et al (2005).

The questions concerning the possible relationships within requirements engineering decision making were structured using a three-point Likert scale for effectively measuring the experiences of the respondents (Jacoby and Matell, 1971). We asked the respondents to state whether a certain characteristic influenced the decision lead–time in a positive, neutral or negative way. All relations examined based on the decision log were also inquired in the questionnaire. For example, a question from the survey was: "Please indicate how the number of products affected by the decision influences the time needed to take the decision". The answer categories were: "This makes the time to decide shorter", "No influence" or "This makes the time to decide longer". During the analysis, we rated the first answer as a score of -1, the second answer as 0 and the last answer as $+1$. This schema allowed to determine how strongly a certain decision characteristic influenced the decision lead–time or outcome. The survey questions can be accessed at (Wnuk, 2012) and in the Appendix.

4.3 Validity

We discuss the validity of research design and the results based on the classification proposed by Yin (2008).

4.3.1 Construct Validity

It is important to use the right sources for measuring the theoretical constructs (Yin, 2008). If we, for example, want to measure the time needed to take a decision, a reliable source is needed determine this amount of time. We analysed the decision log that was actively used in the decision making process at the case company. This decision log is an archival record, which could be considered as stable, exact and quantitative. Whenever decisions in the log were incomplete or ambiguous, we discussed them with the responsible product manager to avoid making wrong interpretations. These discussions can be seen as interviews we had with the responsible product manager. Both data collection methods are highly applicable to software engineering case studies (Runeson and Höst, 2009). Wohlin et al mentioned additional design threats to validity (Wohlin et al, 2000), namely the mono-operation and mono-method bias threats. These threats concern creating a bias while using respectively one case or method within the research. We ensured the validity on these levels by discussing all results with a responsible product manager and the use of several statistical and qualitative methods to analyse the data.

Construct validity of the survey part of the study is mainly concerned with the way how questionnaire questions were phrased. To alleviate this threat to construct validity, an independent senior researcher experienced in the topic reviewed the questionnaire. Moreover, we conducted a pilot study to measure the time required to conduct the survey and minimize the risk of misunderstanding or misinterpreting the survey questions by respondents. Further, the anonymity of questionnaire respondents was guaranteed which minimize the evaluation apprehension threat. Finally, the mono-operational bias threat is to partly alleviated as we managed to collect 50 responses.

4.3.2 Internal Validity

Threats to internal validity concern the investigated causal relationship between studies factors (Yin, 2008). In this study, we have minimized threats to internal validity by investigating as many possible factors that could influence the decision lead–time and outcome as it was possible with the given dataset. The identified relationships were confronted with the results from the survey in which these relationships were further tested. Finally, the potentially impacting additional confounding factors for the studied relationships were discussed in all cases in which the results from the case study and the survey were inconsistent (see Section 5).

To avoid stating false inferences (Yin, 2008), we have based our results on empirically derived data from a large company and confronted the results in a survey. Finally, we discuss the achieved results in Section 5, where we provide several possible explanations and possibilities, especially when the results from the case study and the survey are inconsistent.

4.3.3 External Validity

The external validity is considered as a main threat to validity in case studies due to difficulties to generalize from a single company study (Yin, 2008) even if the size of the data sample is large. To mitigate this threat, we have designed and conducted a survey in order to validate the findings from the case study. Since the majority of survey respondents worked in smaller companies with a typical project generating not more than 100 requests, we could further strengthen the generalizability of the results by comparing a large context with smaller contexts.

4.3.4 Reliability

In order to ensure the reliability of a study, it is important to have created a case study protocol and to maintained a case study database (Yin, 2008). In this way, the performed research could be retraced. We also stored all artifacts from the case study, so conclusions based on the evidence can be retraced as well. Further, we have published the survey questionnaire questions on-line (Wnuk, 2012) and described the sample population in Sections 4.2.2 and 5.2.1. However, we would like to stress that the data given by respondents is not based on any objective measurements and thus its subjectivity may affect the interpretability of the results.

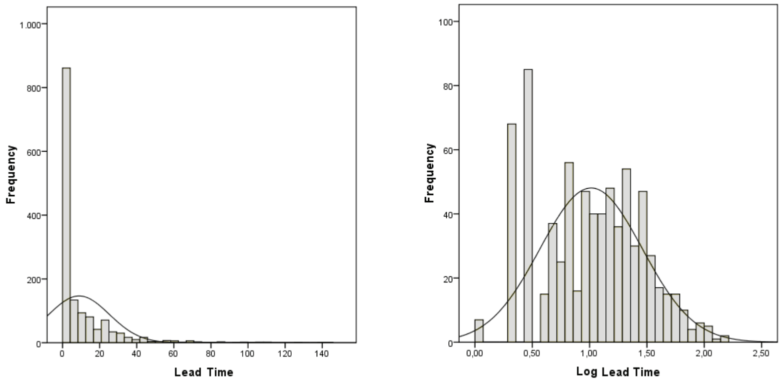
5 Results and Discussion

5.1 Test Selection

Selecting the appropriate test for analyzing the relationships is critical for getting reliable and scientifically sound results to base the conclusions on (Ott and Longnecker, 2008). The choice of the right statistical test is dependent on three major factors, namely (Sheskin, 2004):

- The level of measurement of the variables
- The distribution of the data
- The hypotheses that will be tested

We analyzed five different decision characteristics, which were all translated to quantitative, analyzable variables.



(a) lead–time (b) Log_{10} lead–time

Figure 5.3: Lead–time Gaussian curve fit

In order to perform parametric tests, all ratio level data should be distributed normally (Field, 2009). Since the variable lead–time is the only variable of ratio level of measurement, we ensured this variable complied to the condition stated before. The variable lead–time apparently described a log–normal distribution, so in order to be able to use this variable, the log_{10}-function of the variable was used for analysis. The detail of the transformation are depicted in Figure 5.3. The D'Agostino-Pearson test (1973) was used to see whether the log_{10}-function of the variable lead–time described a Gaussian curve, or was distributed differently. We tested the following hypotheses (H^0):

H^0_0: The sample is derived from a normally distributed population.

H^0_1: The sample is **not** derived from a normally distributed population.

When testing the kurtosis and skewness (DeCarlo, 1997) of the distribution, we found a result of $\chi^2(1, N = 753) = 35.3, p < .01$, which is below the critical value of 67.4 as can be found in the χ^2 distribution table. This means we can not reject H_0, so we can conclude that the log_{10}-function of the variable "lead–time" is **distributed normally** and we can use parametric tests on this variable. However, since the other analyzed variables are either of ordinal or nominal level of measurement, we also used non-parametric tests while analysing their influences and relationships.

5.2 Survey Data Analysis

The answers from the survey create variables of ordinal level of measurement. According to Stevens et al (1946) median and percentile scores should be used as ways of assessing these types of survey results. In our case, calculated medians are means and at least half of the sample has identified a negative relationship. When the median is positive, at least half of the sample in our study has identified a positive relationship, see Table 5.4 and a frequency table can be used to further analyze the results.

5.2.1 Demographics

The survey was answered by 50 respondents. 32% of the respondents came from The Netherlands, 14% from Sweden and 46% came from other countries, including US and UK. Software project (12%) and product manager (48%) roles dominated among our respondents, followed by senior management (12%), consultants (12%) and developers (6%). Our respondents reported, on average, 13 years of professional experience, with standard deviation of about 6 years. Three respondents indicated having less than 5 years of experience: (1) one project manager from the US who worked with off–the–shelf solutions in a small company reported having one year of experience, (2) one requirements engineer from The Netherlands working with bespoke software with an average of 100 change requests per project reported having 2 years of experience and (3) one product manager from The Netherlands working with off–the–shelf product with an average of 10 requests per project reported having 4 years of experience.

The majority of the respondents (68%) worked with companies, in which up to 100 persons were involved in the software engineering process. Further, 52% of the respondents created mostly off–the–shelf software, followed by bespoke software (28%). When looking at the number of change requests per project, a typical project generates not more than around 100 requests for over 70% of the respondents. Finally, 64% of the respondents reported using the SPL approach (Pohl et al, 2005).

5.3 Factors that affect the decision lead–time: RQ1

Table 5.4 shows a list of all hypotheses together with their survey result medians (last column). Column "Level of Significance" supplies all test results, together with their critical values for the analysis of the decision log. The last column in Table 5.4 represents the median score for the survey answers.

To investigate which decision characteristics have a significant impact on the decision lead–time, we have tested three hypotheses (H^1,H^2 and H^3, see the subsections that follow) and confronted the results from the hypotheses testing with the results from the survey, see Table 5.3.

Table 5.3: Survey results - the influence of decision characteristics on the decision lead-time, research question RQ1 and survey question 8 (Wnuk, 2012).

	This makes the time to decide shorter	No influence	This makes the time to decide longer	Rating average/-Median
How a high number of product affects the decision lead–time, H^1	9.3%	9.3%	**81.4%**	0.72 / 1
The decision is late in the release cycle (high release number), H^2	**53.5%**	30.2%	16.3%	-0.37 / -1
The decision is filled by an important external customer, H^3	**62.8%**	23.3%	14.0%	-0.49 / -1

5.3.1 The impact of the number of products that a decision effects on the decision lead–time: H^1

Based on Hogarth et al who stated that the time needed to take a decision is highly dependent on the task complexity (Hogarth, 1975), we suspect a relationship between the number of products affected by a decision, e.g. the decision complexity, and the decision lead–time. The hypothesis testing this relationship (H^1, see Table 5.4) can be stated as:

> H^1_0: The correlation between the number of products affected by a decision and the lead–time needed to take the decision is 0.
>
> H^1_1: The correlation between the number of products affected by a decision and the lead–time needed to take the decision is **not** 0.

We used the non–parametric Spearman's Rank-Order Correlation Coefficient (Spearman, 1904) to assess the correlation size between a variable of ratio level and of ordinal level of measurement. We found $\rho(752) = .222, p < .05$ after performing the test, which is higher than the listed critical value of .197 at a two–tailed level of significance of .05. This means we can reject hypothesis H^1_0 and accept the hypothesis H^1_1 that the correlation between the number of affected products and the lead–time is not 0. Stated more general: when the number of products affected by a decision increases, the lead–time needed to take the decision increases as well. Since the correlation coefficient is rather low, the number of products may not be the only variable influencing the lead–time.

Table 5.4: The results of the hypotheses testing on the data from the decision log together with the median score from the answers from the survey (last column)

H^x	Case Study Results	Level of Significance	Median from the survey
H^1	Significant	$\rho = .222 > .197$	1
H^2	**Not** significant	$\rho = .180 < .197$	-1
H^3	**Not** significant	$p = .558 > .05$	-1
H^4	Significant	$Z = .545 > .440$	-1
H^5	Significant	$Z = 2.566 > .440$	-1
H^6	Significant	$\chi^2 = 7.032 > 2.710$	1
H^7	Significant	$t(752) = 3.940, p = 0.01$	0

Looking deeper, Figure 5.4 shows an increase of the average lead–time related to the number of products affecting change requests for the case company dataset. If we compare the average lead–time for 1 product with the highest lead–time (for 7 products), the lead-time becomes about five times longer. If we look at the lead–time for 1 product (least number) and 13 products (highest number), we can still see an increase of 130% in average lead–time. There appears to be no clear function to predict the time needed to take a decision when the number of products is known, but there is a clear positive trend to be seen. Therefore, the rise is substantial.

The results of the survey show a positive relationship between the decision lead–time and the number of products affected by the decision, see second row in Table 5.3. 81.4% of the respondents confirmed that a high number of products affected by the decision make the decision lead–time longer. This result confirms the value of the median in the second row of Table 5.4.

The concordance between the results from the decision log analysis and the survey could be interpreted as an indication that more complex investigations take more time, which confirms the experiments reported by Hogarth (1975) on requirements engineering decision making. Further, our results in relation to this factor complement our previous findings (Wnuk et al, 2011) that for large projects change proposals investigations take more time than for smaller projects. Finally, the possible practical conclusion from these results could be that if decisions have to be made quickly, their complexity should be reduced, e.g. by splitting one bigger errand into two or using other heuristics to reduce the complexity (Garcia-Retamero and Hoffrage, 2006).

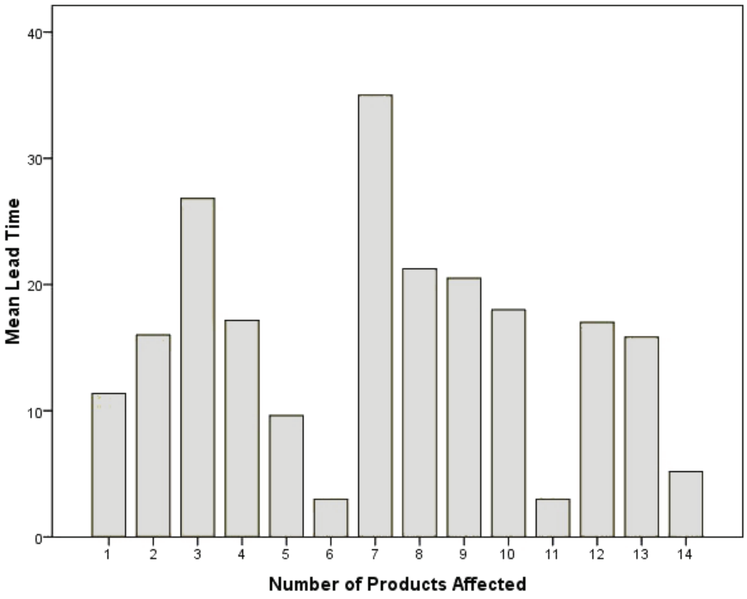


Figure 5.4: Mean lead–time per number of products affected

5.3.2 Effect of a certain release number on the decision lead–time: H^2

To study the relationship between the release number of the product line platform attribute of the change requests and the decision lead–time, we have tested the following hypothesis (H^2, see Table 5.4):

> H^2_0: The correlation between the release number of the product line platform attribute of the change requests and the lead–time needed to take the decision is 0.
>
> H^2_1: The correlation between the release number of the product line platform attribute of the change requests and the lead–time needed to take the decision is **not** 0.

We used Spearman's Rank-Order Correlation Coefficient to test the correlation between a variable of ordinal level (release number of the product line platform) and a variable of ratio level (lead–time). The result of this test is $\rho(752) = .180, p < .05$, what is below the critical value of $\rho = .197$ for an $\alpha = .05$ two–tailed level of significance (see Table 5.4). This means we can't reject H_0 and we can't state that there is a statistically significant correlation between the product line platform release number that changes impact and the lead–time needed to take a decision on our dataset.

The results of the survey regarding this aspect, see the third row in Table 5.3, show that 53.5% of the respondents suggested that decisions made late in the release cycle have a shorter lead–time. On the other hand, 30.2% of the respondents indicated that this factor has no influence on the decision lead–time and 16.3% of the respondents indicated that this factors makes the time to decide longer. To summarize, the results from the survey seems to contradict with the results from the decision log statistical analysis.

One possible interpretation of the discrepancy between the results from the survey and statistical analysis of the decision log may be related to the case company context factor. The lack of statistically significant relationship should be interpreted in the light of the result regarding hypothesis H^1. Since more complex decisions have longer decision lead-times, see hypothesis H^1, this would suggest that decisions affecting late product line platform releases at the case company have limited complexity. The process used by the case company seems to confirm this assumption as the early (major) releases are providing the main functionality of the product line platform and thus more complex decisions should be made for these early releases, see Section 3.

At the same time, the above assumption about the dominance of less complex decisions that affect late product line platform releases could also be interpreted as valid for the survey results. The demographics of the survey respondents, see Table 5.2.1, suggest that the decisions investigated by our survey respondents are less complex than decisions investigated in the case company. This, in turn, may suggest that the lead-time for later software product line releases decreases. Another possible factor affecting the survey results may be the type software projects that the majority of the survey respondents are involved in. In bespoke software projects the scope of the project is often set or implied as a contract and only minor adaptations or changes are allowed (Regnell and Brinkkemper, 2005). Thus, the decision lead-time may decrease even for later software product line releases.

5.3.3 Effect of Important Customers on the decision lead–time: H^3

To test the effect of the type of customer that issues a request on the decision lead–time, we categorized the decisions in the decision log into two categories. The first category are decisions that are requested from somewhere within the company (1003 decisions, 69,7%, were categorized into this category), while the second category are decisions that are requested by important extrenal customers of the case company (436 decisions, see also Section 4.2.1). The following hypothesis was formulated in this case (H^3, see Table 5.4):

> H^3_0: The average lead–time needed to take a decision is **not** different when an important customer issues a request.
>
> H^3_1: The average lead–time to take a decision is different when an

important customer issues a request.

The t–test (Wohlin et al, 2000) results ($t(752) = .586, p = .558$, see the sixth row in Table 5.4) does not allow us to reject H_0^5. Therefore, we can state that based on our data there is no significant difference between the lead–time needed to take decision when the decision is requested by an important customer.

One possible explanation could be the fact the all decision follow the same decision process at the case company so it doesn't matter which customer issued a change request. Another possible explanation may be that 31.3% of the analyzed requests were issued by important external customers which may have influenced the results. Finally, another possible intepretation of this result may be that the case company does not pay enough attention to the requests of important customers and thus their lead-time is not shorter. If that is the case, introducing prioritization of change requests may be a possible workaround.

The result of the survey (see Table 5.3) shows a negative relationship (requests issued by important customers have shorter lead-times), in contrast to the statistical analysis indicating no relationship, see fourth row in Table 5.4. Moreover, 62.8% of the respondents reported that time to make the decision is shorter when the decision is filled in by an important customer while 23.3% of the respondents reported that this factor has no influence on decision lead–time.

The discrepancy between the results from the decision log analysis and the survey needs further investigation. In related work, Taylor et al (2011) reported that the prioritization process is often favoring requirements from large customers and that this "greedy heuristic" produce good results when the customer base is small. At the same time, their preliminary results suggest no biases towards larger customers (Taylor et al, 2011), which confirms our results also conducted in a large–scale setting. However, the study by Taylor focused on the decision outcome rather than the decision lead–time. The possible summary conclusion from the results could be that, for smaller projects, the decision lead–time could be impacted by the type (size) of the customers issuing the requirements, while for larger contexts this relationship doesn't hold.

5.4 Factors that affect the decision outcome: RQ2

In order to investigate which decision characteristics have a significant impact on the decision outcome, we have tested three hypotheses, H^4,H^5 and H^6, see the subsections that follow, and confronted the results from the hypotheses testing with the results from the survey, see Tables 5.4 and 5.5.

Table 5.5: Survey results - the influence of decision characteristics on the decision outcome, research question RQ2 and survey question 9 (Wnuk, 2012)

	This increase the probability of rejection	No influence	This decrease the probability of rejection	Rating average / Median
There is a high number of products affected by the decision, H^4	**54.8%**	33.3%	11.9%	-0.43 / -1
) The decision is late in the release cycle, H^5	**71.4%**	26.2%	2.4%	-0.69 / -1
The decision is filled by an important customer, H^6	9.5 %	7.1%	**83.3%** (35%)	0.74 / -1
The decision took a long time to make, H^7	26.2%	**57.1%**	16.7%	-0.10 / -1

5.4.1 The impact of the number of products that a decision affects on the decision outcome: H^4

To test the relationship between the decision outcome and the number of products affected by the decision (referred as H^4 in Table 5.4), we have formed the following hypothesis:

> H^4_0: The number of products affected by a decision is **not** different for the different decision outcomes.
>
> H^4_1: The number of products affected by a decision is different for the different decision outcomes.

We used the Kolmogorov-Smirnov test for two independent samples (Smirnov, 1939) to test the relationship between an ordinal level variable and a nominal level variable. We found a result of $Z = .545, p < 0.01$, which is higher than the reported critical value listed for Kolmogorov-Smirnov's Z at this level of significance. This means we can reject H^4_0 and accept our alternative hypothesis. Thus, we can conclude that there is a high likelihood the two groups are derived from different populations. More precisely, we can say that the data indicates that rejected decisions have a lower number of products they affect.

A significant relationship was also discovered between the number of products affected by a decision and the decision lead–time, see Section 5.3.1.

Thus we can state with a high certainty that there is a relationship between the decision complexity, the decision outcome and time needed to take a decision in the case company, as suggested in the literature (Hogarth, 1975). Our results complement related research (Saliu and Ruhe, 2005; Bagnall et al, 2001) that also suggested a possible relationship between release planning and decision quality.

The survey results, see the first row in Table 5.5, disprove the statistical analysis of the decision log since 54.8% of the respondents answered that a high number of products affected by the decision increases the probability of rejection. This contradicting result could be caused by the fact that in the case study dataset more rejected than accepted decisions affected only one product. In other words, the case company seems to be more eager to reject than accept small change requests. This may have economical basis if we assume that change requests affecting only one product weakly contribute to revenue generation. In this case, it appears to be more logical to reject those change requests and focus on more complex change requests are potentialy more promissing additional revenue contributors.

Another possible explanation for the conflicting results between the decision log analysis and the survey could be the fact that the majority of the survey respondents (68%) worked with companies up to 100 persons involved in a project and a typical project with not more than around 100 requests. This may suggest that the complexity, understood as the number of products involved in the decision, does not influence the rejection of issued requests for larger projects, but it could for smaller projects. Also, with a low number of around 100 requests per a typical (bespoke) project, project management may need to focus on investigating and possibly accepting all change requests from the customers.

Since most of the survey respondents worked with software product line approach (64%) hence some respondents admitted to work with bespoke and off–the–shelf software, this may suggest that in those contexts complex investigations are more likely to be rejected than accepted. However, this assumption needs to be further investigated for significance. In related work, Wnuk et al (2009) reported five main reasons for excluding a feature candidate from the scope of the project but not analyzed the complexity of these feature candidates. Our results suggest that the complexity is an additional factor that should be further investigated.

5.4.2 Effects of a certain release number on the decision outcome: H^5

As a second relationship investigated for RQ2, we tested if the product line platform release number attribute impacts the decision outcome. We stated the following hypothesis (referred as H^5 in Table 5.4) and confronted the results with the results from the survey, see the third row in Table 5.5:

> H^5_0: The release number a decision affects is **not** different for the different decision outcomes.

> H_1^5: The release number a decision affects is different for the different decision outcomes.

We used the Kolmogorov-Smirnov test for two independent samples, which resulted in a score of $Z = 2.566, p < 0.01$ (see Table 5.4). This result is above the documented critical value of Kolmogorov-Smirnov's Z, what means we can reject H_0^5 and accept the alternative hypothesis H_1^5. Thus, we can state that the changes of accepting a request are higher if that request affects a release late in the release cycle.

The results from the survey show an opposite relation, see the third row in Table 5.5. 71.4% of the respondents indicated that requests affecting products with higher release numbers (planned to be released late in the release cycle) are more likely to be rejected. We suspect this contrast in results between the survey and the case study could be caused by the fact that the case company is simply getting more requests for late releases (65.5% of all requests). Another possible explanation may be that customers could use the products released in the beginning of the release cycle as a potential source of requests for future releases. This could explain the contrasting results between the decision log analysis and the survey. Moreover, since late platform releases are focusing on smaller adaptations of the platform, this may also impact the results.

The fact that the respondents mainly worked with smaller projects than investigated at the case company could also be the cause of the discrepancy of the statistical analysis and survey results. To summarize, the results from the case study and from the survey suggest that the release that decisions concern could be an additional factor that influences decision outcomes and this result complements published related work (Wohlin and Aurum, 2005; Wnuk et al, 2009; Barney et al, 2008; Ruhe and Saliu, 2005). However, the discrepancy between the case study and the survey results suggest that the direction of the relationship may not always be the same.

5.4.3 Effect of Important Customers on the decision outcome: H^6

For the last factor that could affect decision outcome, we have tested if there was any effect on the decision outcome caused by the type of the customer that issues a change request. In order to test this relationship, we performed a χ^2 test for $r * c$ tables.

Our hypothesis (H^6 in Table 5.4) is:

> H_0^6: The frequencies in the contingency table between the decision outcome and involvement of important customers do **not** differ from the normal expected frequencies.

> H_1^6: The frequencies in the contingency table between the decision outcome and involvement of important customers differ from the normal expected frequencies.

The result of this test is with $\chi^2(1, N = 1439) = 7.032, p < .01$ above the listed critical value. This means we can reject H_0^6 and accept our alternative hypothesis. Since the value of χ^2 is rather low, we can state that the change receives a positive decision outcome when it is requested by an important customer. Looking deeper, we identified that 11% more decisions originate from external customers than internal customers.

The majority of the survey respondents (83.3%, see row 3 in Table 5.5) indicated that the importance of the customer that issues the request decreases the probability of rejection, in other words increases the probability of acceptance. Since the value of χ^2 test above is rather low this may indicate that the fact that requests from important external customers were more likely accepted at the case company could be a company specific phenomenon, or related to the project size as the survey respondents most likely worked with projects with fewer than 100 requests.

In a related study, Taylor et al (2011) suggested that larger customers more likely get their requirements accepted, but the paper lacks statistical analysis of the mentioned correlation. Moreover, our results from the statistical analysis regarding the influence of the importance of the customer on the decision lead–time H^2 and the decision outcome H^5 are not consistent, which could suggest additional uncovered factors. Ruhe and Saliu (2005) suggested that the release decisions are made by "contracting the main stakeholders and manually balancing their interests and preferences" which we interpret as accepting more features from important (main) stakeholders. Finally, our results confirm the viewpoint of Bagnall et al (2001), who suggested that requirements from "favored customers" will be viewed as more important than other requirement and thus those requirements will be more often accepted.

5.5 Effect of lead–time on the decision outcome - RQ3

The last relationship we examined is whether the lead–time influences the decision outcome. To test this relation, we stated the following hypothesis (stated as H^7 in Table 5.4):

> H_0^7: The average lead–time needed to make a decision does **not** differ per decision outcome.
>
> H_1^7: The average lead–time to make a decision does differ per decision outcome.

After categorizing decisions to accepted and rejected decisions, we calculated their average lead–times. The average lead–time for accepted and rejected decisions is respectively $\mu = 1.12$ and $\mu = .98$. The t–test result ($t(752) = 3.940, p < 0.01$, see the last row in Table 5.4) indicated a significant differences between the average lead–time for both decision outcomes. This means we can accept H_1^7 and reject the null–hypothesis H_0^7.

Based on these results, we can state that the average lead–time needed to reject a decision is statistically significantly longer than the lead–time needed to accept a decision.

When looking at the survey results presented in the last row in Table 5.5, we see that 57.1% of the respondents indicated that the time to make the decision does not influence the decision outcome. The statistical analysis of the survey results for this question showed a neutral relationship (median equals to 0, see last row in Table 5.4) which prevents us from drawing stong conclusions. However, it is worth noticing that 26.2% of the respondents agreed with the statistically significant result of the decision log analysis.

There could be several possible causes of the discrepancy between the decision log analysis results and the survey results in regards to this aspect. One possible explanation could be the size of the projects analyzed in the case study and by the survey respondents. Since the questionnaire respondents mainly worked with projects that generate not more than 100 requests and with smaller companies, we suspect that the complexity of the issued changes in those contexts is smaller than in the case of the case company investigated. As a results, those assumingly less complex decisions could be proceeded faster by our questionnaire respondents than by the practitioners from the case company, as suggested by Hogarth (1975). Thus, the survey respondents might have not been able to experience as long decision lead–times as the case company practitioners and thus for them this factors does not influence the decision outcome.

Moreover, since the case company operates in the MDRE context, the time pressure to investigate and decide upon incoming requirements is high. Excessive deposition of decisions may cause serious consequences for the success of software projects in the MDRE context as time–to–market is critical (Regnell and Brinkkemper, 2005). For long investigations, decision makers could simply be forced to reject the proposal due to a missed market–window opportunity and this could be one of the possible explanation of the statistically significant result. This interpretation could be supported by the fact that more than 1/4 of the survey respondents worked with bespoke software projects. Furthermore, as visualized by Wnuk et al (2009), accepting new features to the project scope is much easier than reducing the scope which is often performed during the entire time of the project.

6 Conclusions

Although RE decision making has been studies in a number of publications (Alenljung and Persson, 2008; Aurum and Wohlin, 2003; Evans et al, 1997; Ngo-The and Ruhe, 2005; Fogelstrom et al, 2009; Berander and Andrews, 2005; Wohlin and Aurum, 2005; Barney et al, 2008; Rolland et al, 1995), little empirical evidence exists that explicitly and exhaustively investigates the relationships between the change requests attribute and the decision lead–times and outcomes.

In this paper, we report on an investigation of decision making in requirements engineering. We analyzed 1439 change requests looking for statistically significant relationships between the decision making factors i.e., number of products, release number, type of customer and decision lead–times and outcomes. The results from this analysis were confronted with the results from a survey among 50 practitioners from several countries involved in decision making processes. The results from the study could be summarized in the following points:

- The lead–time to make a decision increases when more products (considered as a proxy for the decision complexity) are affected by this decision - this result was confirmed both in the statistical analysis and in the survey. Since the relationship is rather clear, decision makers should be aware that too complex decisions may take a long time (hypothesis H^1).

- The statistical analysis showed that if a request affects a lot of products, it has a higher change of being accepted (hypothesis H^4). The respondents of the survey stated that requests that affect a lot of products have a higher change of being rejected. This may seem counter-intuitive, but this is probable caused by the fact that request that affect a lot of products are often requests related to the platform and thus are important.

- There is no significant relationship between the release of the product line that a change request impacts and the decision lead–time according to the results from the statistical analysis of the decision log. At the same time, the majority of the respondents in the survey suggested that decisions made late in the release cycle have shorter lead–times (hypothesis H^2).

- Change requests affecting late releases have a significantly higher probability of acceptance according to the statistical analysis of the decision log (hypothesis H^5). This result seems to be more characteristic for large contexts as the results from the survey, in which most respondents worked with projects with fewer than 100 decisions, indicate the opposite relationship with a higher probability of rejecting these requests.

- The lead–time for decisions is shorter when the change requests are issued by important customers, according to the respondents (hypothesis H^3). The statistical analysis of the decision log disproved this suggestion. Therefore, no clear relationship was identified for this factor.

- Change requests issued by important customers are more likely to be accepted, (hypothesis H^6) according to the statistical analysis of the decision log. This relationship was confirmed by a clear majority of survey respondents (83.3%).
- The lead–time to reject a decision is significantly longer than to accept a decision (research question RQ3), according to the statistical analysis of the decision log. At the same time, the results from the survey suggests that there is no relationship between the lead–time and the decision outcome.

Our results clearly indicate that the number of products affected by a decision increases the decision lead–time (research question RQ1). This result has a practical importance for requirements engineering decision makers. As more complex decision take more time, it may be wise to decrease their complexity for faster decisions. This could be particularly useful in MDRE where time to market pressure is inevitable (Regnell and Brinkkemper, 2005). Our study reports that lead–times could become up to 400% longer if a complex decision affects multiple products.

Our results also confirm that the importance of the customer who issues a decision log increases the probability of acceptance (research question RQ2). These requests have an 11% higher change to be accepted than other requests. Product management processes could be adapted when being aware of the supported relationships. For example, the change process of Figure 5.1 can be refined by asking for additional details from more important customers in order to reduce the lead–time.

Regarding the relationship between the decision lead–time and the decision outcome (research question RQ3), we report based on the analysis of the deicision log that the average lead–time needed to reject a decision is statistically significantly longer than the lead–time needed to accept a decision. This result couldn't been confirmed by the survey respondents. Decision makers could use this conclusion when planning for effective pruning of possible decisions for a project. At the same time, this relationship seems to hold for larger projects, as the results from the survey suggests that there is no relationship between the lead–time and the decision outcome.

Related to the differences observed between the statistical analysis results and the survey results, we report that there are premises that less complex decisions are more often rejected in large projects but not in smaller projects. Moreover, for smaller project the decisions affecting products planned to be released late in the release cycle are more likely to be rejected than for larger projects. At the same time, the majority of the survey respondent reported that time to make a decision is shorter when this decision is filled by an important customer, while for the large case company this relationship doesn't seem to hold.

Future research is planned to go more in depth on the possible relationships among requirements engineering decision making characteristics. Two relationships could be proven and quantified by us, but the other five relationships need further research in order to further explore them. Within the two relationships proven by us, more research is needed as well. For instance, it would be helpful and desirable if a function could be formulated to estimate the lead–time or the chance on a certain decision outcome.

Finally, other decision characteristics, such as the number of stakeholders involved of the number of dependencies between software components, could also be of relevance for the decision lead–time or outcome. Due to lack of data, these characteristics have not yet been taken into account in this research, but could be considered in the scope for future research.

Acknowledgment

The authors would like to thank Thomas Olsson and Prof. Per Runeson for their suggestions in this research project. The authors would like to thank Dr. David Callele for excellent language comments. We also thank the survey respondents. This work was supported by VINNOVA within the UPITER project.

APPENDIX: QUESTIONNAIRE QUESTIONS

INTRODUTION

This is a short survey about decision making in requirements engineering. When managing requirements, often the decision has to be made whether or not to accept a certain requirement request. These possible requirements all have different characteristics such as the number of products they affect or the fact that they are requested by an important customer.

The purpose of this survey is to asses which characteristics of submitted requirements change requests may influence the decision outcome and the decision lead-time. We already performed a quantitative analysis on the decision logs of a large software products company and we want to validate our results using experiences from other companies.

After your participation in the survey we will get back to you with the analyzed results of the survey and you will also get access to the results of the quantitative analysis of decision logs. Thanks in advance for your cooperation!

BACKGROUND

In order to compare your answers with our quantitative analysis results, we need to know some things about you, your company and your project context.

Question 1: What region are you most active in?
( ) The Netherlands
( ) Belgium
( ) Germany
( ) Sweden
( ) Worldwide
( ) Other (please specify)

Question 2: What is your current role within the company?
( ) Project Manager
( ) Product Manager
( ) Quality Expert
( ) Developer
( ) Senior Management
( ) Consultant
( ) Other (please specify)

Question 3: How many years of professional experience do you have in software engineering?

Question 4: How many people are involved in the software engineering process in your company? Please consider all employees, including, but not limited to developers, testers and management.
( ) < 10
( ) 10 - 100
( ) 101 - 500
( ) 501 - 1000
( ) > 1000

Question 5: What kind of relationship does your company have with its customers?
( ) We create mostly custom bespoke software
( ) We create mostly off-the-shelf software
( ) Other (please specify)

Question 6: How many requirement requests does a project in your company have on average?
( ) Around 10
( ) Around 100
( ) Around 1.000
( ) Around 10.000
( ) Other (please specify)

Question 7: Does your company apply a software product line approach? (Does your company release a collection of similar software products from a shared set of software assets?)
( ) Yes
( ) No
( ) Other (please specify)

RATINGS

Please answer the questions below according to your own experiences. Please indicate how the following decision characteristics influence the time needed to take the decision.

Question 8. Please indicate how the following decision characteristics influence the time needed to take the decision

Question 9. Please indicate how the following decision characteristics influence the decision outcome.

Table 5.6: Question 8. Please indicate how the following decision characteristics influence the time needed to take the decision.

	This makes the time to decide shorter	No influence	This makes the time to decide longer
There are a high number of products affected by the decision	( )	( )	( )
The decision is late in the release cycle	( )	( )	( )
The decision is filed by an important customer	( )	( )	( )

Table 5.7: Question 9. Please indicate how the following decision characteristics influence the decision outcome.

	This increase the probability of rejection	No influence	This decrease the probability of rejection
There are a high number of products affected by the decision	( )	( )	( )
The decision is late in the release cycle	( )	( )	( )
The decision is filed by an important customer	( )	( )	( )
The decision took a long time to make	( )	( )	( )

Bibliography

Alenljung B, Persson A (2008) Portraying the practice of decision-making in requirements engineering: a case of large scale bespoke development. Requirements Engineering 13(4):257–279

Ashrafi N (1998) Decision making framework for software total quality management. International Journal of Technology Management 16(4-6):532 – 543

Aurum A, Wohlin C (2003) The fundamental nature of requirements engineering activities as a decision-making process. Information and Software Technology 45(14):945–954

Aurum A, Wohlin C (2005) Engineering and managing software requirements. Springer Verlag

Bagnall A, Rayward-Smith V, Whittley I (2001) The next release problem. Information and Software Technology 43(14):883 – 890

Barney S, Aurum A, Wohlin C (2008) A product management challenge: Creating software product value through requirements selection. Journal of Systems Architecture - Embedded Systems Design 54(6):576–593

Berander P, Andrews A (2005) Requirements prioritization. In: Aurum A, Wohlin C (eds) Engineering and Managing Software Requirements, Springer Berlin Heidelberg, pp 69–94

Cleland-Huang J, Settimi R, BenKhadra O, Berezhanskaya E, Christina S (2005) Goal-centric traceability for managing non-functional requirements. In: Proceedings of the 27th international conference on Software engineering, ACM, New York, NY, USA, ICSE '05, pp 362–371

Cooper R (1990) Stage-gate systems: a new tool for managing new products. Business Horizons 33(3):44–54

D'Agostino R, Pearson E (1973) Tests for departure from normality. empirical results for the distributions of $b^2 and \sqrt{b}^1$. Biometrika 60(3):613

DeCarlo L (1997) On the meaning and use of kurtosis. Psychological Methods 2(3):292–307

DeGregorio G (1999) Enterprise-wide requirements and decision management. Proceedings of 9th International Symposium of the International Council on System Engineering

Easterbrook S, Singer J, Storey MA, Damian D (2008) Selecting empirical methods for software engineering research. In: Shull F, Singer J, Sjøberg D (eds) Guide to Advanced Empirical Software Engineering, Springer London, pp 285–311

Egyed A, Wile D (2006) Support for Managing Design-Time Decisions. IEEE Transactions on Software Engineering 32(5):299–314

Evans R, Park S, Alberts H (1997) Decisions not requirements decision-centered engineering of computer-based systems. In: The 1997 IEEE Conference and Workshop on Engineering of Computer-Based Systems, pp 435–442

Field A (2009) Discovering statistics using SPSS. Sage Publications Ltd

Fogelstrom N, Barney S, Aurum A, Hederstierna A (2009) When product managers gamble with requirements: attitudes to value and risk. Berlin, Germany, pp 1 – 15

Garcia-Retamero R, Hoffrage U (2006) How causal knowledge simplifies decision-making. Minds and Machines 16(3):365 – 80

Hallowell R (1996) The relationships of customer satisfaction, customer loyalty, and profitability: an empirical study. International Journal of Service Industry Management 7(4):27–42

Hogarth R (1975) Decision time as a function of task complexity. Utility, probability, and human decision making: selected proceedings of an interdisciplinary research conference, Rome, 3-6 September, 1973 pp 321–338

Jacoby J, Matell M (1971) Three-point Likert scales are good enough. Journal of Marketing Research 8(4):495–500

Jonsson P, Lindvall M (2005) Impact analysis. In: Aurum A, Wohlin C (eds) Engineering and Managing Software Requirements, Springer Berlin Heidelberg, pp 117–142

Kabbedijk J, Wnuk K, Regnell B, Brinkkemper S (2010) What decision characteristics influence decision making in market-driven large-scale software product line development? In: Proceedings of the Product Line Requirements Engineering and Quality Workshop at the 16th REFSQ Conference, Duisburg, Germany, p 42–53

Karlsson J, Ryan K (1997) A cost-value approach for prioritizing requirements. IEEE Software 14(5):67–74

Karlsson L, Dahlstedt A, Regnell B, Natt och Dag J, Persson A (2007a) Requirements engineering challenges in market-driven software development–An interview study with practitioners. Information and Software technology 49(6):588–604

Karlsson L, T T, Regnell B, Berander P, Wohlin C (2007b) Pair-wise comparisons versus planning game partitioning-experiments on requirements prioritisation techniques. Empirical Software Engineering 13(3):3–33

Khatri N (2000) The Role of Intuition in Strategic Decision Making. Human Relations 53(1):57–86

Kitchenham B, Pfleeger S (2002) Principles of survey research part 2: designing a survey. ACM SIGSOFT Software Engineering Notes 27(1):18–20

Kitchenham BA, Travassos GH, von Mayrhauser A, Niessink F, Schneidewind NF, Singer J, Takada S, Vehvilainen R, Yang H (1999) Towards an ontology of software maintenance. Journal of Software Maintenance 11(6):365–389

Klein G, Orasanu J, Calderwood R, Zsambok C (eds) (1995) Decision making in action: Models and methods. New Jersey: Norwood

Leffingwell D, Widrig D (2003) Managing Software Requirements: A Use Case Approach, 2nd edn. Pearson Education

Lethbridge TC, Sim SE, Singer J (2005) Studying software engineers: Data collection techniques for software field studies. Empirical Softw Engg 10:311–341

Li C, van den Akker M, Brinkkemper S, Diepen G (2010) An integrated approach for requirement selection and scheduling in software release planning. Requirements Engineering 15(4):375 – 96

Messerschmitt D (2004) Marketplace issues in software planning and design. IEEE Software 21(3):62–70

Natt och Dag J, Regnell B, Carlshamre P, Andersson M, Karlsson J (2001) Evaluating automated support for requirements similarity analysis in market-driven development. Proceeding of the 7th International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ'01)

Natt och Dag J, Regnell B, V Gervasi SB (2005) A Linguistic-Engineering Approach to Large-Scale Requirements Management. IEEE Software 3:32–39

Ngo-The A, Ruhe G (2005) Decision support in requirements engineering. In: Aurum A, Wohlin C (eds) Engineering and Managing Software Requirements, Springer Berlin Heidelberg, pp 267–286

Ott L, Longnecker M (2008) An introduction to statistical methods and data analysis. Duxbury Pr

Paech B, Koenig T, Borner L, Aurum A (2005) An Analysis of Empirical Requirements Engineering Survey Data. Engineering and Managing Software Requirements pp 427–452

Pohl K, Böckle G, Van Der Linden F (2005) Software Product Line Engineering: Foundations, Principles, and Techniques. Springer-Verlag New York Inc

Regnell B, Brinkkemper S (2005) Market-driven requirements engineering for software products. Engineering and managing software requirements pp 287–308

Regnell B, Kuchcinski K (2011) Exploring software product management decision problems with constraint solving - opportunities for prioritization and release planning. In: Proceedings of the Fifth International Workshop on Software Product Management (IWSPM 2011), pp 47 –56

Rolland C, Souveyet C, Moreno M (1995) An approach for defining ways-of-working

Ruhe G (2009) Product Release Planning: Methods, Tools and Applications. Auerbach Publications

Ruhe G, Saliu M (2005) The Art and Science of Software Release Planning. IEEE Software 22(6):47–53

Runeson P, Höst M (2009) Guidelines for conducting and reporting case study research in software engineering. Empirical Software Engineering 14(2):131–164

Saliu O, Ruhe G (2005) Supporting software release planning decisions for evolving systems. Proceedings of the 29th Annual IEEE/NASA Software Engineering Workshop pp 14–26

Sheskin D (2004) Handbook of parametric and nonparametric statistical procedures. Chapman and Hall

Smirnov N (1939) On the estimation of the discrepancy between empirical curves of distribution for two independent samples. Bulletin Mathematics Univiversity Moscow 2:3–14

Spearman C (1904) General intelligence: Objectively determined and measured. The American Journal of Psychology 15(2):201–292

Stevens S (1946) On the theory of scales of measurement. Science 103(2684):677–680

Strigini L (1996) Limiting the Dangers of Intuitive Decision Making. IEEE Software 13(1):101–103

Taylor C, Miransky A, Madhavji N (2011) Request-implementation ratio as an indicator for requirements prioritisation imbalance. Trento, Italy, pp 3 – 6 van den Akker M, Brinkkemper S, Diepen G, Versendaal J (2008) Software product release planning through optimization and what-if analysis. Information and Software Technology 50(1-2):101 – 11

Wild C, Maly K, Zhang C, Roberts C, Rosca D, Taylor T (1994) Software engineering life cycle support-decision based systems development. New York, NY, USA, vol vol.2, pp 781 – 4

Wnuk K (2012) The survey questions can be accessed at. `http://fileadmin.cs.lth.se/cs/Personal/Krzysztof_Wnuk/SQJ2012/SurveyQuestions.pdf`

Wnuk K, Regnell B, Karlsson L (2009) What happened to our features? visualization and understanding of scope change dynamics in a large-scale industrial setting. In: Proceedings of the 17th IEEE International Requirements Engineering Conference (RE 2009), pp 89–98

Wnuk K, Regnell B, Berenbach B (2011) Scaling up requirements engineering - exploring the challenges of increasing size and complexity in market-driven software development. In: Proceedings of the 17th international working conference on Requirements engineering: foundation for software quality, Springer-Verlag, Berlin, Heidelberg, REFSQ'11, pp 54–59

Wohlin C, Aurum A (2005) What is important when deciding to include a software requirements in a project or release? In: Proceedings of the International Symposium on Empirical Software Engineering (ISESE 2005), pp 246–255

Wohlin C, Höst M, Runeson P, Ohlsson M, Regnell B, Wesslén A (2000) Experimentation in software engineering: an introduction. Kluwer Academic Pub

Yin R (2008) Case study research: Design and methods. Sage Publications, Inc

Zur H, Breznitz S (1981) The effect of time pressure on risky choice behavior. Acta Psychologica 47(2):89–104

Paper VI

Scope Tracking and Visualization for Very Large-Scale Requirements Engineering

Krzysztof Wnuk[1], David Callele[2], Tony Gorschek[3], Even-André Karlsson[4] and Björn Regnell[1]
[1]Department of Computer Science,
Lund University, Sweden
`{Krzysztof.Wnuk,Bjorn.Regnell}@cs.lth.se`
[2] TRLabs, Saskatoon, Saskatchewan, Canada ,
`dcallele@trlabs.ca`
[3]School of Computing Software Engineering Research Lab,
Blekinge Institute of Technology,
SE-371 79 Karlskrona, Sweden
`Tony.Gorschek@bth.se`
[4]Add a Lot, Sweden,
`even-andre.karlsson@addalot.se`

submitted to the
Transactions on Software Engineering journal 2012

Abstract

Requirements scope management is an important part of software engineering. Deciding the optimal scope that fulfills the needs of the most important stakeholders is challenging due to a plethora of aspects may impact decisions. In rapidly changing software markets, frequently changing needs of stakeholders force decision makers to continuously adjust the scope of their projects. How to support decision makers facing many changes occurring in large software projects? This paper presents a visual technique designed to give a quick and effective overview over requirements scoping process for large and very-large projects. The technique provides multiple-views and it can visualize various information based on given filtering criteria. Furthermore, the paper analyzes the decision making patterns based on a large dataset from industry. The results are evaluated during interviews with practitioners.

1 Introduction

The probability of project success is directly correlated with the quality of the scoping exercise; for some, the scope of a software project defines its success or failure (Iacovou and Dexter, 2004). Overscoping, scope-creep and requirements scrap are other aspects of scoping that may have negative impacts upon software projects (Bjarnason et al, 2012). Optimizing and maintaining scope while addressing the changing needs of key customers are critical for successful development projects.

Large software projects operating in a Market-Driven Requirements Engineering (MDRE) context (Regnell and Brinkkemper, 2005) experience many scope changes during both requirements and development phases (Wnuk et al, 2009). These changes need to be analyzed, decisions made, actions taken and documentation maintained. When the number of changes is large, and the complexity of the impact analysis is significant, understanding the consequences of changes is challenging. Current requirements tools offer only partial solutions to this challenge.

In this paper, we present a technique for visualizing scope changes in large software projects, successfully applied to datasets from projects with thousands of features. The implementation supports visualizing large feature sets, as well as sorting, filtering and interactive zooming. The technique was evaluated on a very large industry dataset then evaluated by and refined with the assistance of several industry practitioners.

This paper is structured as follows: Section 2 presents background and related work, Section 3 presents the case study company description while Section 4 presents research methodology. The visual technique is presented in Section 5 while Section 6 presents and discusses the identified decision patterns. In Section 7 we conclude the paper.

2 Background and Related Work

Market-Driven Requirements Engineering (MDRE) (Regnell and Brinkkemper, 2005) is defined as the process of managing requirements for software products that are offered to an open marked. Producing software for the open market differs from developing bespoke software in several ways. Among the most important characteristics, Regnell and Brinkkemper (2005) identified: (1) release planning focused on time-to-market and return-on-investment, (2) large volume of potential requirements and (3) consequences of RE decisions are suffered only by the company responsible for the decisions. The presence of a constant stream of incoming requirements, asynchronous to the development cycle, combined with market pressures from competitors' products, increases the difficulty and importance of effective decision making in MDRE.

Several studies reported a variety of challenges in MDRE contexts (Regnell and Brinkkemper, 2005; Karlsson et al, 2002; Lubars et al, 1993; Gorschek and Wohlin, 2006; Höst et al, 2001; Carlshamre et al, 2001). Among the reported decision making challenges in MDRE, Karlsson et al (2002) identified managing the communication gap between marketing staff and development as significant. Managing the constant flow of requirements, and the number of requirements, were also identified (Karlsson et al, 2002) as threats that could cause requirements process overloading (Höst et al, 2001; Regnell and Brinkkemper, 2005), and overscoping (Bjarnason et al, 2012). The absence of actual customers with which to negotiate requirements (Potts, 1995) and the many different forms and abstraction levels of incoming requirements (Gorschek and Wohlin, 2006) also complicate the decision making process. Given that requirements are (typically) highly interrelated (Carlshamre et al, 2001), selecting a requirement requires checking its dependencies and can even necessitate including dependent requirements. Identifying an appropriate balance between market pull and technology push (Regnell and Brinkkemper, 2005) in the context of release planning based on uncertain estimates (Karlsson and Ryan, 1997; Karlsson et al, 2002; Ruhe and Saliu, 2005) is a challenging task.

Decision making occupies a central place in requirements engineering (Aurum and Wohlin, 2003) and is considered inevitable when managing requirements (DeGregorio, 1999). The decisions in requirements engineering process can range from the organizational down to the project level (Aurum and Wohlin, 2003; Ngo-The and Ruhe, 2005) and making these decisions is considered knowledge-intensive (Klein et al, 1995). Several studies identified challenges in requirements engineering decision making including incompleteness of available information (Ngo-The and Ruhe, 2005), shifting, ill-defined or competing goals (Alenljung and Persson, 2008), finding the right balance between the commercial requirements over internal quality requirements (Karlsson and Ryan, 1997; Fogelström et al, 2009), conflicting priorities between stakeholders (Karlsson and Ryan, 1997), dependencies between requirements (Carlshamre et al, 2001), and the identification of business goals when performing release planning (Ngo-The and Ruhe, 2005). Further challenging aspects include the fact that decisions in requirements engineering are often semi-structured or unstructured (Ngo-The and Ruhe, 2005) and the fact that often decisions need to be changed or altered (Wnuk et al, 2009). Finally, as requirements decision making area is still in its initiation (Ngo-The and Ruhe, 2005), more research need to be conducted, e.g. in identifying objective criteria for guiding decision makers that could be based on past experiences (Strigini, 1996).

Requirements prioritization, selecting the most valuable requirements for implementation, is challenging for many reasons including conflicting stakeholder priorities (Karlsson and Ryan, 1997), dependencies between requirements (Carlshamre et al, 2001), different requirement abstraction levels rendering comparison difficult (Gorschek and Wohlin, 2006) and limited requirements prioritization method scalability (Lehtola and Kauppinen, 2006). Techniques such as cost-value analysis (Karlsson and Ryan, 1997), dividing requirements into three different piles (Beck and Andres, 2004) can be employed. Scoping decisions based on requirements prioritization are often changed or altered due to a number of factors such as stakeholder business decision or lack of resources (Wnuk et al, 2009). Further research into management of the decision making process, including real-time feedback on ongoing projects, is recommended (Basili and Rombach, 1988).

The release planning process makes software available to users of an evolving software product (Ruhe, 2009) in planned stages (van der Hoek et al, 1997), extending the prioritization process to include temporal aspects such as the assignment of individual requirements to specific releases. Several methods for supporting release planning were proposed, e.g. the EVOLVE method (Greer and Ruhe, 2004), EVOLVE* - balancing release time, effort and value (Ruhe and Ngo, 2004), a method based on what-if analysis (van den Akker et al, 2008), a method based on combining cost-value analysis with requirements interdependencies (Carlshamre, 2002), a method based on fixed release dates that allocates requirements to a "must" element and a "wish" element (Regnell et al, 1998). Incremental methods for release planning (Denne and Cleland-Huang, 2004) are a principal element of agile software development processes. Finally, approaches for release planning based on integer linear programming (van Den Akker et al, 2005) or constraint programming (Regnell and Kuchcinski, 2011) were also proposed.

Despite the large number of proposed methods for release planning, Svahnberg et al (2010) reported on 24 strategic release planning models presented in academic papers investigating market-driven software development, the activity remains challenging. Jantunen et al (2011) identified three main challenges of software release planning: (1) requirements prioritization, (2) resolving requirements interdependencies and (3) continuously changing problem definitions and viewpoints. The number of factors possible affecting priorities of requirements in software product companies is high (Lehtola and Kauppinen, 2006) including the volume of information that must be considered when performing release planning (Ruhe, 2009) and uncertainties about the sources and quality of this information (Ruhe, 2009). This work addresses aspects of these issues by providing visualization techniques that support the release planning process.

Requirements scoping is considered a core function in software release planning (Schmid, 2002), particularly in the context of software product lines (Schmid, 2002; Kishi et al, 2002; Savolainen et al, 2007). The software product line literature contains work on the identification aspect of scoping (Schmid, 2002; Savolainen et al, 2007) while our previous work has shown that scoping is a continuous activity, an activity where overscoping (Bjarnason et al, 2012) and scope reductions are frequent phenomena (Wnuk et al, 2009). Scoping is also prominent in the project and product management literature (Institute, 2000; Iacovou and Dexter, 2004; Legodi and Barry, 2010).

Regarding related work on scoping in project and product management, several studies focused on a phenomenon of *scope creep* defined as uncontrolled expansion of initially decided scope (Carter et al, 2001; Hall et al, 2002; DeMarco and Lister, 2003; Iacovou and Dexter, 2004; Kulk and Verhoef, 2008). Scope creep could be cased by sales stuff agreeing to deliver unrealistically large features without checking the scheduling implications first (Hall et al, 2002) or by stakeholders unable to concur on project goals (DeMarco and Lister, 2003) and have serious negative consequences on project, including project failure (Iacovou and Dexter, 2004). To mitigate negative effects of scope creep, Carter et al (2001) suggested combining evolutionary prototyping and risk-mitigation strategy. Another related phenomenon is *requirements scrap* which is defined as a situation when the scope of a project both increases but also decreases (Kulk and Verhoef, 2008). Whether increase of the project scope is simply scope creep, decrease of the scope could be a result of shrinking budgets or running out of schedule (Kulk and Verhoef, 2008). In our recent study about overscoping, defined as setting a scope that requires more resources than are available (Bjarnason et al, 2012), we identified and empirically investigated six main causes of overscoping and we confirmed that overscoping can have serious negative effects, including: many changes after the project scope is set, quality issues, wasted effort and customer expectations not always met (Bjarnason et al, 2012). The visualizations presented in this paper help to address both scope creep and scope scrap challenges by providing a robust overview over scoping decisions.

Visualizing information improves comprehension, particularly with multidimensional data sets (Tufte, 1990) such as the requirements engineering decision making tasks considered by Gotel et al (2007). Visual notation supplemented by accompanying text according to dual channel theory (Moody, 2009) have been shown to improve comprehension even further.

Given that customers of software products are often non-technical people (Avison and Fitzgerald, 1998) and that requirements engineering is a communication intensive phase of software development, it is surprising that only a small number of studies focus on visualization in requirements engineering (Gotel et al, 2007). Visualizations appear to be principally used in later phases of the development lifecycle (Gotel et al, 2007), e.g. for creating visual representations of the code (Ball and Eick, 1996), to support test data selection (Leon et al, 2000) and to summarize bug reports (D'Ambros et al, 2007). Several researchers focused on visualizing open source software development projects, proposing visual techniques for: code clone analysis (Livieri et al, 2007), project communication (Oezbek et al, 2010), software evolution (Hanakawa, 2007) and analysis of developer social networks (Jermakovics et al, 2011).

Three principle applications for visualization were identified in related work in requirements engineering (Gotel et al, 2007). The first application is to visualize structure and relationships between requirements, e.g. using graph-based visualization (Austin et al, 2006). The second application is to support requirements elicitation by visualizing soft goals (Sen and Jain, 2007) or creating a 'Rich Picture' of the system to be developed (Checkland, 1981). The third application of requirements visualization is modeling. Several visual modeling techniques were proposed, e.g. UML diagrams (UML, 2010), the i* framework (Yu, 1997) and the KAOS framework (van Lamsweerde and Letier, 2004). Gotel et al (2007) postulated the need for more techniques to visualize requirements using metaphors to improve communication. The small number of studies that focus on providing visual assistance for decision makers in requirements engineering may suggest that requirements engineering field has yet to realize the benefits that could be delivered by information visualization techniques (Gotel et al, 2007).

Regnell et al (2008) proposed a requirements engineering classification scheme based on the number of requirements and dependencies between them, claiming that most research papers seek to validate a proposed requirements engineering method or tool in small and medium scale contexts. This approach contrasts with work by Berenbach who reports a common misconception about requirements engineering is the expectation that processes that work for a small number of requirements will scale (Berenbach et al, 2009). There is evidence that scalable requirements engineering tools and methods should be implemented before the company and the number of requirements begin to grow rapidly (Berenbach et al, 2009; Wnuk et al, 2011) but the body of published work addressing large and very-large requirements engineering contexts is small.

Among the related work that reports a technique or solution suitable for large-scale software engineering contexts, Carman et al (1995) proposed a framework for software reliability while Garg (1989) presented an information management model for large automotive projects. Among the related work that reports empirical evidence from large-scale contexts, Konrad and Gall (2008) reported lessons learned from large-scale requirements engineering projects - mentioning scope change and creep as one of the main challenges. Bergman et al (2002) investigated the political nature of requirements engineering for large systems while Ebert (2004) presented a technique for pragmatically dealing with non-functional requirements in large systems. Boehm (2006) identified increasingly rapid change as one of the future challenges for software engineering processes, Herbsleb (2007) called for more research to improve management of global development efforts while Northrop et al (2006) explored challenges of Ultra-Large-Scale systems. Across this body of related work, little research was reported for supporting large-scale requirements decision making by using visualization techniques.

3 Case Study Description

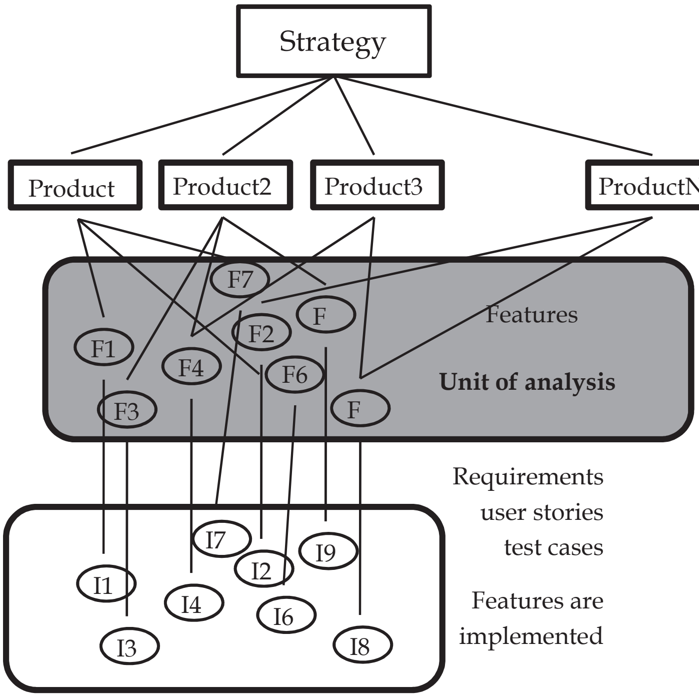


Figure 6.1: The company's context.

The case study was conducted at a large company that uses integrated and incremental development model to deliver embedded software system product lines (Schmid, 2002) for the global market. The company has recently transitioned to an agile-inspired methodology inspired by ideas and principles from eXtreme programming (XP) (Beck and Andres, 2004) and Scrum (Schwaber and Beedle, 2002). The continuous development model, with frequent product releases, replaces a phase-based process.

As a part of the process transitions, the company has adopted the following practices: (1) one continuous scope and release planning flow, (2) cross-functional development teams, (3) gradual and iterative detailing of requirements, (4) integrated requirements engineering and (5) user stories. The company emphasizes user stories (Cohn, 2004) for capturing user intentions and emphasizes user story acceptance criteria to ensure that user stories are correctly implemented. Requirement details are now iteratively refined and all requirements related work is delivered by a cross-functional development team containing developers, business analysts, requirements engineering and software architects.

The company's context is depicted in Figure 6.1. The corporate strategy describes the long term (two to three year) goals and strategies, similar to the concept of roadmaps described by Regnell and Brinkkember (2005). The company releases about 50 products on yearly basis with product definitions derived from the strategies and containing up to 100 new features. Some products are only small adaptations of previous products while other products contain significantly more innovation represented by (up to) 100 new features. Features are elaborated as user stories and requirements then implemented using test-driven development.

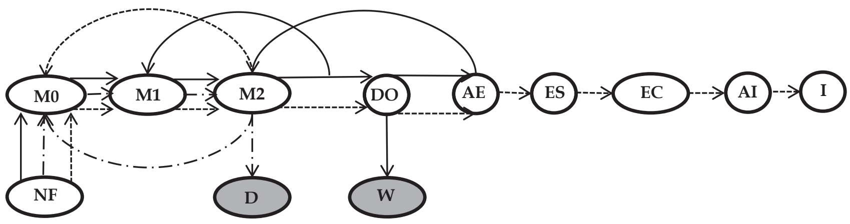


Figure 6.2: An example history of changes for three features. The first feature, marked with dashed lines was successfully implemented. The second feature marked with solid arrows was withdrawn. The third feature marked with dashed-dotted arrows was discarded. The states are marked as follows: NF - New Feature, DS - Definition Started, DO - Definition Ongoing, AE - Awaiting Execution, ES - Execution Started, EC - Execution Completed, W - Withdrawn, D- Discarded. The states marked gray are the terminal states used in the analysis in Section 6

Feature management is performed using the state-machine shown in Figure 6.2. Each newly created feature begins in an administrative state called *New feature* then enters the process at state *M0*. M0 validates that this newly created feature has a sponsor, sufficient business justification and is in line with the associated product strategy. Upon success, the feature is promoted to *M1*. Sets of features in M1 are prioritized, across all product lines, by scope owners using a one-dimensional prioritization method based on business value. The prioritized list is later used by the development teams to guide implementation and integration scheduling. A feature may be returned to the M0 state if further definition or refinement is required.

After prioritization, features are promoted to *M2*. Development resources are consulted and implementation schedules are defined within a continuous delivery pipeline tool that controls resource and delivery scheduling. Due to resource constraints, it is unusual for feature priorities to be modified at this stage due to overbooked development resources or other reasons that impact the delivery date. Generally, the higher the feature priority, the faster it should be implemented and delivered.

Features then pass through several development-related states, including: *Definition Started*, *Definition Ongoing*, *Awaiting Execution*, *Execution Started* and *Execution Completed*. A feature can exit this sequence at any time, transitioning to one of the following states: *Withdrawn*, *Discarded* or *Already Supported*.

Records of all state changes, including their timestamp and reason, are stored in a corporate database. This development history, combined with feature attributes such as product line or stakeholder, was used as the data source for the visualizations presented in this paper. In the three examples depicted in Figure 6.2, we note that two features were sent back to the 'previous' state. This behavior is not prohibited by the process but it negatively impacts overall development efficiency. While the visualization depicted in Figure 6.2 can support the analysis of a small number of features, it is not suitable for the simultaneous analysis of hundreds or thousands of features.

4 Research methodology

In this section, we present the research methodology, the research questions, and the data collection methods used in the study.

4.1 Research questions

Two research questions investigated in this study are outlined in Table 6.1, complemented by aim. The first research question is focusing on different aspects related to efficiency and scalability of the visualized techniques. This question is detailed in four research subquestions. Research questions RQ1a, RQ1b, RQ1c and RQ1d focus on improvement to the Feature Survival Chart+ technique introduced in our previous work. Research question RQ2 investigated visual techniques for decision patterns.

4.2 Research design and methods used

The research reported in this paper has been conducted using an engineering approach (also called pragmatism stance (Easterbrook et al, 2007)) which focuses mainly on obtaining practical knowledge as a valuable source of information. We opted for using a flexible research design and derive theories regarding scope visualization from interviews with practitioners (Robson, 2002). We selected a case study research strategy since

Table 6.1: Research questions.

Research question	Aim
RQ1: How useful is FSC+ as decision support for enabling scope management in very-large agile-inspired context?	To investigate how to visualize scope changes in very-large agile-inspired context.
RQ1a: What should be visualized on the Y-axis of the FSC+?	To investigate what should be represented on the Y-axis.
RQ1b: How visualize the temporal aspect on the X-axis of the FSC+?	To understand how to visualize time on the X-axis.
RQ1c: How the FSC+ should be sorted?	To investigate the most efficient ways of sorting.
RQ1d: What should be filtered when creating the FSC+?	To identify the optimal way of filtering the information to be visualized by the FSC.
RQ2: How do requirements decision patterns facilitate requirements process analysis?	To investigate possible ways of visualizing decision patters.
RQ 2a: How the analysis of decision patterns can help in understanding the requirements scoping process?	To investigate possible ways to analyze decision patterns.
RQ2b: How can visualization of decision patterns can be used to facilitate process improvement?	To investigate possible ways of visualizing decision patters.

it is reported as suitable for software engineering research (Runeson and Höst, 2009) and considered as well suited for requirements engineering research (Wieringa and Heerkens, 2007). Finally, case studies could be used to investigate phenomena in their natural contexts (Yin, 2003).

The study was conducted in an action research mode (Robson, 2002). Action research which focuses on investigating real and relevant problems and improve studied context. Our goals were not only to understand the scope process at the case company but also to make improvement proposals resulting from applying the proposed visualization and analysis techniques directly on the studied case company. In this way, we can receive both (1) an improvement of a practice of some kind and (2) and improved understanding of the studied problem (Wieringa and Heerkens, 2007).

Two methods for collecting empirical data were used in this study. In the first phase of the study, we used content analysis to analyze the data related to features scoping at the case company. This technique is well suited for collecting large volumes of data (Singer et al, 2007). Further, content analysis is an unobtrusive technique where the act of measuring doesn't change the actual values measured (Robson, 2002). Finally, the information collected during content analysis is in a written form which allows automatic analysis and tool support.

In the second part of the study, we used interviews to collect empirical data. Interviews are a straightforward and inquisitive way of collecting opinions and other information (Singer et al, 2007). Despite its time consuming nature, interviews allow asking following up questions, interpreting the tone of the participants' voice and intonations. Furthermore, interviews improve the control over the quality of information gathered during the first part of the study (Singer et al, 2007).

In this study, semi-structured interviews (Robson, 2002) with a high degree of discussion were conducted. We used a set of predetermined question, but the order how asking them could be changed upon the interviewer's perception of what seems to be the most appropriate. Finally, interviews allowed us to discuss the results from the content analysis as well as interpret and validate our findings together with practitioners from the company.

Planning/Selection. Several brainstorming sessions were held. During the sessions, researchers discussed and adjusted the goals of the research with the needs of the practitioners. The requirements scoping process and the data generated by the process were also discussed during the meetings. Further, the decisions which data should be extracted and investigated in this study were made.

When deciding which practitioners to interview, we used a combination of convenience sampling and variation sampling (Patton, 2002). The roles of participants were selected with respect to the interview instrument. One practitioner at the case company acted as a 'gate keeper' ensuring that the selected participants are representative and suitable data points in the study.

Data collection. Researchers were granted access to a database where the history of feature scope changes is stored. Several scripts that extract the history of features were developed and executed. The extracted dataset contained a detailed history of 8728 features . The extracted data was evaluated with one practitioner from the case company to ensure its correctness and usefulness for further analysis.

The history was then analyzed using a tool developed in Java. The initial analysis included parsing and calculating lead-times between various scope states. After the history of each feature was read and analyzed the tool created detailed reports in an Excel format. Finally, the tool visualized the scope changes with given criteria, depicted in Figures 6.3, 6.4 and 6.5.

The interview sessions were scheduled for 60 minutes with the possibility to reduce time or prolong it. The interviews were recorded and analyzed using content analysis. The results from each interview were send back to the interviewees for validation.

4.3 Validity evaluation

In this section, threats to validity are discussed according to four perspectives on validity proposed by Yin (2003).

Construct validity is concerned with establishing appropriate methods and measured for the studied phenomena or concepts. By using the scope changes history stored in a database, we minimized the subjectivity of and analyzed data. Further, the process of data extraction was supervised by one practitioner from the case company who ensured that all administrative changes in the scope and other 'unimportant' data was removed. Finally, we used interviews as a second source of evidence about the studied phenomenon in order to increase our understand and the quality of analyzed data.

Internal validity is concerned with uncontrolled confounding factors that may affect the studied causal relationships. All inferences reported in Section 5 based on the analysis of the datasets were discussed with the interviewees during the interviews. In that way, not only the correctness of the inferences were confirmed but also alternative explanations for observed phenomena were explored. Finally, we used pattern matching of similar behavior of features when developing inferences.

Reliability is concerned with the degree of repeatability of the study. We have reported the procedures used in the study to avoid biases and enable seamless replications. The analysis of the features scope changes was automated and thus complete replicability was ensured. The results from the interviews were recorded and written down to enable further analysis. Finally, the interviewees acted as audits towards the results of the scope changes database analysis.

External validity is concerned with the ability to generalize the study's findings. We are aware that our results are based on a single case study with a single unit of analysis. However, the fact that all available features at the case company (8728) were analyzed in this study greatly increase external validity of the findings.

5 Feature Survival Charts+

The Feature Survival Chart+ technique is demonstrated in Figure 6.3 and Figure 6.5, illustrating two significant development efforts at the case company. The company uses an iterative development model with continuous creation and execution of features. Therefore, the FSC+ charts are an evolution of the previously reported visualizations (Wnuk et al, 2008, 2009) with time now represented on the X-axis as a relative value: all features begin at the origin of the X-axis regardless of their actual start date. Ordering along the Y-axis is based on the time spent in the process before being removed from scope. Features that remain in scope for the duration of a release are at the bottom of the chart; features that are removed early in the release schedule (e.g. in the Withdrawn or Discarded) states are at the top of the chart. Features still in scope are in green, completed features are in blue and withdrawn or discarded features are in red.

This presentation facilitates identification of two behaviors: (1) the corporate response time for scoping decisions and (2) the behavior of features that remain in the process for extended periods. The sooner a feature transitions to blue (analyzed and implemented) or red (removed from scope), the faster the response time and the less opportunity for wasted effort (Wnuk et al, 2012), particularly in rapidly changing MDRE contexts.

FSC+ charts are generated using a Java implementation that supports dynamic filters, multiple y-axis presentation sort orders and control of the meaning and length of the X-axis based on user preferences. The implementation also supports pan and zoom operations and data introspection for individual features. Control over the color scheme and temporal representations (absolute and relative) are also supported.

A magnified view, using the zoom feature, is shown in Figure 6.4. We see an area of 14 features and their early life in the process. One feature (feature number 9855) has been in the process for a very short time (a small green area) and was discarded quite early. To avoid cluttering the view, the implementation shows the time and state names only for features with more than 100 hours in process. The displayed information can vary depending on the magnification level.

Visual inspection of the the main application features stream (Figure 6.3) and the core software components stream (Figure 6.5) shows a green region with a similar shape in both charts. However, the red area seems to be larger in Figure 6.3 (application stream). Further analysis revealed that 831 features from the core components stream and 1351 from the ap-

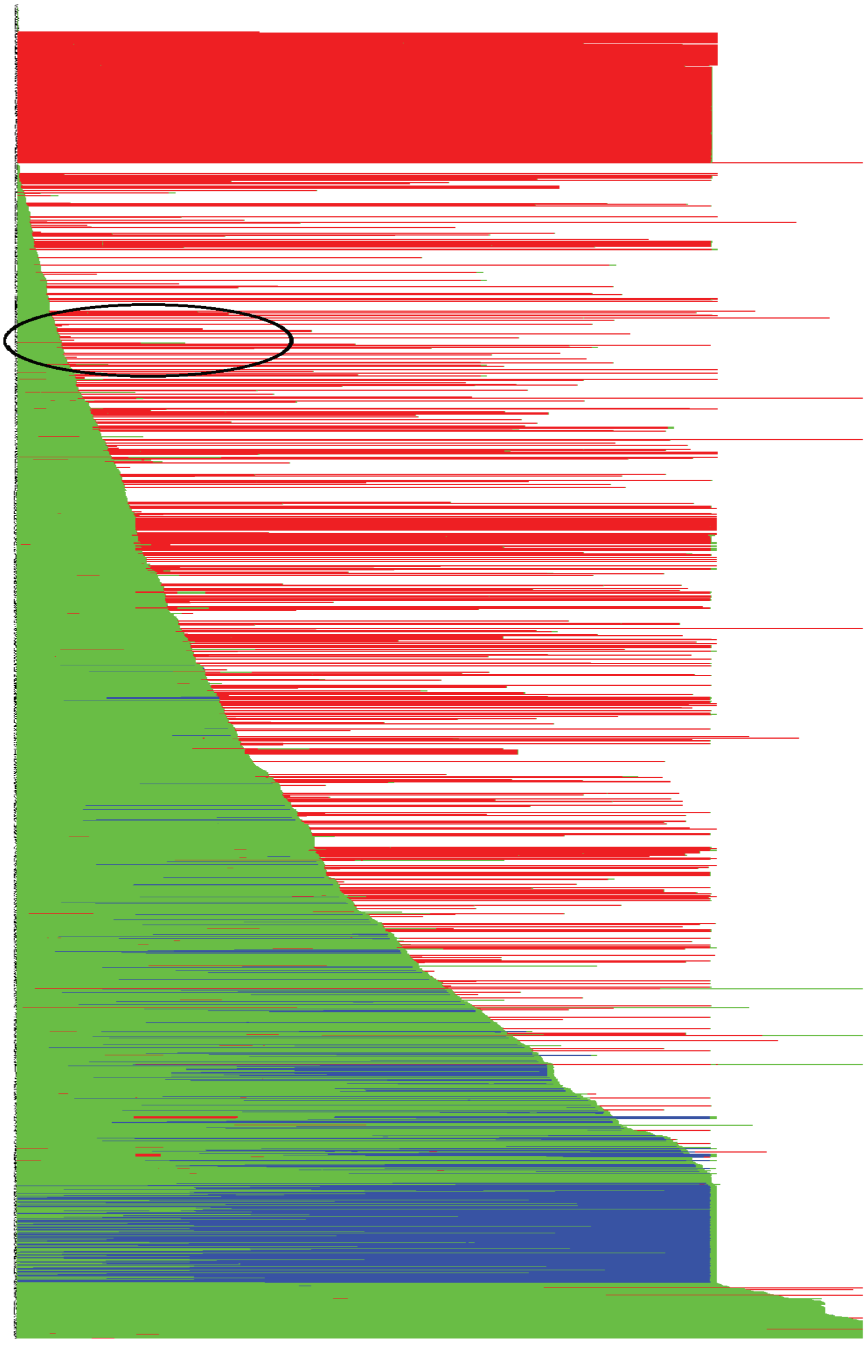


Figure 6.3: FSC+ for the main application features stream of development at the case company.

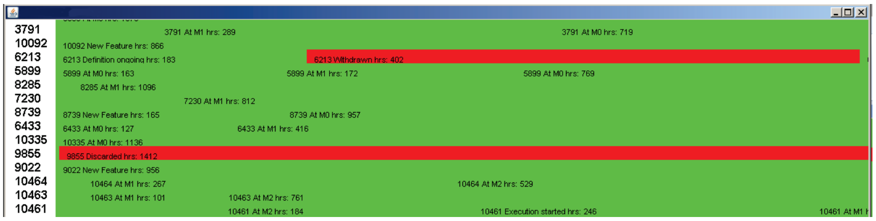


Figure 6.4: An example of a zoomed in view from Figure 6.3

plication stream were rejected. The average time in process before removal was 77 days for the core components features and 65 days for application stream features. This difference is statistically significant (p-value<0.001 for two-sided Kolmogorov-Smirnov test), illustrating the utility of the visualization for guiding further investigative efforts.

The differences between the number of features withdrawn or rejected in these two development streams may indicate that the company prioritizes application features over core component features. This behavior is acceptable, but only to a certain degree, since implementing application features requires implementing core software components first (e.g. one core software components feature can enable several application features). The FSC+ facilitates identification of these differences with a simple visual inspection.

There is also a blue region on both visualizations identifying features (Figures 6.3 and 6.5) that were sent to the development team then quickly completed. This behavior is highly desirable desired by the case company but it can seen that only a small number of features exhibited this behavior. In most cases, significant time elapsed before the features were either implemented or withdrawn. FSC+ supports a rapid assessment of the proportion of features that demonstrate this behavior pattern.

There were 428 core software component features (Figure 6.3) and 635 application features (Figure 6.5) that reached execution completed in the dataset. The average period from first entry in the process to a completed implementation was 132 days for core component features and 90 days for application features, a statistically significant difference (p-value<0.001 for two-sided Kolmogorov-Smirnov test).

A total of 3818 features were withdrawn or discarded from the two streams and their average time in the process was 62 days for the application features and 102 days for core component features. The difference is statistically significant (p-value<0.001 for two-sided Kolmogorov-Smirnov test). The average implementation time, including software definition, was 57 days. Requirements analysis, definition and decision making took, on average, 17 days longer for withdrawn features than for implemented features.

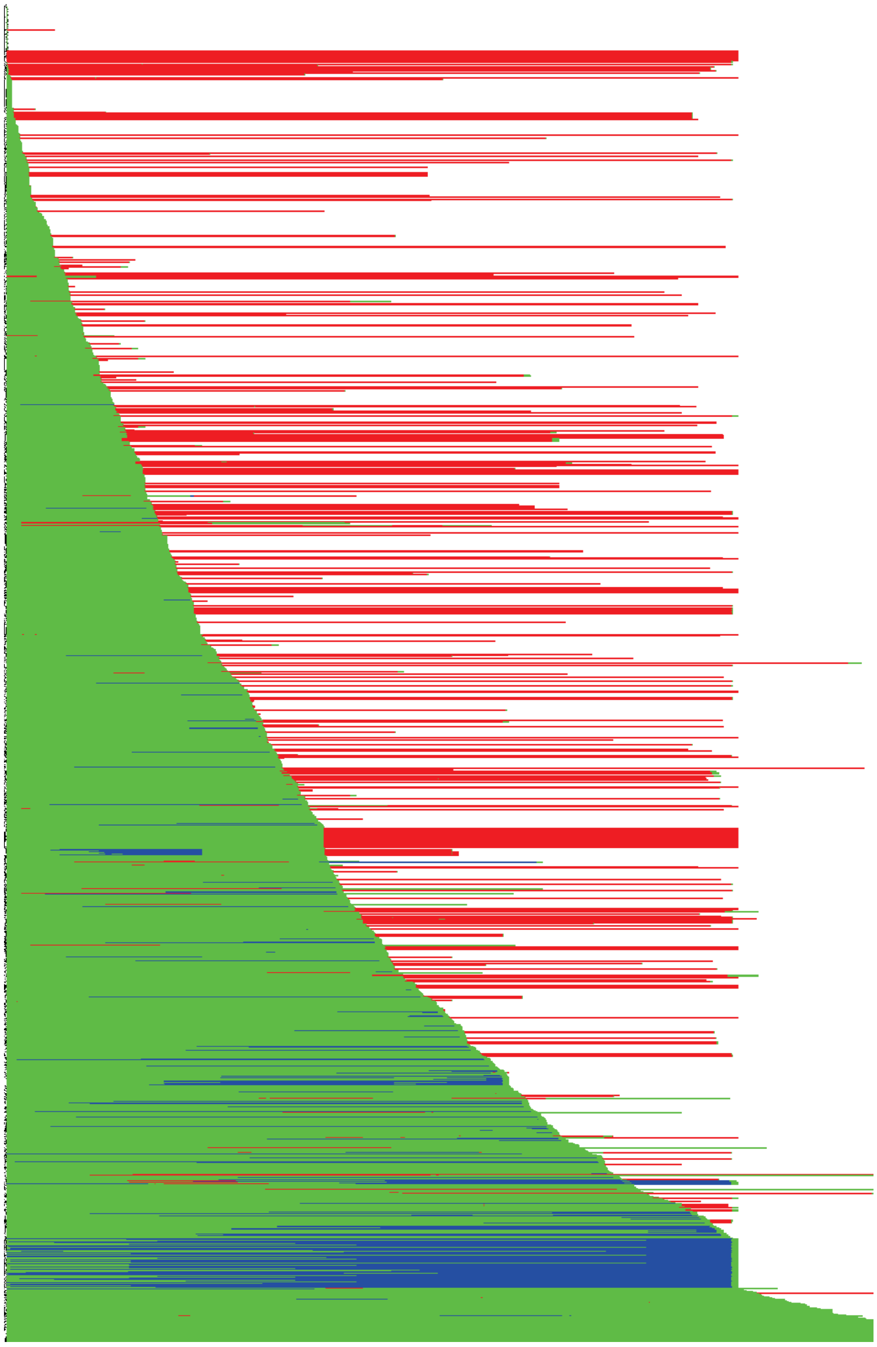


Figure 6.5: The FSC+ for the core software components stream of development at the case company.

Next, we looked at the time spent in requirements definition, analysis and decision making for features that were implemented and discarded. Implemented features spent an average of 21 days in requirements definition and decision making while withdrawn or discarded features spent 44 days in the same states. This difference is statistically significant (p-value<0.001 for two-sided Kolmogorov-Smirnov test). One possible interpretation of this result is that more effort put on requirements engineering and preparing for decision making resulted in rejecting features rather than accepting them.

Feedback during the workshop. The FSC+ visualizations were presented to industry practitioners in a workshop setting. All workshop participants expressed positive opinions about the FSC+ technique. According to the participants, FSC+s facilitate a quick review of scoping history. However, explaining the FSC+, and how they should be interpreted, took more time than expected. In particular, explaining the sort method chosen for the visualization and the accompanying reasoning required extensive explanations for the participants and several ways of sorting the charts were discussed.

The coloring scheme was readily understood and participants considered color an effective presentation of how many features were withdrawn and how many were implemented. Participants considered the ability to visualize all features in database useful, but felt that visualizing subsets, e.g. one product or one development stream, as considerably more useful.

The large number of decision patterns was highly surprising for the workshop participants and several of them insisted on booking individual follow up interviews with the lead researcher to further investigate this phenomenon.

5.1 FSC+ as a visual technique for very-large agile-inspired projects: RQ1

Process managers and those who work with process development graded FSC+ as *very useful*. Two respondents gave FSC+ a score of 9 out of 10 points (*extremely useful*) and three respondents graded FSC+ as *very useful*. However, respondents who worked with limited sets of features (usually not more than 100) on a daily basis graded FSC+ as only *partly useful*. One respondent mentioned that "I remember what is happening with my features so this solution is not particularly useful for me". Thus, the usefulness of FSC+ appears to be directly correlated with the size of the dataset, reaching full potential with very-large datasets. Nevertheless, the respondents who worked with features on daily basis expressed a desire to visualize and analyze the subset of features for which they were responsible.

All respondents considered pan and zoom capabilities positively. Two respondents suggested that the panning capability facilitated direct comparisons between adjacent charts, facilitating alignment. One respondent, who worked with features on a daily basis, did note that filters could be used instead of zooming.

Tasks that FSC+ could support. Interview respondents identified several further tasks that FSC+ could potentially support. Communicating the current scope and scope changes were the most frequently mentioned tasks. Five respondents mentioned providing an overview of the "scope history" and the "health of the scope". One respondent suggested that FSC+ could help visualize congestion and to visualize how various subprocesses are synchronized. Another interviewee suggested that FSC+ can show the wasted implementation effort for features that were sent to implementation and then canceled. Three respondents mentioned that FSC+ can help analyze features that are "stuck in the process": FSC+ are much better than the current database filters since they also visualize how long the features remained in the process without a final decision. Nine out of ten respondents pointed out that the ability to compare FSC+ that were generated for differing criteria is *very useful* for improved process understanding and can provide valuable feedback for process improvement activities.

5.2 X-axis and Y-axis representations: RQ1a, RQ1b

The majority of interview respondents suggested that the Y-axis should represent development effort and the greater the effort required for a certain feature the thicker the line. As a result, the green color areas will more accurately represent the effort spent on both implemented and withdrawn features. At the same time, the green areas for implemented features will represent the total effort spent on them. Two respondents suggested that priority should be on the Y-axis. On the other hand, three respondents said that priority is changing frequently and therefore it would be less useful than effort. Finally, one respondent pointed out that the current 'one-size for all features' representation is sufficient since it gives a good overview about the number of features.

Regarding the X-axis, four respondents suggested analyzing one year in the past. One respondent suggested that the unit on the X-axis should be one week. On the other hard, two respondents working with process management and improvement wanted to see the entire time on the X-axis. One respondent suggested starting the FSC+ chart from the *M2* state since then features are discussed with development effort. Another respondent wanted FSC+s to start from the *definition ongoing* state since that is the moment were the implementation effort and waste can be calculated. The current FSC+ implementation allows starting the chart from any point on the X-axis.

5.3 Sorting and filtering FSC+s: RQ1c and RQ1d

Four respondents suggested that the FSC+ should be filtered per products. At the same time, 5 respondents suggested that looking at the entire stream of development is interesting from the management and overview perspective. One respondent wanted to filter out only legacy features, so features that have already been noticed in the system for earlier activities. Five respondents suggested filtering out: (1) executed features, (2) withdrawn features and (3) features in the process. Among them, three respondents wanted to filter out features that were stuck in the process for a long time. Finally, one respondent suggested filtering per stakeholders that issued features.

Two respondents suggested sorting per lead-time in a specific state. One respondent focused on sorting by the lead-time in *M2* state, one focused on the *M1* state. Sorting per *M2* leadtime gives an overview of how log it took to make an agreement with the development. Sorting per *M1* state gives an overview of how long the prioritization phase took. Further, three respondents stressed that the current way of sorting (see Figure 6.3 and Figure 6.5) is sufficient and gives a good overview of the agility of the analysis and decision processes. Finally, two respondents suggested sorting by priority and placing the most important features at the top of FSC+. This view allows comparing the lead-time for the most important features (placed at the top) with the least important (pleaced at the bottom).

The three respondents that wanted to filter out features that are stuck in the process suggested sorting per total time in the process, including requirements analysis and software definition states. One respondent suggested sorting per time in *definition ongoing* state while other respondent suggested sorting per time in *execution started*. All mentioned ways of sorting are possible within the current implementation.

6 Feature decision patterns analysis and visualization: RQ2

Decision patterns are state changes that features undergone. 2248 decision patterns were identified in the dataset. The decision patterns were analyzed in collaboration with industry participants from the case company. Table 6.2 depicts the top twenty decision patterns. The patterns were sorted according to their frequency in the dataset. The top 10 patterns have over 100 features. The patterns were categorized into *unsuccessful*, *successful* and *in process*. *Successful* decision patterns represent features that were implemented, *unsuccessful* represent features that were withdrawn or discarded and *in process* represent features still in the process.

There seems to be no dominant pattern in the dataset as the most frequent pattern was followed by only 294 features out of 8728. The the two

Table 6.2: Feature decision patterns.

Frequency	Type	Decision pattern
294	Unsuccessful	M0 –> D –> M0 –> D –> M1
244	Unsuccessful	M0 –> M2 –> M0 –> D
231	Successful	M0 –> M1 –> M2 –> DO –> AE –> ES –> EC
201	In process	M0 –> M1 – M0 –> NF
194	In process	M0 –> M1 –> M2
151	Unsuccessful	M0 –> W –> M0 –> W
132	In process	M0 –> M1 –> M2 –> DO
109	In process	M0 –> M2
103	In process	M0 –> M1 –> M2 –> DO –> AE –> ES
102	Successful	M0 –> M2 –> AE –> EC –> M0 –> EC –> M0 –> EC
98	In process	M0 –> M1 –> M0
93	Successful	M0 –> EC –> M1 –> EC –> M0 –> EC –> M0 –> EC –> M0 –> EC
88	Unsuccessful	M0 –> D –> M1 –> D
80	Unsuccessful	M0 –> M2 –> D –> W –> D –> W
79	Successful	M0 –> M1 –> M2 –> EC
75	In process	M0 –> M1
73	Successful	M0 –> M2 –> EC –> M0 –> EC –> M0 –> EC
72	In process	M0
66	In process	M0 –> NF
64	Unsuccessful	M0 –> M1 –> W

most frequent patterns turned out to be unsuccessful patterns. Further, two most frequent patterns have 4 or 5 state changes. In both patterns, features were send back in the process to the initial state called *M0*. The third most frequent decision pattern was a successful pattern. In this decision pattern features went straight through the state machine, according to the official process description, and were not send back in the process.

Nine out of 20 most frequent decision patterns are ongoing patterns. As these features have not yet reached their final states, we could assume that their final decision patterns will change. Five patterns are successful patterns and six patterns are unsuccesful patterns. Looking further at the frequency of the patterns, there seems to be no clear dominance of one type of decision pattern over another. Therefore, we further analyzed the identified decision pattern to identify decision archetypes.

6.1 Decision archetypes: RQ2a

Decision archetypes are the feature decision characteristics based on the analyzed decision patterns. The archetypes combine time characteristics and decision patterns of features. We have identified five decision archetypes based on our dataset:

- *Fast execution* - is the most desirable feature archetype from the process efficiency perspective. Fast executions are features that were quickly analyzed and executed with none or not many backwards transitions. The third most frequent decision pattern (with 231 features) is an example of not sending features back in the process. Only 190 features out of 8728 went through the process in less than 3 month and with one or none decision cycles. 437 features were implemented in not more than 3 months and with up to 3 decision cycles.

- *Fast rejection* - is the second most desirable feature archtype from the requirements and product definition efficiency perspective. We define a fast rejected feature as a feature that was rejected in less than one month from its inception and with not more than one decision cycle. Only 135 features were categorized as fast rejection based on the above criteria. Increasing the time to reject to 3 months (more than the average leadtime for all withdrawn features) gave 249 fast rejections. Therefore, it appears that the number of decision cycles is influencing the behavior more than the leadtime. Allowing up to 3 decision cycles and 3 months lead-time gave 1104 fast rejection features.

- *Slow execution* - is still successful but less desirable feature archetype from the process efficiency perspective. Slow executions are features that were executed in more than 3 months. 2176 features were categorized to this archetype. Further, for 188 features there were four or more decision cycles (passes through the same state in the state machine) before these features reached implementation.

- *Slow rejection* - is the decision archetype where a feature stays in the process for quite some time, circulates over the state machne and is finally rejected. 2293 feature took more than 3 months to reject or withdrawn. Further, 617 features cycled three or more times throughout the state machine before being withdrawn or executed.
- *Evolving* - is a feature that was sent back in the process process. 5863 features (67%) were categorized as evolving, 1950 features (22%) were not evolving and for 915 features it was not yet sure if they will be evolving or not.

6.2 Interview results regarding decision patterns analysis: RQ2

All respondents considered analyzing decision patterns as useful. Five respondents suggested that analyzing decision patterns helps to see how many back and forth transitions were made. Three respondents suggested that the decision patterns analysis could help in understanding to what degree the process was followed. In particular, one respondent suggested that decision patterns analysis could help to understand how the process was implemented by different organizations of the company.

All interview respondents were negatively surprised about the number of decision patterns identified. One respondent suggested that the high number is a result of using the process to discuss features by changing the states back and forth rather than making a state change after these discussions. Three respondents pointed out that the high number of patterns is related with a lot of administrative changes in the features databse. Further it may be a result of pure process knowledge among some practitioners and thus incorrect usage of the tool in relation to the process description. Finally, one respondent mentioned that "this analysis is very interesting and should be presented to the top management".

6.3 Visualizing decision patterns: RQ2b

Figures 6.6 and 6.7 depict atomic decisions visualized using a flowgraph. The arrows in the graph represent atomic transitions. The length of the arrows does not have any meaning. The width of the arrows represent the frequency of a certain transition in the entire dataset. The transitions are divided into *forward transitions* (Figure 6.6) and *backward transitions* (Figure 6.7). Forward transitions are transitions that lead to implementation while backwards transitions are the opposite transitions. Transitions from the two terminal states (*withdrawn* and *discarded*) and from *execution completed* are also considered as backwards transitions.

The most frequent transition among all analyzed transition is the transition between *M0* and *M1* states. The second most frequent transition is between *M1* and *M2*. Interesthignly, the fourth most frequent transition in the dataset is the transition between *M0* state and *execution completed*. This transition, visualized at the top of Figure 6.6 is a way of "bypassing" the entire process and sending the features from the start to the end. Administrative changes in the tool are one of the reasons for this transitions together with the fact that several features cycled in the process back and forth.

Results from interviews. Four of the interview respondents considered the visualizations (Figures 6.6 and 6.7) as partly useful and the remaining respondents very useful. The reason why some respondent considered the visualization partly useful was because they would prefer the visualizations of the streams done separately since every organization implemented some additions to the process. However, all these four respondents agreed that the visualization of the entire dataset shows general process adherence at the case company.

Respondents working with process improvement and management, including high-level management considered visualizations as very useful. The most interesting transitions to analyze turned out to be transitions *M0 –> Execution completed*. One manager mentioned that these kind of transitions are not possible according to the official process description and thus it is very valuable to investigate the reasons for this way of using the process. Another respondent suggested that the transition *M0 –> Execution completed* was used to clean up features that were in the process for a long time (execution completed state was used instead of withdrawn or discarded). Another respondent explained that the high number of these transition s is caused by the fact that one feature was reused in several product and thus resubmitted to the processes several times. One respondent suggested that visualization (Figure 6.6 and Figure 6.7) clearly shows that the process was used as a way to communicate and resolve questions regarding features instead of email of live negotiations.

Three respondents mentioned that Figures 6.6 and 6.7 clearly show that some people did not use the tool and the process in the right way. Finally, one respondent pointed out that the visualizations show that the democratic model was insufficient at the company and suggested that a dictatorship model would have been much better in this case. His conclusion in especially interesting in relation to the Ultra Large-Scale Systems literature (Northrop et al, 2006) suggesting that for ultra-large systems overall control is not possible and thus these systems need to exist as separately managed system. In our case, it seems to be that around 10 000 features is not yet large enough for efficient decentralized management.

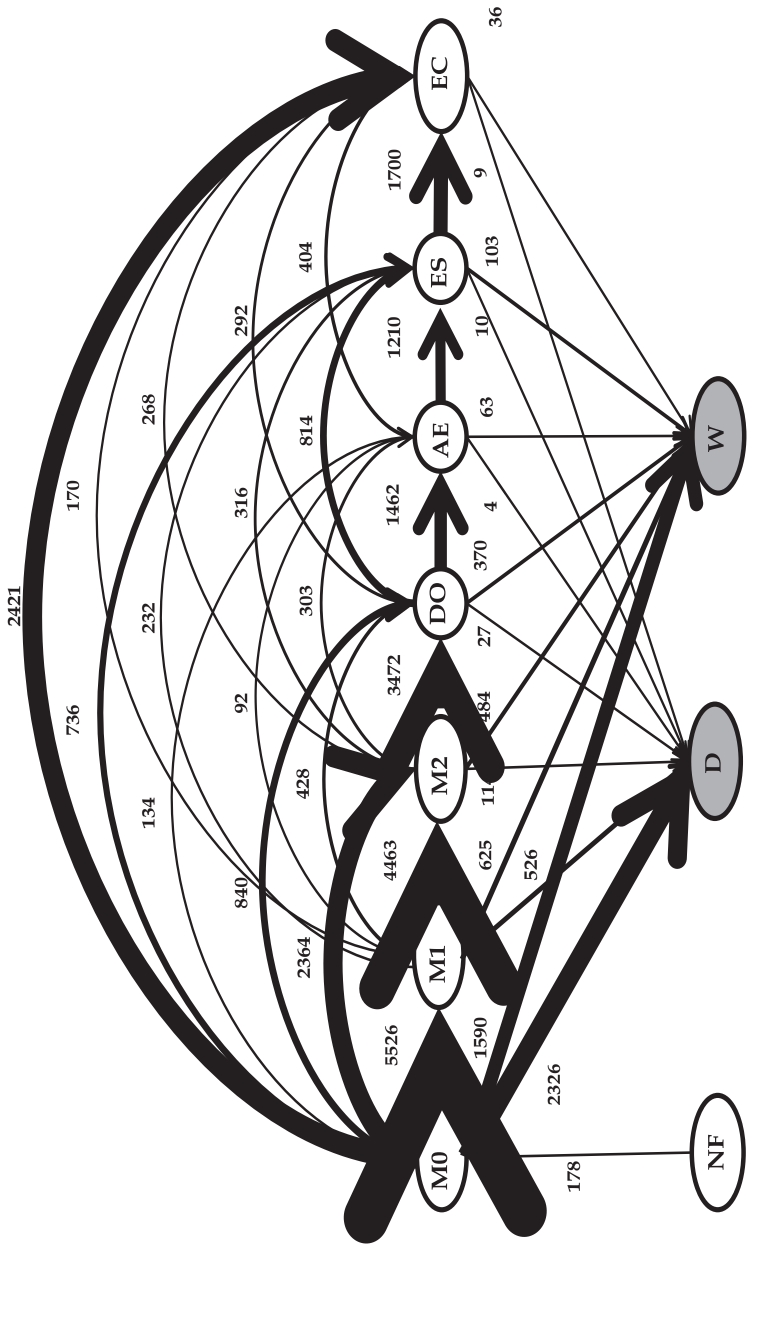


Figure 6.6: Forward transitions visualized.

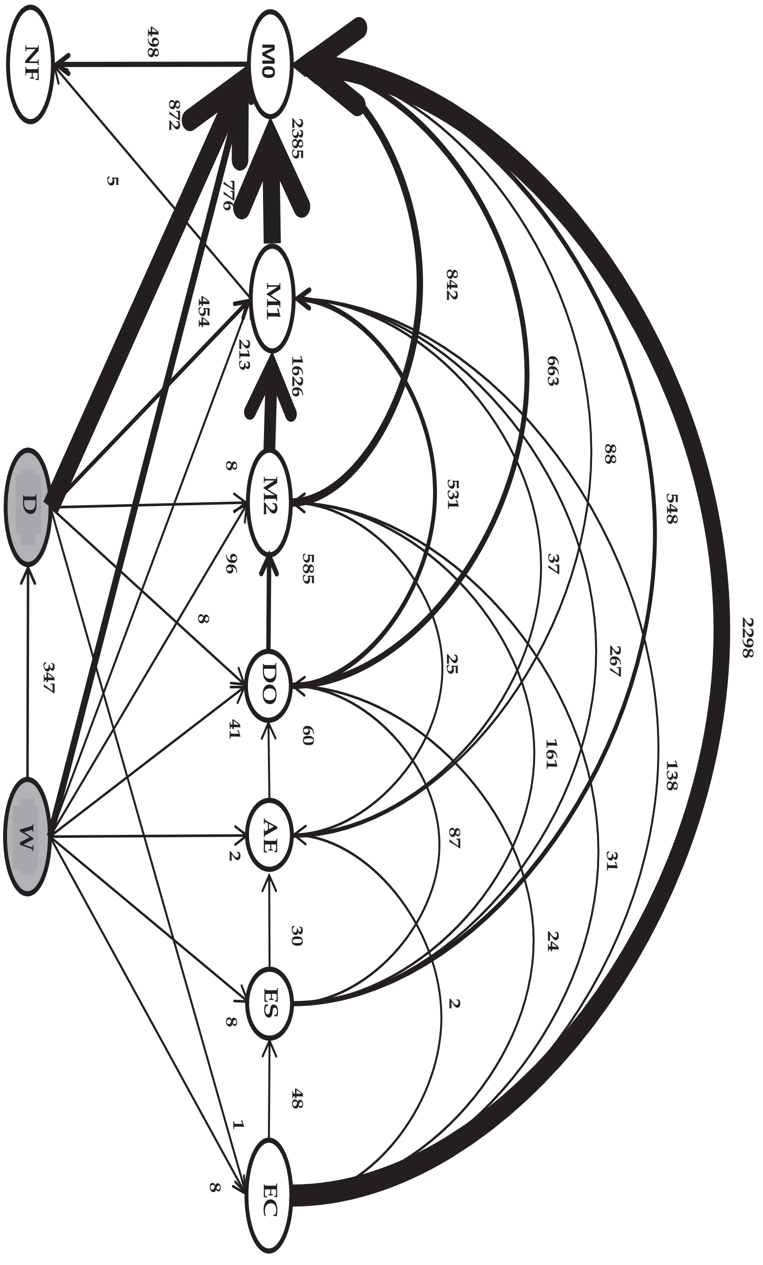


Figure 6.7: Backward transitions visualized.

7 Conclusions

Effective scope management plays an important role in successful project management (Iacovou and Dexter, 2004; Bjarnason et al, 2012). Issues with managing the scope may have negative consequences upon software projects (Bjarnason et al, 2012). Development process information for 8728 features was analyzed to investigate topics of interest such as process efficacy, efficiency and adherence. Management decision patterns were extracted from the production dataset and five archetypical decision patterns in use at the case company were identified: fast execution, slow execution, fast rejection, slow rejection and evolving. The detailed analysis of process patterns identified unexpected behaviors at the case company, triggering further internal investigations.

Given the quantity of data available in this very-large scale requirements engineering context, new visualization techniques were required to effectively capture meaning and information from the data set. The FSC+ technique was developed to visualize scope changes across the large number of features and then reviewed by 10 industry practitioners. FSC+s provided useful overviews of the scope history and the "health" of the development efforts currently in scope. In general, FSC+ was positively received but the practitioners indicated that considerable flexibility was necessary to meet the widely varying needs of the stakeholders, particularly in the area of filtering the datasets. The FSC+ charts are generated using a Java implementation that supports dynamic filters, multiple y-axis presentation sort orders and control of the meaning and length of the X-axis based on user preferences. The implementation also supports pan and zoom operations and data introspection for individual features.

Atomic decision patterns throughout the dataset were summarized using flow graphs with visually weighted edges representing frequency of occurrence within the dataset. The industry practitioners found that this visualization allowed them to quickly identify unexpected behavior in the development process.

The reviewing practitioners found utility in all aspects of the present work with senior management finding the work slightly more useful than those charged with day to day execution. The new visualization techniques greatly facilitate comprehension of process operations compared to direct inspection of the raw data and practitioners felt that they could be used to guide management decisions such as process optimization.

Future work focuses on further improvements to the FSC+ technique and further improvements to the performance of the visualization tool. Further empirical studies to confirm or deny the industrial applicability of the technique are planned. Finally, further investigations of feature decision archetypes will attempt to determine whether the identified archetypes are general, rather than specific to the case company, and whether there may be any omissions from the set of archetypes.

Acknowledgment

This work was supported by VINNOVA within the UPITER2 project. We would like to thank all anonymous interviewees and workshop respondents for their invaluable contribution to this project.

Appendix A: Interview questions

Usefulness of visualizing the scope and FSC+:

1. How useful is according to You the FSC+ technique for scope visualization?

2. Which tasks can FSC+ support?

3. How often would you use scope visualization in your daily work?

Show a FSC for their context and discuss the possible extensions in Y -axis

4. What do you think about the moving map solution?

5. What do you think about the zooming in solution?

6. How would you like to be represented in the Y-axis? (priority, criticality, size, impact etc.)

Filtering and sorting

7. What type of information should be filtered by FSC+?

8. How would you like the chart to be sorted?

9. What is the most optimal time-spam to visualize?

Decision patterns

10. Do you think that analyzing decision patterns would be useful for your work?

11. Do you think that visualizing decision patterns would be beneficial in your work?

Analyzing decision pattern

12. How would you interpret the number of unique patterns in the dataset?

13. How would you interpret the number of cycles in the patterns?

14. Why do so many paths terminate at illogical states?

Bibliography

Alenljung B, Persson A (2008) Portraying the practice of decision-making in requirements engineering: a case of large scale bespoke development. Requirements Engineering 13(4):257–279

Aurum A, Wohlin C (2003) The fundamental nature of requirements engineering activities as a decision-making process. Information and Software Technology 45(14):945–954

Austin M, Mayank V, Shmunis N (2006) Paladinrm: Graph-based visualization of requirements organized for team-based design. Syst Eng 9(2):129–145

Avison D, Fitzgerald G (1998) Information systems development methodologies techniques, and tools. John Wiley

Ball T, Eick S (1996) Software visualization in the large. Computer 29(4):33 –43

Basili V, Rombach H (1988) The tame project: towards improvement-oriented software environments. IEEE Transactions on Software Engineering 14(6):758 –773

Beck K, Andres C (2004) Extreme Programming Explained: Embrace Change (2nd Edition). Addison-Wesley Professional

Berenbach B, Paulish D, Kazmeier J, Rudorfer A (2009) Software & Systems Requirements Engineering: In Practice. Pearson Education Inc.

Bergman M, King J, Lyytinen K (2002) Large scale requirements analysis as heterogeneous engineering. Scandinavian Journal of Information Systems 14(4):37–55

Bjarnason E, Wnuk K, Regnell B (2012) Are you biting off more than you can chew? a case study on causes and effects of overscoping in large-scale software engineering. Information and Software Technology 54(10):1107 – 1124

Boehm B (2006) Some future trends and implications for systems and software engineering processes. Systems Engineering 9(1):1–19

Carlshamre P (2002) Release planning in market-driven software product development: Provoking an understanding. Requirements Engineering 7:139–151

Carlshamre P, Sandahl K, Lindvall M, Regnell B, Natt och Dag J (2001) An industrial survey of requirements interdependencies in software product release planning. In: Proceedings of the Fifth IEEE International Symposium on Requirements Engineering (RE 2001), pp 84–91

Carman D, Dolinsky A, Lyu M, Yu J (1995) Software reliability engineering study of a large-scale telecommunications software system. In: Proceedings of the Sixth International Symposium on Software Reliability Engineering, pp 350 –359

Carter RA, Antón A, Dagnino A, Williams L (2001) Evolving beyond requirements creep: A risk-based evolutionary prototyping model. In: Proceedings of the 5th IEEE International Symposium on Requirements Engineering (RE'01, pp 94–101

Checkland P (1981) Systems Thinking, Systems Practice. John Wiley and Sons Ltd.,

Cohn M (2004) User Stories Applied: For Agile Software Development. The Addison-Wesley Signature Series, Addison-Wesley

D'Ambros M, Lanza M, Pinzger M (2007) "a bug's life" visualizing a bug database. In: Proceedings of the 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis, VISSOFT 2007, pp 113 –120

DeGregorio G (1999) Visual tool support for configuring and understanding software product lines. In: Proceedings of the 9th International Symposium of the International Council on System Engineering, pp 1–7

DeMarco T, Lister T (2003) Risk management during requirements. Software, IEEE 20(5):99 – 101

Denne M, Cleland-Huang J (2004) The incremental funding method: Data-driven software development. IEEE Software 21:39–47

Easterbrook S, Singer J, Storey M, Damian D (2007) Guide to Advanced Empirical Software Engineering, Springer, chap Selecting Empirical Methods for Software Engineering Research, pp 285–311

Ebert C (2004) Dealing with nonfunctional requirements in large software systems. Annals of Software Engineering 3(1):367–395

Fogelström N, Barney S, Aurum A, Hederstierna A (2009) When product managers gamble with requirements: Attitudes to value and risk. In: Proceedings of the 15th International Working Conference on Requirements Engineering: Foundation for Software Quality, Springer-Verlag, Berlin, Heidelberg, REFSQ '09, pp 1–15

Garg P (1989) On supporting large-scale decentralized software engineering processes. In: Proceedings of the 28th IEEE Conference on Decision and Control, pp 1314 –1317 vol.2

Gorschek T, Wohlin C (2006) Requirements abstraction model. Requirements Engineering Journal 11:79–101

Gotel O, Marchese F, Morris S (2007) On requirements visualization. In: Proceedings of the Second International Workshop on Requirements Engineering Visualization, IEEE Computer Society, Washington, DC, USA, REV '07, pp 11–20

Greer D, Ruhe G (2004) Software release planning: an evolutionary and iterative approach. Information and Software Technology 46(4):243 – 253

Hall T, Beecham S, Rainer A (2002) Requirements problems in twelve software companies: an empirical analysis. IEEE Software 149(5):153 – 160

Hanakawa N (2007) Visualization for software evolution based on logical coupling and module coupling. In: Proceedings of the Asia-Pacific Software Engineering Conference, APSEC 2007. 14th Asia-Pacific, pp 214 – 221

Herbsleb J (2007) Global software engineering: The future of socio-technical coordination. Future of Software Engineering 1(1):188–198

Höst M, BRegnell, JNatt och Dag, Nedstam J, CNyberg (2001) Exploring bottlenecks in market-driven requirements management. Journal of Systems and Software 59(3):323–332

Iacovou C, Dexter A (2004) Turning around runaway information technology projects. Engineering Management Review, IEEE 32(4):97 –112

Institute PM (2000) A Guide to the Project Management Body of Knowledge (PMBOK Guide), 2000, Project Management Institute, Four Campus Boulevard, Newtown Square, PA 19073-3299, USA, chap Chapter 5: Project Scope Management, pp 47–59

Jantunen S, Lehtola L, Gause D, Dumdum U, Barnes R (2011) The challenge of release planning. In: Proceedings of the Fifth International Workshop on Software Product Management (IWSPM'2011), pp 36 –45

Jermakovics A, Sillitti A, Succi G (2011) Mining and visualizing developer networks from version control systems. In: Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering, ACM, New York, NY, USA, CHASE '11, pp 24–31

Karlsson J, Ryan K (1997) A cost-value approach for prioritizing requirements. IEEE Software 14(5):67–74

Karlsson L, Dahlstedt s, Natt Och Dag J, Regnell B, Persson A (2002) Challenges in market-driven requirements engineering - an industrial interview study. In: Proceedings of the Eighth International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ 2002)

Kishi T, Noda N, Katayama T (2002) A method for product line scoping based on a decision-making framework. In: Chastek G (ed) Software Product Lines, Lecture Notes in Computer Science, vol 2379, Springer Berlin / Heidelberg, pp 53–65, 10.1007/3-540-45652-X22

Klein G, Orasanu J, Calderwood R, Zsambok C (1995) Decision making in action: Models and methods. New Jersey Norwood

Konrad S, Gall M (2008) Requirements engineering in the development of large-scale systems. In: Proceedings of the 16th International Requirements Engineering Conference (RE 2008), pp 217–222

Kulk G, Verhoef C (2008) Quantifying requirements volatility effects. Sci Comput Program 72(3):136–175

van Lamsweerde A, Letier E (2004) From object orientation to goal orientation: A paradigm shift for requirements engineering. In: Wirsing M, Knapp A, Balsamo S (eds) Radical Innovations of Software and Systems Engineering in the Future, Lecture Notes in Computer Science, vol 2941, Springer Berlin / Heidelberg, pp 153–166

Legodi I, Barry M (2010) The current challenges and status of risk management in enterprise data warehouse projects in south africa. In: Proceedings of the 2010 Proceedings of PICMET '10: Technology Management for Global Economic Growth, pp 1 –5

Lehtola L, Kauppinen M (2006) Suitability of requirements prioritization methods for market-driven software product development. Software Process: Improvement and Practice 11(1):7–19

Leon D, Podgurski A, White L (2000) Multivariate visualization in observation-based testing. In: Proceedings of the 22nd International Conference on Software Engineering, ACM, New York, NY, USA, ICSE '00, pp 116–125

Livieri S, Higo Y, Matushita M, Inoue K (2007) Very-large scale code clone analysis and visualization of open source programs using distributed ccfinder: D-ccfinder. In: Proceedings of the 29th international conference on Software Engineering, IEEE Computer Society, Washington, DC, USA, ICSE '07, pp 106–115

Lubars M, Potts C, Richter C (1993) A review of the state of the practice in requirements modeling. In: Proceedings of the IEEE International Symposium on Requirements Engineering (RE'93), pp 2 –14

Moody D (2009) The physics of notations: Toward a scientific basis for constructing visual notations in software engineering. IEEE Transactions on Software Engineering 35:756–779

Ngo-The A, Ruhe G (2005) Engineering and Managing Software Requirements, Springer, chap Decision Support in Requirements Engineering, pp 267–286

Northrop L, Felier P, Habriel R, Boodenough J, Linger R, Klein M, Schmidt D, Sullivan K, Wallnau K (2006) Ultra-Large-Scale Systems: The Software Challenge of the Future. Software Engineering Institute

Oezbek C, Prechelt L, Thiel F (2010) The onion has cancer: some social network analysis visualizations of open source project communication. In: Proceedings of the 3rd International Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development, ACM, New York, NY, USA, FLOSS '10, pp 5–10

Patton M (2002) Qualitative Research & Evaluation Methods. Sage Publication Ltd

Potts C (1995) Invented requirements and imagined customers: requirements engineering for off-the-shelf software. In: Proceedings of the Second IEEE International Symposium on Requirements Engineering (RE 95), pp 128–130

Regnell B, Brinkkemper S (2005) Engineering and Managing Software Requirements, Springer, chap Market–Driven Requirements Engineering for Software Products, pp 287–308

Regnell B, Kuchcinski K (2011) Exploring software product management decision problems with constraint solving - opportunities for prioritization and release planning. In: Proceedings of the Fifth International Workshop on Software Product Management (IWSPM'2011), pp 47 –56

Regnell B, Beremark P, Eklundh O (1998) A market-driven requirements engineering process: Results from an industrial process improvement programme. Requirements Engineering 3:121–129

Regnell B, Berntsson Svensson R, Wnuk K (2008) Can we beat the complexity of very large-scale requirements engineering? In: Lecture Notes in Computer Science, vol 5025, pp 123-128

Robson C (2002) Real World Research. Blackwell Publishing

Ruhe G (2009) Product Release Planning: Methods, Tools and Applications. Auerbach Publications

Ruhe G, Ngo A (2004) Hybrid intelligence in software release planning. Int J Hybrid Intell Syst 1(1-2):99–110

Ruhe G, Saliu M (2005) The Art and Science of Software Release Planning. IEEE Software 22(6):47–53

Runeson P, Höst M (2009) Guidelines for conducting and reporting case study research in software engineering. Empirical Software Engineering Journal 14(2):131–164

Savolainen J, Kauppinen M, Mannisto T (2007) Identifying key requirements for a new product line. In: Proceedings of the 14th Asia-Pacific Software Engineering Conference, IEEE Computer Society, Washington, DC, USA, APSEC '07, pp 478–485

Schmid K (2002) A comprehensive product line scoping approach and its validation. In: Proceedings of the 24th International Conference on Software Engineering (ICSE 2002), pp 593–603

Schwaber K, Beedle M (2002) Agile software development with scrum. Series in agile software development, Prentice Hall

Sen A, Jain S (2007) A visualization technique for agent based goal refinement to elicit soft goals in goal oriented requirements engineering. In: Proceedings of the Second International Workshop on Requirements Engineering Visualization (REV 2007), pp 2–11

Strigini L (1996) limiting the dangers of intuitive decision making. Software, IEEE 13(1):101 –103

Svahnberg M, Gorschek T, Feldt R, Torkar R, Saleem S, Shafique M (2010) A systematic review on strategic release planning models. Inf Softw Technol 52(3):237–248

Tufte E (1990) Envisioning Information. Graphics Press LLC

UML (2010) The unified modeling language webpage. `http://www.uml.org`

van Den Akker J, Brinkkemper S, Diepen G, Versendaal J (2005) Determination of the next release of a software product: an approach using integer linear programming. In: Proceeding of the 11th International Workshop on Requirements Engineering: Foundation for Software Quality REFSQ'05, pp 247–262

van den Akker M, Brinkkemper S, Diepen G, Versendaal J (2008) Software product release planning through optimization and what-if analysis. Inf Softw Technol 50(1-2):101–111

van der Hoek A, Hall R, Heimbigner D, Wolf A (1997) Software release management. In: In Proceedings of the Sixth European Software Engineering Conference, Springer, pp 159–175

Wieringa R, Heerkens H (2007) Designing requirements engineering research. In: Proceedings of the 5th International Workshop on Comparative Evaluation in Requirements Engineering, pp 36–48

Wnuk K, Regnell B, Karlsson L (2008) Visualization of feature survival in platform-based embedded systems development for improved understanding of scope dynamics. In: Proceedings of the Third International Workshop on Requirements Engineering Visualization (REV 2008), pp 41–50

Wnuk K, Regnell B, Karlsson L (2009) What happened to our features? visualization and understanding of scope change dynamics in a large-scale industrial setting. In: Proceedings of the 17th IEEE International Requirements Engineering Conference (RE 2009), pp 89–98

Wnuk K, Regnell B, Berenbach B (2011) Scaling up requirements engineering: Exploring the challenges of increasing size and complexity in market-driven software development. In: Berry D, Franch X (eds) Requirements Engineering: Foundation for Software Quality, Lecture Notes in Computer Science, vol 6606, Springer Berlin / Heidelberg, pp 54–59

Wnuk K, Callele D, Karlsson EA, Regnell B (2012) Controlling lost opportunity costs in agile development: The basic lost opportunity estimation model for requirements scoping. In: Cusumano M, Iyer B, Venkatraman N (eds) Software Business, Lecture Notes in Business Information Processing, vol 114, Springer Berlin Heidelberg, pp 255–260

Yin R (2003) Case Study Research: Design and Methods. Sage Publications

Yu E (1997) Towards modelling and reasoning support for early-phase requirements engineering. In: Proceedings of the Third IEEE International Symposium on Requirements Engineering (RE'97), pp 226 –235