Understanding and supporting large-scale requirements management

Wnuk, Krzysztof

2010

Understanding and Supporting Large-Scale Requirements Management

Krzysztof Wnuk

Licentiate Thesis, 2010

Department of Computer Science
Lund University
Faculty of Engineering

ISSN 1652-4691
Licentiate Thesis 10, 2010
LU-CS-LIC: 2010-1

Department of Computer Science
Faculty of Engineering
Lund University
Box 118
SE-221 00 Lund
Sweden

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 software 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 understanding and supporting 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, variability management 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 possible improvements for managing variability and presents visualization techniques to assist scope management for large-scale software product development contexts. Both reported ideas are empirically evaluated in case studies in a large-scale context. Additionally, the benefits of using linguistic methods for requirements consolidation are investigated in a replicated experimental study based on a relevant industry scenario.

Acknowledgements

The work presented in this thesis was funded by the Swedish Governmental Agency for Innovation Systems under the grant for UPITER, 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, Dr. Martin Höst, for his enthusiastic guidance and excellent comments on my work.

The research presented in this thesis was conducted in close cooperation between academia and industry. Therefore, I would like to thank everyone involved at Sony Ericsson Mobile Communication AB for their commitment, in particular Dr. Lena Karlsson, Lic. Eng. Thomas Olsson, Claes Schrewelius, and Dr. Even-André Karlsson. I am grateful to all anonymous participants and their companies who have helped in making the data collection possible for 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 would like to thank Lars Nilsson for his 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 would also like to mention all other colleagues at the Department of Computer Science, thanks for providing an excellent environment to work in.

Last but not least, I would like to thank my wife Agata, the light 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 2010

Contents

Introduction

Software is currently more and more pervasive and gaining 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, variability 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 concentrate on understanding and supporting large-scale requirements engineering with a focus on embedded systems product development context. 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, variability management and consolidation tasks of requirements management. The presented visualization techniques are confirmed to scale up to the in this thesis presented large-scale company case context, increasing the understanding and assisting in characterizing and improving the scoping process at the case company.

This thesis includes a collection of five 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 five papers are included in the thesis:

I **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 (REFSQ09), 2009, LNCS vol. 5512, pp. 118-123

II **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 (VaMoS2009), 2009, pp.155-164

III **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 International Requirements Engineering Conference (RE09), 2009, pp.89-98

IV **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 (REV09), 2009

V **Replication of an Experiment on Linguistic Tool Support for Consolidation of Requirements from Multiple Sources**
Krzysztof Wnuk, Björn Regnell and Martin Höst
To be Submitted to Empirical Software Engineering Journal

Contribution Statement

Krzysztof Wnuk is the main author for all 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,III,IV and V 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, III and IV, with the assistance from Professor Björn Regnell and industrial practitioners Dr. Lena Karlsson and Claes Schrewelius respectively.

For Paper II, Krzysztof Wnuk wrote most of the text with the assistance from Professor Björn Regnell, and Jonas Andersson and Samuel Nygren in a master thesis project supervised by Krzysztof Wnuk. Most of the design was performed together with the co–authors, while most of the execution and analysis was performed by Jonas Andersson and Samuel Nygren.

For Paper V, Krzysztof Wnuk contributed in the design of the replicated experiment, experiment execution and analysis. All authors contributed in the discussions and writing; however, Krzysztof Wnuk wrote most of Paper V.

Related Publications

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

VI **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 (REFSQ08), LNCS 5025, pp. 123-12.
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.

VII **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 Requirements Engineering Visualization (REV08), 2008, pp.41-50.

VIII **Investigating Upstream versus Downstream Decision-Making in Software Product Management**
Krzysztof Wnuk, Richard Berntsson Svensson,Björn Regnell
Third International Workshop on Software Product Management (IWSPM 2009), 2009

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 in hardware over the past 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 products. To build these software products, software engineers use their knowledge of computers and computing to solve problems. Therefore, the essential part in the definition of software engineering, provided by Pfleeger, 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 a more general system engineering discipline (Sommerville 2007).

Today's software solutions are often very large and complex. The size and complexity explosion was partly responsible for establishing the software engineering field at the NATO Software Engineering Conference in 1968 (Naur and Randell 1968). Since then, it has 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. As a result, 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 creating 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 target systems. In this context, 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 difference in the 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 (Sommerville 2007, 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):

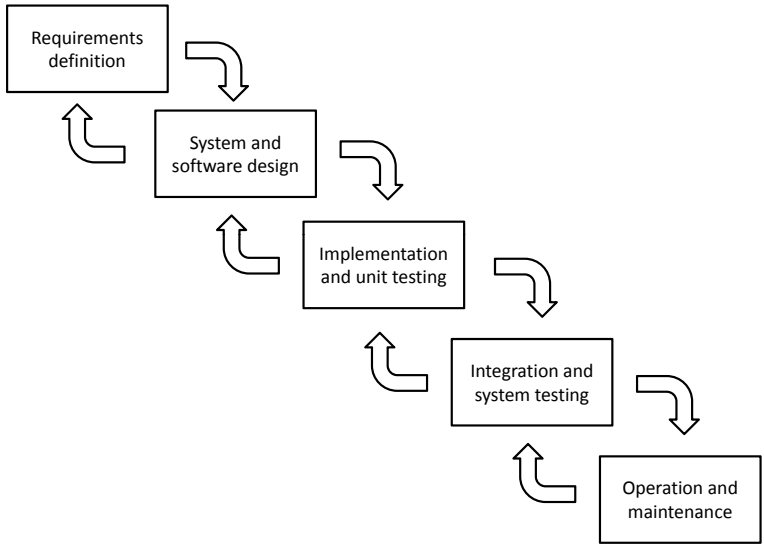

Figure 1: The software life cycle waterfall model (Sommerville 2007, Royce 1970)

- *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 model has also had many critics since being introduced. Among them, 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 models of the software development process have been 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 tries to reduce the opportunity for error by eliminating several development steps (Balzer 1981). Finally, new software development models that have recently emerged, such as Agile (Cunningham 2001) and Software Product Lines (Pohl et al. 2005) 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 2001). 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. This approach helps to cope with complexity of today's software-intensive systems by using platform and mass customization during their development (Pohl et al. 2005).

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 have been 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 given, by stating that requirements engineering is concerned 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 (Sommerville 2007, Kotonya and Sommerville 1998, Neill and Laplante 2003, Berenbach et al. 2009). One of the presented views, 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 process model provided by Sommerville (2007) are complemented by the "feasibility study" subprocess concerned with assessing whether the system is useful to the busi-

ness. 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. These models often fall short on capturing the inevitable complexity of requirements engineering processes in large companies releasing their products to the open market.

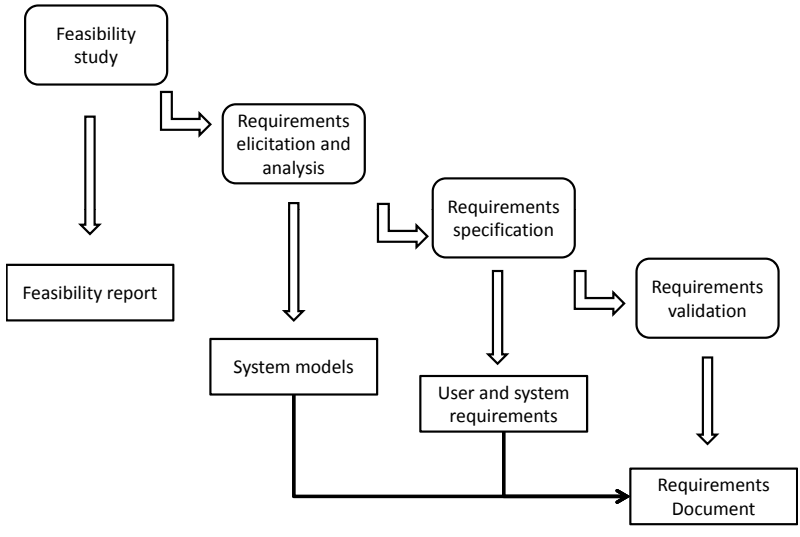

Figure 2: A requirements engineering process example (Sommerville 2007).

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). As pointed out by Kotonya (1998), a common case is that a significant part of an initial system's requirements will be modified before it is put into service. 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 a requirements management context. Therefore, requirements management activities are performed in parallel to the requirements engineering activities and they play a supportive role for them. As already mentioned, change management and change impact assessment are not the only concerns of requirements management. 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. He lists 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 is focusing on some requirements management activities along the requirements engineering life-cycle as it is considered to be a crucial activity for the Market-Driven Requirements Engineering (MDRE) context described below in this section.

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 the market-driven mode, the situation is however much more complex. Here, the success of the software product is mainly de-

pendent on the market response which can not be fully assumed 'a-priori'. Therefore, the release time is also important (Chen et al. 2005, Wohlin et al. 1995, Sawyer 2000), or even for some cases even more important than the functionality that the newly released product is providing. The previous fact puts hard time constrains on the requirements engineering and management activities, demanding them to be more flexible, scalable and less time consuming (McPhee and Eberlein 2002). For example, when setting the scope in a bespoke software project includes time-consuming negotiations and conflict mitigations, in the requirements engineering scenario the scope of the project has to be set using prioritization techniques based on market predictions and effort estimates (Carlshamre 2002b, Karlsson 1998, Sawyer 2000). 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 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 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. As a result, the volume of the requirements database has no chance to shrink and continues to grow, putting requirements management techniques and documentation systems to test. Moreover, 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). Software products in market-driven mode are evolving continuously 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 is focusing 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 the market-driven mode, 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. The amount of embedded software is growing, and the amount of variability is growing even faster. The increased role and importance of software comes with an increased number of requirements. The explosion of new ideas is particularly an inevitable part of a company that operates in MDRE. This 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 one of the related publications, Paper VI, while the definitions and descriptions also are repeated in Section 3.1. The size of the requirements databases in this case may exceed tens of thousands of requirements, which puts new expectations on requirements management 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 V, has been conducted in a VLSRE context.

The concept of producing software by utilizing Software Product Lines (SPL) has gained more and more importance, especially among very large software companies. Moreover, the software product lines concept has already proven to be a successful approach in providing a strategic reuse of assets within an organization (Pohl et al. 2005). The key for the reuse of the common features between products is variability management. Variability can be defined in this context as the possibility of more than one behavior of a software artifact at some point in its lifecycle (Svahnberg 2003). The increased importance of variability management is a result of a changing nature of software product, from originally rather static systems to highly extensible and dynamically changing contemporary software systems (van Gurp et al. 2001). Another part of SPL concept is the process of selecting requirements to implement in the forthcoming project, called *scoping* (Wohlin and Aurum 2005, Greer and Ruhe 2004). Scoping is considered as as a key activity for achieving economic benefits in product line development (Schmid 2002). It is important to mention here that the software product line concept can be applied to more than just the source code and extended to a variety of other artifacts that are used to construct the software product. As a result, the creation of new products is performed by reusing as much software artifacts as possible. From an organizational point of view, 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). The cost decreases, while the complexity of the system when using SLP approach increases. Finally, SPL increases the possibility of flexibility within the organization, since the knowledge in the organization is more widely deployed (Clements and Northrop 2002). The industrial partner, where the research for this thesis has been conducted, is

using the SPL concept.

2 Research Focus

The main research focus of this thesis is requirements management in large or very large-scale contexts. The main research goals are understanding and supporting very large-scale requirements engineering. This goal was later refined and further investigated in Paper I. Furthermore, Paper VI provides three following areas of very large-scale requirements engineering, demanding further research efforts:

- **Sustainable requirements architectures: fighting information overload.** The term requirements architecture is here understood as the underlying structure by which the requirements are organized, including 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 definition of the VLSRE context, the amount of information that must be managed is immense and not possible to grasp in all its details by a single person. In order to fight information overload, we need requirements architectures that are sustainable in the sense that they allow for controlled growth and help the requirements engineers in a large organization to keep track of the myriad of issues that continuously emerge
- **Effective requirements abstractions: fighting combinatorial explosions.** In VLSRE situations where interdependencies among requirements are critical, such as prioritization, resource estimation, and change impact analysis, we inevitably stumble on combinatorial explosions, further fuelled by product line engineering that significantly increases the complexity of the requirements architecture. A major vehicle for fighting these combinatorial explosions may be the use of abstraction mechanisms and experience-based heuristics
- **Emergent quality predictions: fighting over-scoping.** Given a competitive market and a large and demanding set of stakeholders, there seems to be an inevitable shortage of resources to meet quality expectations. The prediction of the system level quality aspects that emerge from a myriad of details is very difficult. As a result, a sustained risk of defining a too large scope for platform development can be observed partly due to the inherent difficulty in understanding quality requirements and predicting their impact on required development resources

In order to increase the understanding of the large-scale requirements engineering context, an empirical interview study was conducted and reported in Paper I. The results from this study played an important role in

setting further goals for the research reported in this thesis. The results of efforts published in the research literature regarding linguistic methods in assisting requirements management activities (Natt och Dag et al. 2005), provided basics for further investigation in this area. Computer linguistic provides methods for automatic analysis of various aspects of natural language documents, including natural language requirements. One of the linguistic methods is calculating the similarity between two requirements. This information can be further used as a help in the analysis of from customers or proxy-customers incoming requirements against requirements already present in the requirements repository. This process is called *requirements consolidation* and is the central part of research question RQ5. Supporting requirements consolidation has been recognized as a relevant industrial problem by Natt och Dag et al. (2006) in a very large-scale requirements management context, where the change management activity gets particularly challenging and requires automatic support methods to be successfully performed. The resulting main research questions investigated in this thesis are as follows:

- **RQ1**: What are the challenging aspects of requirements management processes in a very large-scale context?
- **RQ2**: How is variability managed in practice in a large-scale software product line context?
- **RQ3**: How to characterize and visualize scope change dynamics in a large-scale software development context?
- **RQ4**: How can multiple scope changes be characterized and visualized in a large-scale software development context?
- **RQ5**: Can linguistic methods of finding similar requirements overperform searching and filtering methods for a task of requirements consolidation?

The relationships between the research questions are depicted in Figure 3. Paper I addresses RQ1 which was posed in order to investigate practices of requirements engineering in a very large-scale requirements engineering context and to find areas for further investigations. In this paper, an interview study at Sony Ericsson, where the investigation of the current ways of working with very large sets of requirements was conducted, issues and challenges were identified. These challenges played a central role in the process of further refinement of goals, and provided input to the rest of the studies. One of the identified challenges, namely emergent quality predictions, is addressed by RQ3 and RQ4. RQ3 aims at finding a new way to characterize and visualize the size and dynamics of scope changes in large projects. As a result, the Feature Survival Chart (FSC) concept is proposed and evaluated in Paper III. This concept

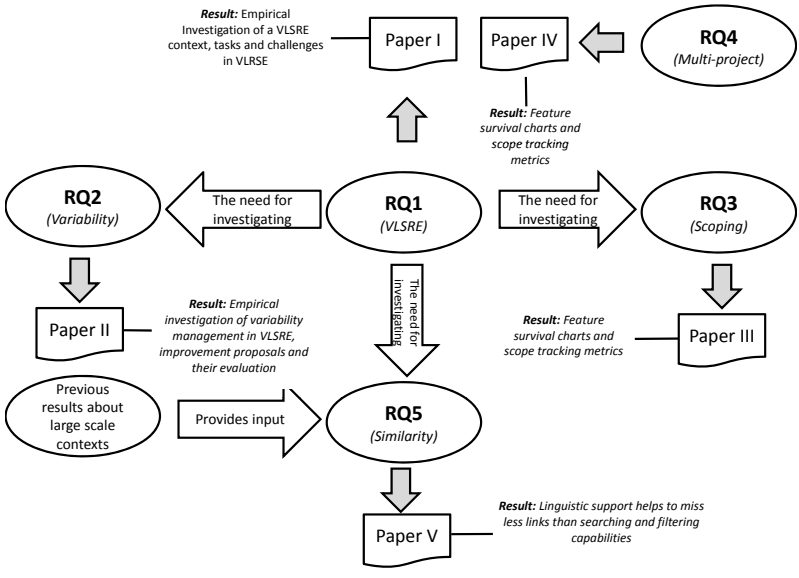

Figure 3: The relationships between challenges, research questions and papers presented in this thesis.

is later extended in Paper IV which addresses RQ4. RQ4 aims for finding optimal extensions to the in Paper III introduced visualization technique to cover multiple project scope changes. As a result, the Feature Transition Chart (FTC) concept is presented and empirically validated in Paper IV. RQ2 is addressing other challenge identified in Paper I, namely effective requirements abstractions. RQ2 provides empirical investigation of how variability is managed in practice in a large-scale software product line context. The results are reported in Paper II. The third challenge, the need to improve the tool support for coping with large-scale requirements sets, identified in Paper I and supported by the research literature, is refined to RQ5. RQ5 aims at investigating whether linguistic methods can provide a better support than searching and filtering functionalities in a task of requirements consolidation. The results from the experiment are summarized in Paper V.

The list of challenges, albeit derived from a very large company in an empirical investigation, should not be considered as the only and closed one list of issues in large-scale requirements engineering . Among other challenges not investigated in this thesis are, for example: managing requirements interdependencies (Carlshamre et al. 2001), managing quality

requirements (Berntsson Svensson 2009), resource allocation (Trautmann and Philipp 2009), cost estimation (Magazinovic and Pernstål 2008) or defining scalable requirements management processes (Berenbach et al. 2009).

3 Related Work

In this section, the background of the context of the research in this thesis is described. The research in this thesis relates to various aspects of software engineering and requirements engineering and management. The concepts that the research is mostly related to are large-scale software engineering, market-driven requirements engineering, requirements management with a special emphasis on 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 within the field 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 in one of the related publications (Paper VI), the classification of the orders of magnitude in requirements engineering is introduced and repeated here. Table 1 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

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

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.

is believed in Paper VI 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 as well as an increased complexity of requirements engineering. Furthermore, Paper VI 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 dramatically 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. 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 VI 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), can be found whether the author believes that LSRE is common industrial practice, confirmed also by Brinkkemper (2004). The belief presented in Paper VI 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 thousand of requirement that requirement engineers and managers at Siemens work with. Berenbach et al. also mention 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 (Berenbach et al. 2009). 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 requirements that may need to be managed as requirement definition nears completion. As an example (also repeated in Figure 4), Berenbach et al. (2009) give 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 re-

quirements 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.

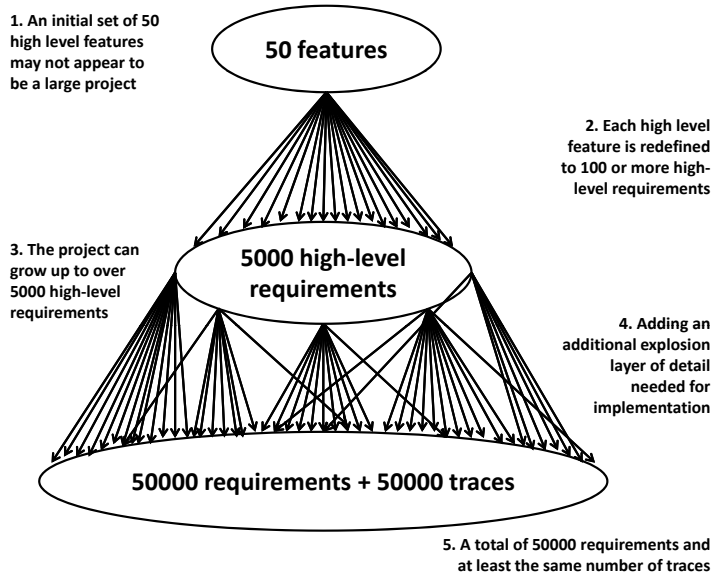

Figure 4: An example of an explosion of the number of requirements during a project (Berenbach et al. 2009).

To illustrate the complexity in VLSRE, an industrial example outlined in Paper VI 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 6 000 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, software design, 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 to the company. Table 2 gives ex-

Table 2: Examples of stakeholders that generate requirements (see Paper VI 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)
Share holders	Product management
Standardization bodies	Product management
Subcontractors and component providers	Sourcing, supply and manufacturing
	Technology and research development
	Usability engineering

amples 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), 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.

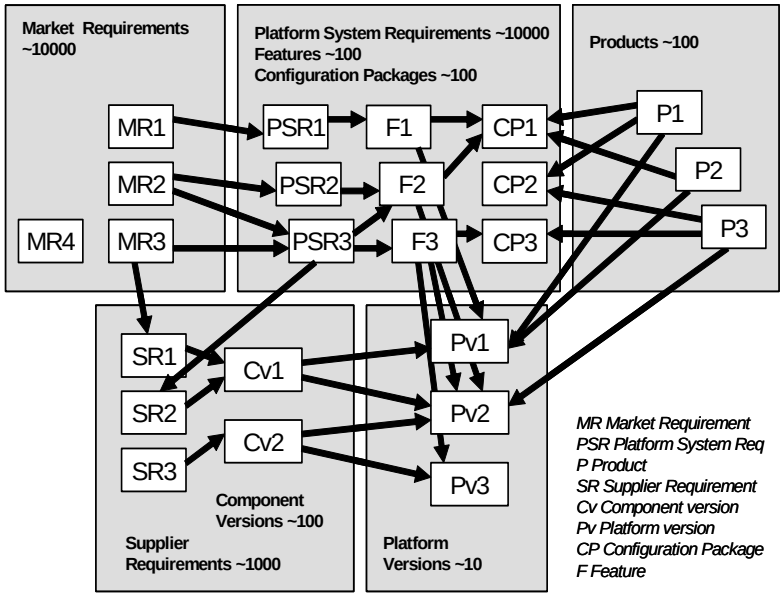

Figure 5: Orders of magnitude in different artifacts of a specific VLSRE case (see Paper VI for more details).

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. On the other hand, communication and coordination are considered by Garg (1989) as playing a central role in any large-scale cooperative effort. The article presents an

active automation base design based on "Intelligent Software Hypertext System" which: (1) supports a flexible and general purpose model for information management (hypertext) and (2) can support various software life cycle models based on an agents-tasks-products perspective. Linger et al. (2007) discuss "an emerging next-generation software engineering research area: function extraction technology". The technology is aiming to extend the possibilities of automatic computation of software components and their compositions into systems, meaning that it addresses only the source code. On the other hand, Travassos et al. (2008) present and discuss an experimentation environment that was built to support large-scale experimentation and scientific 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 as well as scientific knowledge concerned with different study types through the web. The methodological viewpoint on large-scale software engineering has also been taken by Kitchenham et al. (2007) who recommend adopting a more systematic approach to accumulating and reporting large quantities of empirical evidence. The proposed solution is to use quasi-experimental designs in improving the methodology used for performing large-scale empirical studies in software engineering.

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. One of the publications of this type is the paper by Conrad and Gall (2008), which report on lessons learned about requirements engineering in the development of large-scale systems based on eight challenges faced in a large-scale industrial project. 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 many requirements engineering experts at Siemens Corporate Research, with the general agreement that similar challenges exist across projects with similar characteristics at numerous

companies. One of the mentioned lessons learned is to establish a traceability model. Traceability management is the main research focus of Jane Cleland-Huang's et al. work (2005, 2009). Duan et al. use clustering techniques to assist in the prioritization process. The techniques are evaluated on an example of 202 requirements (Duan et al. 2009), and the other technique on an example comprising 180 requirements (Cleland-Huang et al. 2005) which places their work in the MDRE area according to Paper VI. The example from Siemens Corporate Research is also presented in the book by Berenbach et al. (2009), where the importance of requirements engineering for large projects is stressed. Berenbach et al. also present a list of the common misconceptions about requirements engineering, where one of them states that processes that 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 therefore he puts this issue on a list of industrial challenges in requirements engineering.

Another view on LSRE is presented by Bergman et al. (2002b). They investigate the political nature of requirements for large systems and argue that requirements engineering theory and practice must become more engaged with these facets. Requirements for large systems are, according to Bergman et al. (2002b), 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.

On the other hand, Ebert (2004) presents insight in techniques for pragmatically dealing with non-functional requirements. He provides examples of dealing with four types of nonfunctional requirements in large telecommunication systems, namely performance, usability, reliability and maintainability requirements. Cleland-Huang et al. (2008) propose an elicitation and prioritization process that utilizes data-mining and recommender technologies to facilitate the active involvement of many thousands of stakeholders. Their solution is claimed as scalable and capable to support elicitation and prioritization of requirements in 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 in order to stimulate the community to tackle some of the research challenges and opportunities. Among them, Boehm (2006) identifies 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 connectiv-

ity 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. For the increasingly complex "systems of systems" trend, Boehm (2006) states that traditionally, and even recently for some forms of agile methods, system and software development processes were recipes for "standalone stovepipe systems" with high risk of inadequate interoperability with other stovepipe systems. The lack of a common denominator in such collections of stovepipe systems may cause unacceptable delays, uncoordinated and conflicting plans, ineffective or dangerous decisions and inability to cope with rapid change. On the other hand, Herbsleb (2007) reflects on an increasing global connectivity trend and describes a desired future for global development together with the problems that stand in the way of achieving that vision. Herbsleb reviews research and outlines research challenges in four critical areas: software architecture, eliciting and communicating requirements, environments and tools as well as orchestrating global development. He calls large-scale software development, when teams are geographically distributed Global Software Development (GSD), and focuses on technical coordination in this context. 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. On the other hand, Damian (2007) considers 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). As another challenge, Curtis et al. mention that the need to significantly improve the ability to support the ongoing negotiation processes prevalent throughout the project lifecycle (Curtis et al. 1988, Damian and Zowhgi 2003). Furthermore, the next challenge in Global Software Development context is to understand, in a detailed way, what media are suitable for the different kinds of communication among all the business stakeholders, analysts and developers. Damian (2007) provides 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

The Software Engineering Institute (SEI), together with U.S. Department of Defense (DoD) established in 2006 a team of researchers and practitioners in order to investigate very complex systems characterized by thousand of platforms, sensors, decision nodes, weapons and war fighters connected through heterogeneous wired and wireless network. 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). Similarly to a biological ecosystem, a ULS system comprises a dynamic community of interdependent and competing organisms in a complex and changing environment. The concept of ecosystems includes complexity, decentralized control, hard-to-predict effects of certain kinds of disruptions, difficulty of monitoring and assessment, and the risks in monocultures, as well as competition with niches, robustness, survivability, adaptability, stability and health (Northrop et al. 2006). The characteristics of ULS systems outlined by Northrop et al. (2006) are as follows :

- **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. The inevitable result of this fact will be the impossibility of solving all the conflicts between stakeholders needs
- **Continuous evolution and deployment.** There will be an increasing need to integrate new capabilities into a ULS system while it is operating. The system will be evolving not in phases, but continuously
- **Heterogeneous, inconsistent, and changing elements.** A ULS system will not be constructed from uniform parts causing many misfits and incompatibilities especially in the extension phase

Today's systems are constructed based on the idea that conflicts must be addressed and resolved (Bergman et al. 2002b). The scale of ULS systems will make it impossible to resolve all conflicts and to resolve conflicts centrally. Instead, mechanisms will be proposed, or will automatically emerge to resolve conflicts locally among those who have an immediate interest in the issue. ULS systems will also encounter so-called 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, and no solution is considered to have "solved" the problem (Northrop et al. 2006).

3.2 Requirements prioritization, product management, release planning and roadmapping

Current state-of-the-art research in software engineering has established an opinion that decision processes are the driving forces to organize corporations' success (DeGregorio 1999). 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 a suite of methods and techniques (Ruhe 2003). The decision-making process has also been addressed by researchers working in the requirements engineering field, since it 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. 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. Furthermore, Leffingwell and Widrig (2003) propose 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. Finally, Wiegers (2003) presents a method that combines the customer value, penalty if the requirements is not implemented, implementation cost and risk.

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. 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. Finally, as mentioned in Section 4 and in Paper VI, 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 (see Paper VI for more details).

Releasing software to an open market brings a new potential of growth comparing to the Bespoke Software Development. In order to use this potential better, the software product development paradigm gains greater acceptance (AlBourae and Ruhe 2006). 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 standard product (van de Weerd et al. 2006a). 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 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).

Another important activity in software product management is release planning, also called release 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) and linear programming techniques using requirements interdependencies (Carlshamre 2002a). The focus of this thesis is on the scope management part of the release planning. 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 III analyzes the reasons for deciding what to remove from the scope of a project. In that paper, the reasons identified are similar to outlined by Wohlin and Aurum (2005).

Researchers have been investigating various aspects of MDRE for quite some time, 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) present 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). 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). Moreover, diagrams have an advantage of more concisely conveying the information (De Marco 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 notation conventions 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 a set of 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). While designing visual notations in software engineering, the decisions about semantics (content) seam to be treated with great care. At the same time, the visual representation issue (form) is considered a matter of aesthetics rather than effectiveness (Hitchman 2002). However, the positive influence

of the visual forms of notations on their understandability by novices has already been confirmed in a number of empirical studies (Purchase et al. 2002, Nordbotten and Crosby 2001, Masri et al. 2008). As a step towards establishing a scientific foundation for designing visual notations in software engineering, Moody (2009) provides a set of design principles in order to achieve a visually efficient notation. His analysis is focusing 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 a communication intensive phase. Therefore, it requires intensive and efficient communication among multiple stakeholders in order to agree upon the needs for a new software system or its extension. Effective visualization techniques may significantly improve this communication by unhiding the real value of communicated requirements. Following Gotel (2007), it can be stated that visualization has mainly been used to support three aspects of requirements engineering:

- Structure and relationships - visualizing the hierarchical structure of requirements documents, 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)

- 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)

- Modeling - providing a visualization of requirements specified in a formal language in order to facilitate validation activities. I* 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)

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 raising the question if it is really needed to document all this information. Since visual notations offer more dimensions to represent information than text (Tufte 1990), they are more efficient in representing the mentioned complex information structures. As an example, Gotel et al. (2007) propose 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 like 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 large-scale requirements engineering context.

Recent research efforts 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) propose 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) techniques provide new possibilities of improving requirements management related tasks, even though as mentioned by Ryan (1993) the use of these techniques have to be supervised by practitioners. These possibilities have been explored by a number of researchers and reported in a number of publications. Among those publications that include some kind of empirical evaluations, the vast majority of natural language processing techniques and tools built based on them 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) by using ambiguity rates of sentences depending on the degree of syntactic and semantic uncertainty (Macias and Pulman 1995), or detecting ambiguities by applying an inspection technique (E.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 analyzed using natural language processing techniques, Fabbrini et al. (2001) proposed a tool that is improving understandability, consistency, testability and correctness of requirements documents. On the other hand, Edwards et al. (1995) presented a tool that uses rule-based parsing to translate requirements from natural languages and by that helping requirements analysis and maintenance throughout the system life-cycle, and 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 D.Berry 1991, Goldin and Berry 1997) or synthesizing crucial requirements from a range of documents that includes standards, interview transcripts, and legal documents (Sawyer et al. 2002). On the other hand, the possibilities of applying statistical language processing techniques in requirements engineering were explored, for example by Sawyer et al. (2005) in supporting early phase requirements engineering, or in helping to identify and analyze domain abstractions (Rayson et al. 2000). Natt och Dag et al. (2004, 2005, 2006) also used statistical natural language processing methods to support the requirements consolidation process. Finally, a different approach to requirements understanding is taken by Gervasi (1999) who 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.

4 Research Methodology

The research effort in software engineering is aiming for answering questions regarding the general phenomenon of developing, maintaining and evaluating software. In the pursuit of answers to these questions certain research methods are utilized. The research presented in this thesis has mainly been conducted using an engineering approach, where situations are observed and better solutions are proposed. This pragmatic approach can also be characterized by a greater interest in obtaining practical rather than theoretical knowledge as a more valuable source of information (Easterbrook et al. 2007). The aims of the research effort summarized in this thesis are: (1) to explore, describe and provide empirical evidences towards understanding the large-scale requirements engineering phenomenon, (2) to discover what improvements are requested and (3) to decide which improvements that may be rewarding. This initial vision and research focus were used to formulate research questions, presented 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. The *fixed* research design, also called the *quantitative* research design, can be characterized by the fact that the design is finished before the data collection phase starts. Because of that, this approach can also be called "theory-driven" research design (Robson 2002). The *fixed* research design is often used to find which one of two or more proposed solutions that in average can exhibit a different behavior

than the others. The results in this case 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* approach, 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

In this section, only the strategies used in this thesis are described. 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). On the other hand, Robson argues that it is virtually impossible to cover all possible forms of inquiries, but at the same time he gives an impressive list of widely recognized research strategies (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 and Höst (2009) 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 research methodology 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. This strategy, because of its evaluative nature, serves a different purpose than other strategies. The important part of this strategy is to influence or change some aspects of whatever is in the focus of the inquiry. As pointed our by Robson, improvement and involvement are central to action research (Robson 2002). The collaboration between researchers and those who are the focus of the research and their participation in the process, is typically seen as central to action research. 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. According to Easterbrook et al. (2007), it could be argued 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 (Wieringa and Heerkens 2007).

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).

4.3 Research methods used

After selecting the research strategy, a researcher should decide which data collection techniques are the most suitable for gathering data (Easterbrook et al. 2007). Selecting research methods is a necessary and imporant part of the research methodology. Since requirements engineering, as a part of software engineering, involves real people working in real environments, researching requirements engineering is essential to study people-software practitioners as they solve real problems in real environments. This means that studies are conducted in a field setting (Lethbridge et al. 2005). Wrongly selected data collection methods may not reveal all characteristics of the data under analysis and may by that harm the analysis phase and even the results of the study. There are many data collection methods available (Robson 2002), meaning that the researcher who's goal is to select suitable techniques must perform 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. Therefore, multiple techniques can be used to overcome limitations of each single technique, for example while gathering data from multiple perspectives. This section focuses on the research methods used in this thesis, with respect to the taxonomy of techniques based on the degree of involvement of software engineers presented by Lethbridge et al. (2005). Among the first degree techniques, where software engineers were directly involved in the study, conceptual modeling and interviews 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). The interviews can be used in studies where the goal is to gain some general information, including opinions, about the process or product (Lethbridge et al. 2005). In this case, the researcher collects the mentioned information during the interview. The previous fact makes this technique flexible and inquisitive (Lethbridge et al. 2005). In spite of its time consuming nature, which is considered beeing its disadvantage, it brings the possibility to follow up answers given by participants of the study, interpret the tone of their voice, expressions and intonations, which documents or written answers cannot reveal. This technique has been used in Papers I,II,III and IV. Depending on the resources available, interviews can be used to collect small or large volumes of data. 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. The semi-structured interview uses a set of predetermined questions, but the order of how they are asked can be modified based upon the interviewer's perception of what seems most appropriate. Questions wording can be changed and explanations given, for example particular questions which seem inappropriate with a particular interviewee can be omitted, or additional ones included. In this thesis, the semi-structures interviews were performed in Papers I, II, III and IV.

Content analysis. Instead of, as previously mentioned, directly observing or interviewing for the purpose of the inquiry, content analysis technique is dealing with artifacts produced for some other purpose. It is classified as an "unobtrusive measure", which means that collecting the data 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 Pa-

pers II, III and IV. 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), which is the case in Paper III. The content analysis can also be used as a secondary or supplementary method (Robson 2002), and that was the way of using it in Papers II, III and IV.

Instrumenting systems. The prerequisite of using this techniques is to have access to the software engineer's 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 III and IV to visualize information recorded in the requirements management tool. The visualization technique provides process monitoring facilities not available by the current requirements management tool set without time commitment from requirements engineers. 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, which is a disadvantage of this method that may require, for example, a better understanding of the working environment.

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. 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 has the advantage of analyzing large amounts of data. The data is not influenced by the presence of researchers. 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 Papers III and IV, where the decision logs were studied in order to calculate one of the defined measurements.

4.4 Research classification

This section presents the classification of the research conducted in this thesis. The attempt to classify conducted research has been made with the help of the previously described research strategies and methods. Table 3 provides a mapping between the presented papers, research questions, strategies, designs and methods.

A case study research strategy is used in Paper I. The reason behind selecting this strategy is the goal of this study which was to understand

Table 3: Research Classification

Paper	Research Question	Research Design	Research Strategy	Research Method
I	*What are the challenging aspects of Requirements Management process in a very large-scale context?*	flexible	Case study	Interviews
II	*How variability is managed in practice in large-scale software product line contexts?*	flexible	Case study, action research and survey	Interviews and analysis of electronic databases of the work performed
III	*How to characterize and visualize scope changes dynamics in a large-scale software development context?*	flexible	Action research	Analysis of electronic databases of the work performed and interviews
IV	*How can multiple scope changes be characterized and visualized in a large-scale software development context?*	flexible	Action research	Analysis of electronic databases of the work performed and interviews
V	*Can linguistic methods of finding similar requirements overperform searching and filtering methods for a task of requirements consolidation?*	fixed	Experiment	Measuring the effect of manipulating independent variables on dependent variables through given tasks and data collection forms used by subjects

the large-scale requirements engineering practice in an industrial example. It can be argued here that a literature review, which most research efforts starts with, could have been used instead. However, due to a limited number of publications within the requirements engineering field that exclusively addresses issues related to the scalability of requirements management techniques, an interview study about current practices in working with large-scale requirements repositories has been performed as the introductionary step of understanding the problem area and defining further research questions. The study focuses on a senior requirements engineering role at the case of the company under study, called requirements architect, being responsible for quality and coordination of large requirements repositories. The interview method is also suitable here because of the exploratory character of this study, since it brings brings possibilities to interpret tone of the voice, expressions, and intonations of the interviewee. Finally, during the interviews researchers should explain misunderstood questions and follow up answers.

In Paper II, a case study is chosen as a research strategy. The main reason to use this strategy is the nature of the research questions that this study is addressing. The phenomenon under investigation, in this case the process of managing variability, has been studied in its natural environment (Yin 2003). Therefore, a case study strategy was used in this study. Moreover, the in-depth analysis of a single case helps to understand the surrounding context of the investigated phenomenon. One of the issues defined in Paper II, which the participants of this study reported as especially challenging, was handling the complexity in large-scale contexts in terms of multi-projects and multi-products coexisting together often as a derivation of the same code base. In this study, both direct and indirect methods of data collection were used. First, the interviews were performed to understand how variability requirements and variability points are managed in practice in software product lines, and what the problems are with large-scale variability management. Then, the analysis of the variability documentation database has been performed in order to address the third question stated in Paper II. Finally, to assess the usefulness of proposed improvements, the survey strategy has been used.

The aims for Papers III and IV were accordingly: to present a method for visualizing the scoping process in platform-based development of embedded systems complemented with scope tracking measurements (Paper III), and to extend this method by a cross-project scope changes visualization technique (Paper IV). The overall goal for both studies is to improve the understanding and provide techniques for analysis and control over scope management in a large project, where many scope changes occur and where current requirements management tools can not cope with the paste and complexity of them. Both studies utilize mixed case study and action research strategies, using multiple methods for data collection. In both Papers III and IV, researchers have been involved in several steps towards im-

plementing the visualization technique, applying the technique and scope tracking measurements 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. In Paper III, the instrumentation tool in terms of a data exporter has been built into the requirements management tool used by requirements engineers in the case company. The exported data provides input for the implementation of the visualization technique which produces visual representations of scoping decisions over time. Then, the analysis of visual representation of the work performed in the scoping process has been done. Three large product lines projects, each with hundreds of features, were analyzed in Paper III, and two in Paper IV. Furthermore, in both studies meetings with practitioners and informal interviews have been conducted while developing the visualization technique in order to collect feedback and suggestions about the solution. The more formal approaches of collecting evaluation data have been used as the final step of research in both projects. In Paper IV, interviews with practitioners have been conducted in order to discuss the results from applying the technique as well as its usefulness. The critique of the solution and suggestions for further improvements have also been collected during interviews. In Paper III, researchers collected practitioners feedback regarding the solution along the study in a continuous matter, mostly in a form of meetings and unstructured interviews. As the final, step researchers asked practitioners to rank the usefulness of proposed scope tracking measurements.

Finally, the aim of Paper V was to experimentally assess whether a tool enriched by a natural language processing functionality can provide a better assistance in a task of finding similar requirements than the searching and filtering functionalities implemented in most commercially available requirements management tools. Due to the fact that two treatments were compared in this study, the experiment research strategy has been used. The effect of the manipulation was measured on students, who played the role of practitioners in this case. The details of the design were fully prespecified before the data collection began. Finally, statistical methods have been used to assess if the hypotheses stated a priori can be rejected.

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. 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 all pos-

itivistic studies, including controlled experiments, most case studies and survey research (Easterbrook et al. 2007). In this thesis, threats to validity are presented in respect to their classification outlined by Wohlin et al. (2000).

Internal validity concerns the question about the issues that may affect the causal relationship between treatment and outcome (Wohlin et al. 2000). In experiments, the internal validity questions whether the effect is caused by independent variables or by other factors. 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 (Yin 2003). 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. 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 II, III and IV, (2) sending transcripts back to interviewees to validate the correctness of derived causal relationships in Paper I, (3) performing a replication on an experiment to show the range of conditions under which experimental results hold in Paper V or (4) performing multiple studies on the phenomenon of scope visualization in Papers III and IV.

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 repeatability of the study, such as data collection procedures. In this case, the typical criticism of a single interview case study is the fact that a follow-up interview study may produce different results, even if the same research procedures are followed. If the same study is repeated and the same results are obtained, then the study has a high degree of reliability (Yin 2003). In this thesis, 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 III, IV and the related Paper VII, and replication of an experiment in Paper V.

Construct validity is concerned with the relation between theories behind the research and the observations (Wohlin et al. 2000). 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). This type of threats is addressed in Paper III, which proposes a set of scope tracking measurements that are targeted for other case studies reuse.

External validity is related to establishing the domain 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. Due to the fact that all studies were performed in the same industrial context, the transferability of achieved results to other domains may only be addressed in a form of hypotheses.

5 Research Results

In this section, the main contributions of this thesis in relation to each research question are presented. 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.

Main contribution of RQ1. The main contribution is an increased understanding of very large-scale requirements engineering practices. The main focus in this study was put on understanding the diversity and complexity of very large- scale requirements engineering in a given industrial example. The first detailed contribution comprises a set of tasks related to a role at the case company called "requirements architect", which is working with large and complex requirements repositories. The second contribution of Paper I is the practitioners' views on the notion of "requirements architecture" and its desired quality attributes. This study contributes also by presenting the list of further research opportunities in VLSRE that has been addressed in RQ2, RQ3, RQ4 and RQ5. To summarize, Paper I can be considered as an exploratory study where more specific issues and challenges were defined.

Main validity issues of RQ1. One major treat to the validity in this case is the number of companies involved in this study. It is obviously risky to draw more general conclusions from a single-company case study in terms of issues and challenges, as they may be company- or even domain-specific instances. However, due to the lack of publications that characterize the large-scale requirements management phenomenon when the study was performed, the focus of this study was to explore this phenomenon on a given case company example and to report findings with the respect to the case study context. In this way, no attempt to present the findings as generally applicable for all large-scale requirements management contexts has been made. However, it is planned to extend the initial study by inviting more companies from other domains so that the initial results can

be confirmed or rejected in the light of new evidences. The second threat related to construct validity is the way how subjects were selected to be interviewed. This treat is addressed in two ways: firstly by inviting all requirements architects from one department of the company responsible for a certain number of products dedicated to a specific market segment, and secondly by asking practitioners for recommendations of which persons to interview so that the risk of getting a subjective picture can be minimized. The final major threat to validity in this study is related to the list of the quality attributes of a good requirements architecture. Although the reported quality attributes may be considered more general, the confirmation of derived results requires additional interview studies (see Section 6 for more details).

Main contribution of RQ2. The contributions of RQ2 are threefold: (1) the results from an empirical interview study, (2) the improvement proposals of managing variability at the case company and (3) their evaluation. The first research contribution is the result from an interview study, were the processes of product derivation (Deelstra et al. 2000) and the concept of managed variability (Pohl et al. 2005) were investigated. During the interview study, 29 persons working with requirements engineering, implementation and testing were interviewed in order to understand how the variability is represented, implemented, specified and bound during the product configuration. The results contribute in an improved understanding of large-scale variability management for software product lines in the mobile handset domain which is addressing RQ1 in Paper II. The second detailed result that contributes to RQ2 is a set of challenges in managing variability in large-scale software product lines projects. The last result that comprises to RQ2 is the improvement proposal to the current way of working and its evaluation. The proposal includes a new structure of variability information that aims for enabling linking product configuration to the initial requirements by splitting the configuration into two levels of granularity. The proposal has been empirically tested by applying it to the existing configuration structure in a pilot study and performing a survey about its potential benefits and drawbacks.

Main validity issues of RQ2. One major threat to validity is the research strategy utilized in this study. The case study strategy is often criticized for offering poor basics for generalizing (Yin 2003). However, the focus of this study is to increase the understanding of large-scale variability management in a given large-scale company example, and not to draw general conclusions that certain issues and experiences in this case also will be present in other cases, especially in other domains. The second threat to validity is the way and number of persons that were asked to participate in the interview study. This threat to construct validity is always problematic in case study interview research, since the "subjective" judgments are used to collect the data. This treat is addressed in two ways. Firstly, researchers asked practitioners for recommendations to which persons to interview so

that the risk of getting a subjective picture decreased. Secondly, to cover the entire process of variability, researchers invited representatives from all groups of specialists that were involved in the process of variability management including both management and development sites. A more detailed analysis of threats to validity is presented in Paper II.

Main contribution of RQ3 and RQ4. The main contributions of RQ3 and RQ4 are visualization techniques for showing scope changes over time in a project or across projects. These contributions are addressing one of the challenges defined in Paper I, namely analysis methodology and visualization models for large-scale requirements management. This issue is investigated in Papers III and IV. Firstly, in Paper III a technique called Feature Survival Charts (FSC) for visualization of scoping change dynamics is implemented and evaluated in three projects. The results of this empirical evaluation demonstrate that the charts can effectively help in investigating reasons behind scoping decisions. Furthermore, Paper III contributes to RQ3 by providing a set of scoping measurements, theoretically analyzed and applied to the empirical data given by the case company. Finally, the last contribution in Paper III is the integration of the visualization techniques with the current requirements management analysis and measurement practices. The main contribution of Paper IV is a visualization technique called Feature Transition Charts (FTC) that gives an overview of scoping decisions involving changes across multiple projects. The technique is an extension of the in Paper III presented FSC concept. FTC is initially validated using industrial data from the embedded systems domain in a multi-project product line engineering context in dialogue with practitioners. The additional contribution of Paper IV, which addresses RQ4, is the results of the analysis of the symbols that can be used in providing an effective overview of the timing and magnitude of feature transitions.

Main validity issues of RQ3 and RQ4. In the two scope visualization studies in Papers III and IV, the major threat is concerned with external validity. The visualization techniques, both feature survival charts and feature transition charts, have been designed with the requirements management process of the case company in mind. This fact is especially important for the feature transition charts, since the company utilizes the concept of software product lines where many consecutive releases of the common code base coexist (Pohl et al. 2005). However, the author believes that the in Papers III and IV presented visualization techniques can provide means of describing complex scoping processes also for other software management and requirements management process models. The second major threat to the external validity is concerning the general application of both FSC and FTC. Both concepts have been tested on the 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 and rapidly changing datasets, the questions regarding this

threat can be limited to details of presented visual techniques rather than their general usefulness. The decisions on which projects the visualization techniques should be tested on were made together with practitioners, minimizing the the construct validity threat. Finally, the solution has been 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.

Main contribution of RQ5. The last contribution of this thesis is the results of an experiment performed to assess if a linguistic method for finding similar requirements can over-perform searching and filtering methods. The experiment was a replicated experiment. The results from the original experiment are confirmed in five out of six tested hypotheses. The second contribution of this study is the result of the cross-experimental hypotheses testing. The final contribution is the discussion of the reasons behind results discrepancy.

Main validity issues of RQ5. The first main threat, related to the 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 big set of requirements, which often is the case in industrial settings. The second main threat is related to the conclusion validity, since on the contrary to the original study, where subjects worked independently in the replication case were asked to form pairs. This threat has been 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 interface 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 fourth main threat, the compensatory rivalry, may be a problem in this case, since the group that used the 'open-source' solution or the commercial solution may try to perform much better to make their favor type of software win. This threat is addressed by explicitly stating in the beginning of the experiment that there is no favor or assumingly better method. Finally, the last main threat to construct validity 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 has been 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.

Table 4: Further research plans and ideas.

Further Research	Description	Research Approach
FR1	Interview study with more companies involved	Interviews, content analysis and surveys
FR2	Additional empirical studies of variability management in large-scale contexts	Interview and document studies for data collection
FR3	Extending the proposed visualization techniques on the system requirements level visualization. Improving the user interaction. Additional empirical evaluations.	Building a tool support that can be reused in other companies.
FR4	Additional investigations of possible usage of linguistic and machine learning tool support for Requirements Management related tasks	Literature study and prototype tools for data collection.

6 Further Research

In this section, a research plan for the future is presented. 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 and to supporting some aspects with new methods or tools. The important aspect of further research effort is to provide results that can demonstrate scalability to large or very-large scale requirements engineering contexts. Therefore, further efforts are required to provide scalable methods and tools to assist requirements management in the mentioned scale context. Furthermore, it is also important to continue the empirical investigation of VLSRE contexts to more precisely understand their nature. These goals are planned to be realized through more empirical studies with both quantitative and qualitative approaches. A more detailed plan for further research is presented below, while being summarized in Table 4.

FR1. Further empirical investigations of large-scale requirements engineering contexts. The interview study presented in Paper I presents a single company example of tasks related to large-scale requirements management and coordination. This purposive sampling strategy (Easterbrook

et al. 2007) hinders drawing more general conclusions from the study. Thus, in order to make a possible distinction between an extreme and typical nature of results derived in Paper I, a multiple case follow-up study is planned to be conducted to increase the validity. In the follow-up interview study, researchers are planning to assess, focusing on the tasks, whether issues and challenges derived from the original study are caused by the size factor or not. This approach can hopefully provide an in-depth understanding of what can be considered as large-scale specific tasks, issues or challenges. The results are also planned to be confronted with recent publications touching upon challenges in large and very large-scale requirements engineering contexts (Berenbach et al. 2009, Konrad and Gall 2008, Northrop et al. 2006, Herbsleb 2007, Bergman et al. 2002a, Boehm 2006). Additionally, the in Paper VI and in Section 3.1 presented orders of magnitude classification in requirements engineering are planned to be extended to achieve a taxonomy of RE techniques and approaches that were successfully applied in practice or provided a strong scientific evidence to cope with the size and complexity of VLSRE.

As a part of further research within this area, it is also planned to continue conceptual and empirical investigation of the notion of requirements architectures. The author believes that a better understanding of the role of requirements architectures in large-scale requirements management can help in tackling problems of this context. That is also why the additional investigation of the organizational and process aspects in relation to the requirements architecture is planned for further research. Furthermore, the research plan includes investigating features of computer-aided tools for managing requirements architectures and also visualizing these architectures in large-scale product line engineering. Finally, it is planned to develop assessment instruments for requirements architecture quality and competence certification of requirements architects, but only after a better understanding of the notion of the requirements architecture quality.

FR2: Additional empirical studies of variability management in large-scale contexts. As provided in Paper II, the inevitable cost for a greater degree of reuse and increased productivity of Software Product Lines (SPL) is an increased complexity of coexisting product variants and an increased cost of managing them (Pohl et al. 2005). Thus, more efficient methods and tools for specifying and managing variability are aims for further research. It is planned to conduct additional empirical studies at other companies. In these planned studies, the already derived results can be compared and confronted with new cases in terms of way of working, issues, challenges and research opportunities. Furthermore, additional studies can enable possibilities to evaluate in Paper II proposed improvements and to assess their transferability to other empirical environments.

FR3: Extending the proposed visualization techniques on the system requirements level visualization. Improving the user interaction. Additional empirical evaluations. It is rather clear that 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, which usually is impeded by the need to browse through disjoint textual requirements documentation and accompanying models (Gotel et al. 2007). The visualizations presented in Papers III and IV 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. In parallel, it is planned to focus the further research steps on an enhanced tool support that utilizes various zooming algorithms together with extended interaction capabilities. 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 explanations for complementary aspects of scoping not mentioned by the current visualization technique and for other requirements management related tasks are planned as further research topics.

FR4: Additional investigations of possible usage of a linguistic tool support for requirements management related tasks. Paper V presents the results from an evaluation of linguistic support for identification of similar requirements. The natural language processing field can provide a vast number of other techniques that automatically can analyze natural language documents. Therefore, it is planned to test other techniques for other relevant tasks in large-scale requirements management. In particular, the further research should focus on unsupervised natural language processing methods, for example clustering (Duan et al. 2009) or searching methods, that may provide valuable help with impact analysis or traceability.

References

M. Abramovici and O. J. Sieg. Status development trends of product lifecycle management systems. In *Proceedings of International Conference Integrated Product and Process Development*, pages 55–70, 2002.

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

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

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

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

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

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

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

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

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

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

K. Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 2000.

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

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

M. Bergman, J. L. King, and K. Lyytinen. Large scale requirements analysis as heterogeneous engineering. *Scandinavian Journal of Information Systems*, 14(4):37–55, 2002a.

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

R. Berntsson Svensson. Managing quality requirements in software product development. *Licentiate Thesis*, April 2009. ISSN 1652-4691.

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

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

S. Biffl, B. Thurnher, G. Goluch, D. Winkler, W. Aigner, and S. Miksch. 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)*, pages 437–446, 2005.

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

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

R. Booth, B. Regnell, A. Aurum, R. Jeffrey, and J. Natt och Dag. 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)*, pages 41–47, 2001.

S. Brinkkemper. 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)*, pages 41–54, 2004.

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

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

P. Carlshamre and B. Regnell. 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*, pages 961–965, 2000.

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

D.W. Carman, A.A. Dolinsky, M.R. Lyu, and J.S. Yu. Software reliability engineering study of a large-scale telelcommunications software system. In *Proceedings of the International Symposium on Software Reliability Engineering*, pages 350–359, 1995.

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

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

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

J. Cleland-Huang and B. Mobasher. 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*, pages 3–6, 2008.

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

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

Ward Cunningham. Manifesto for agile software development. `http://agilemanifesto.org/`, December 2001.

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

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

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

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

T. De Marco. *Structured Analysis and System Specification*. Yourdon Press, 1978.

S. Deelstra, M. Sinnena, and J. Bosch. Product deriviation in software product families: a case study. *The Journal of Systems and Software*, 74(2):173–194, 2000.

G. DeGregorio. Visual tool support for configuring and understanding software product lines. In *Proceedings of the 9th International Symposium of the International Council on System Engineering*, pages 1–7, 1999.

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

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

S. M. Easterbrook, J. Singer, M. Storey, and D. Damian. *Guide to Advanced Empirical Software Engineering*, chapter Selecting Empirical Methods for Software Engineering Research, pages 285–311. Springer, 2007.

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

M. L. Edwards, M. Flanzer, M. Terry, and J. Landa. 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*, pages 278–281, 1995.

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

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

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

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

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

M. S. Feather, S. L. Cornford, J. D. Kiper, and T. Menzies. 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)*, pages 80–89, 2006.

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

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

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

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

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

L. Goldin and D. M. Berry. Abstfinder, a prototype natural language text abstraction finder for use in requirements elicitation. *Automated Software Engineering*, pages 375–412, 1997.

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

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

O. C.Z. Gotel, F. T. Marchese, and S.J. Morris. On requirements visualization. In *Proceedings of the Second International Workshop on Requirements Engineering Visualization (REV 2007)*, pages 80–89, 2007.

O. C.Z. Gotel, F. T. Marchese, and S.J. Morris. The potential for synergy between information visualization and software engineering visualization. In *Proceedings of the 12th International Conference Information Visualisation*, pages 547–552, 2008.

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

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

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

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

E. Hornecker and J. Buur. 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*, pages 437–446, 2006.

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

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

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

J. Karlsson. *A Systematic Approach for Prioritizing Software Requirements. Doctorial Dissertation,*. PhD thesis, Linköping University, Sweden, 1998.

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

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

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

Lena Karlsson, Åsa G. Dahlstedt, Johan Natt Och Dag, Björn Regnell, and Anne Persson. 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)*, 2002.

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

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

C. Knight and M. Munro. Virtual but visible software. In *Proceedings of the IEEE International Conference on Information Visualization*, pages 198–205, 2000.

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

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

R. Koschke. 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, 2003.

G. Kotonya and I. Sommerville. *Requirements Engineering*. John Wiley & Sons, 1998.

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

M. D. Lee, R. E. Reilly, and M. A. Butavicius. An empirical evaluation of chernoff faces, star glyphs and spatial visualization for binary data. In *Proceedings of the Australian Symposium on Information Visualization*, pages 1–10, 2003.

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

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

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

F. J. Linden, K. van der Schmid, and E. Rommes. *Software Product Lines in Action The Best Industrial Practice in Product Line Engineering*. Springer-Verlag, 2007.

R. C. Linger, M. G. Pleszkoch, L. Burns, A. Hevner, and G. H. Walton. 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)*, pages 9–17, 2007.

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

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

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

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

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

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

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

J. Natt och Dag, V. Gervasi, S. Brinkkemper, and B. Regnell. 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)*, pages 283–294, 2004.

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

J. Natt och Dag, T. Thelin, and B. Regnell. 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, 2006.

P. Naur and B. Randell. Software engineering: Report of a conference sponsored by the nato science committee. Technical report, NATO Scientific Affairs Division, 1968.

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

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

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

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

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

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

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

S.L. Pfleeger. *Software Engineering – Theory and practice*. Prentice–Hall, 2001.

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

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

C. Potts. 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)*, pages 128–130, 1995.

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

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

T. A. Kappel R. E. Albright. Roadmapping in the corporation. *Research-Technology Management*, 46(1):31–40, 2003.

P. Rayson, L. Emmet, R. Garside, and P. Sawyer. 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*, pages 288–300, 2000.

B. Regnell and S. Brinkkemper. *Engineering and Managing Software Requirements*, chapter Market–Driven Requirements Engineering for Software Products, pages 287–308. Springer, 2005.

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

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

B. Regnell, B. Ljungquist, T. Thelin, and L. Karlsson. 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*, pages 89–93, 2004.

B. Regnell, H. O. Olsson, and S. Mossberg. Assessing requirements compliance scenarios in system platform subcontracting. In *Proceedings of the 7th International Conference on Product Focused Software Process Improvement*, pages 362–376, 2006.

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

C. Robson. *Real World Research*. Blackwell Publishing, 2002.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

J. Singer, S. E. Sim, and T .C. Lethbridge. *Guide to Advanced Empirical Software Engineering*, chapter Software Engineering Data Collection for Field Studies, pages 9–34. Springer, 2007.

I. Sommerville. *Software Engineering*. Addison–Wesley, 2007.

M. Svahnberg. *Supporting Software Architecture Evolution - Architecture Selection and Variability*. PhD thesis, Blekinge Institute of Technology, 2003.

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

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

N. Trautmann and B. Philipp. 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)*, pages 1143–1148, 2009.

G. H. Travassos, P. S. M. dos Santos, P. G. M. Neto, and J. Biolchini. 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)*, pages 193–202, 2008.

E. Tufte. *Envisioning Information*. Graphics Press LLC, 1990.

UML. The unified modeling language webpage. `http://www.uml.org`, January 2010.

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

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

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

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

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

S. Vasile, P. Bourque, and A. Abran. Visualization - a key concept for multidimensional performance modeling in software engineering management. In *Proceeding of the S2006 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR 2006)*, pages 334–339, 2006.

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

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

C. Wohlin and A. Aurum. 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)*, pages 246–255, 2005.

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

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

R.K. Yin. *Case Study Research: Design and Methods*. Sage Publications, 2003.

A. Zenebe and A. F. Norcio. 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*, pages 7–12, 2007.

Paper I

Architecting and Coordinating Thousands of Requirements - An Industrial Case Study

Krzysztof Wnuk[2], Björn Regnell[1,2], Claes Schrewelius[1]
[1]Sony Ericsson Mobile Communication, Lund, Sweden
{claes.schrewelius,bjorn.regnell}@sonyericsson.com
[2]Dept. of Computer Science, Lund University, Sweden
{krzysztof.wnuk,bjorn.regnell}@cs.lth.se

In Proceedings of the 15th International Working conference on Requirements Engineering: Foundation for Software Quality (REFSQ09), June 2009, Amsterdam, The Netherlands

ABSTRACT

[Context and motivation] When large organizations develop systems for large markets, the size and complexity of the work artefacts of requirements engineering impose critical challenges. **[Problem]** This paper presents an industrial case study with the goal to increase our understanding of large-scale requirements engineering practice. We focus on a senior requirements engineering role at our case company, called requirements architect, responsible for quality and coordination of large requirements repositories. **[Results]** Based on interviews with 7 requirements architects, we present their tasks and views on architecture quality. **[Contribution]** Our results imply further research opportunities in large-scale requirements engineering.

Keywords: Large-scale requirements engineering; Empirical study; Requirements repositories; Requirements dependencies; Requirements architect

1 Introduction

Large software companies are often confronted with large and complex requirements repositories. The requirements originate from multiple sources and address multiple customers and market segments. This paper presents an industrial case study of the tasks involved in managing large and complex requirements repositories. The investigated tasks are related to a role called Requirements Architect that has recently been introduced at the case company. The requirements architects are responsible for the scope of large platform projects that products are based on (Pohl et al. 2005). Our motivation to perform this study was to understand current practices in working with large-scale requirements repositories and to find issues for future research. In this study, we have conducted interviews with requirements architects in order to address the following questions: (1) What are the tasks related to working with large-scale complex requirements repositories on multiple products platform projects? (2) How do practitioners perceive the notion of requirements architecture and how do they describe good requirements architectures?

The second question is related to sustainable requirements architectures (Regnell et al. 2008). With the term requirements architecture we mean the underlying structure of requirements, including the data model of requirements with their preconceived and emerging at-tributes and relations. By sustainable architectures we mean structures that allow for controlled growth while allowing requirements engineers to keep track of the myriad of issues that continuously emerge. Practitioners facing a transformation to large-scale requirements engineering (RE) may use this research to gain insights in what may come, and researchers may use the results to inform their choices of future re-search directions.

The paper is organized as follows: Section 2 describes the industrial context at the case company. Section 3 provides the methodology description. Section 4 and 5 highlights the result of interviews. Section 6 concludes the paper.

2 Industrial case context

The interview study was performed at Sony Ericsson. Due to the technological complexity of the domain, the case company is working in parallel in many advanced system engineering areas such as radio technology, audio and video, and positioning. The complexity of requirements engineering is driven by a large and diverse set of stakeholders, both external to the company and internal. Different stakeholders have different demands on the future functionality of the mobile phone which they express by different types of requirements. Requirements originating from external stakeholders are called market requirements. They are mainly supplied by

mobile operators, which usually submit specifications with thousands of requirements that require gap analysis. Other sources of requirements are the Application Planning and Product Planning departments. The platform and market requirements also have to be checked against supplier requirements to ensure that certain functionality can be delivered by a corresponding platform project, including integration of subcontracted parts. Currently, the case company's requirements database contains around 30 000 platform system requirements and a few thousands supplier requirements. The platform system requirements are organized into features that represent the smallest units that can be scoped into or out from the platform project (Clements 2002). The case company develops products using a product line engineering approach, where one platform project is the basis for many products that reuse the platform project's functionality and qualities (Pohl et al. 2005). Within the platform project, the case company has defined a number of requirements engineer groups called Technical Working Groups (TWGs). They are responsible for elicitation, specification and prioritisation of high-level requirements within a specific sub-domain. Within this industrial context, requirements architects work mainly with platform system requirements and features. Their main responsibility is the management of the scope of platform projects by helping TWGs to specify requirements and project management to see all implications of the scoping decisions. The scoping decisions are made by a Change Control Board (CCB).

3 Research Methodology

To study individual perceptions of requirements architect role at the case company, we conducted seven semi-structured interviews (Robson 2002). Before conducting interviews, a brainstorming and planning meeting was conducted. During this meeting, the scope of the study was agreed upon and an interview instrument was developed with a set of questions, where the wording could be changed and the order could be modified based upon the interviewer's perception (Robson 2002). The third author, acting in his role as manager for requirements architects at the case company, participated in the development of the interview instrument and invited seven interviewees with various experience within the requirements architect role. These persons were chosen from three sub-organizations within the case company, each responsible for products for different market segments. It was sent out via email to all the participants in advance and also discussed at the beginning of each interview to ensure that the scope of the interview was understandable. The interviews were held during the autumn of 2007 and varied in length between 60 and 110 minutes. All interviews were attended by two interviewers and one interviewee. Questions were kept simple and effort was put on avoiding leading or biased questions

(Robson 2002). All interviews were transcribed. After transcription, each of the interviewees received the transcripts for validation. Interviewees analysed their transcripts in order to ensure that the interviewers heard and understood the recordings and notes correctly. In case of misinterpretations, corrections and comments were sent back to the researchers. The data was then imported to a spreadsheet program to perform a content analysis (Patton 2002) based on categorisation. The categories such as tasks or notion of requirements architecture quality, were chosen based on the interview instrument topics and other emerging topics in the interviews. Additionally, for each category notes describing problems and improvements were added. Finally, the results were validated by two interviewees that gave independent comments to the proposal of the tasks derived from the inter-views.

4 Tasks of the Requirements Architect in the case company

Based on the analysis of interviews, we have identified six tasks, listed in Table 1.1, that represent what is considered to be important obligations of the requirements architect role when acting as a senior coordinator in a large-scale setting. Several tasks (T1, T4, and T5) are directly related to change management. In order to cope with the initial definition of the platform projects scope and later incoming change proposals to the platform projects, requirements architects facilitate communication across different groups of requirements engineers. This may indicate that the complexity in both requirements inter-dependencies and organisational structure in the large-scale case imply hard challenges in communicating decisions about changes. The analysis of gaps between market requirements and what is offered by technology suppliers (T2) is increasingly complicated as the number of stakeholders on the market increases and the number of technical areas that are covered gets larger.

Also, for a basic and common task such as checking the quality of requirements (T3), interviewees express challenges related to the cohesion of complex multilayered requirements structures that originate from multiple sources. In our case, requirements architects have to drive complex changes (T4) that span over many technical areas and may impact many product releases in one platform. Another challenge related to these investigations is the ability to ensure that investigations are made by the right persons with the right competence and that the full impact picture will be ready be-fore CCB decision meetings. Missing some of the aspect may have a great impact on the whole platform project. In a large scale case, the task of presenting the current scope (T5) is especially demanding as the requirements architect must understand both technical aspects as well as the business and market impact of all features in order to conclude

Table 1.1: The tasks and goals for requirements architect in the case company

Task	Goal
T1: Scope management	Ensure that the platform project scope changes are ad-dressed and that the change proposals are prepared.
T2: Gap analysis	Ensure that misalignments between market requirements and supplier requirements are addressed.
T3: Enforce requirements quality improvements	Check the quality of requirements. Alert if requirements quality improvements are needed.
T4: Drive CCB investigations	Drive change proposal investigations in order to gain understanding of the impact of the scope changes.
T5: Present the scope	Present the scope of the platform project at milestones.
T6: Request requirements architectures improvements	Ensure that the requirements structure is maintained ac-cording to defined rules.

them in a way that is meaningful to high-level management and marketing. Finally, we report that in a case like the one we have examined, where several parallel large platform projects coexist, there is an expressed need for a per-son with a holistic view that has a mandate to request requirements architecture im-provements (T6). In this case, the responsibility for ensuring architectural consistency of requirements is not delegated to the projects, but is managed across projects by requirements architects.

5 Views on requirements architecture and its quality

In our interviews with practitioners we have confirm our preunderstanding that the concept of requirements architecture is complex and include many aspects. We have deliberately not imposed a preconceived, closed definition of the concept on our interviewees, as we wanted to base our understanding of the requirements architecture on empirical data. We cannot say that a single, generally accepted definition of requirements architecture has emerged, but our findings indicate that all interviewed practitioners included some of the following aspects in their views on requirements architecture: (1) the requirements entities themselves (such as features, system requirements, detailed requirements, functional requirements, quality requirements, etc) and their relationships; (2) the information structure (meta-model) of requirements entities including (a) attribute types of entities, and (b) the relationship types including different types of dependencies to other entities; (3) the evolution of the information structure (a) over time and (b) across abstraction levels as entities are refined both bottom-up and top-down; (4) the implications of organisational structures on requirements structures; (5) the implications of process and methodology on requirements structures; (6) the implementation of tool support and its relation to requirements structures, organisation, process, methodology etc.; (7) the scalability of the requirements structures as the number of entities increase and the interrelated set of entities gets more complex.

In our interviews with requirements architects, we also discussed the notion of quality of requirements architectures. We started the discussion based on the analogy of how system architecture quality supports good design and implementation of systems, and transferred this analogy to how requirements architecture quality sup-ports good requirements engineering. The following quality issues were identified when analysing interview transcripts:

Understandability and cohesion. Responders expressed the opinion that a good requirements architecture should be easy to understand and designed to enable a holistic view of different types of modules and abstraction levels in order to enable easy identification of vital information. Furthermore, the way how the structure of requirements information is

visualized was also mentioned by our responders as an important factor influencing mentioned quality issues.

Robustness, integrity and enforcement of policies. An established process for managing and architecting requirements can result in a consistent, reliable and robust requirements architecture. Lack of clear policies and working rules may result in low reliability of requirements as well as discrepancies in usage of the architectural policies across projects.

Extensibility, flexibility and efficient traceability. According to our responders, a good requirements architecture should allow for controlled growth by being extensible and flexible without endangering the previously mentioned qualities of robustness and integrity. Cost-efficient traceability among requirements at different levels of abstraction when continuous growth and refinement occur is important. A good balance between extensibility, flexibility and traceability on one hand and the complexity driven by these qualities on the other hand has to be achieved in order to avoid the risk of ending up with an unmanageable repository.

6 Conclusions

This paper presents tasks related to a role called requirements architect, which is working with large and complex requirements repositories at the case company. We also present practitioners views on quality attributes of the artefact called requirements architecture. Efficient management of large sets of information is considered to be crucial in many disciplines. Similar to software architecture, the information model is considered to be not only a technical blueprint for a software-intensive system, but it also includes social, organisational, managerial and business aspects of the software architecture (Bass et al. 2003). At our case company, the requirements architecture is an artefact that is managed separately, but in relation to the system architecture, and interviewees express a range of issues that need to be addressed, both soft issues such as organisation and business models as well as technical aspects.

The requirements architect role at our case company is motivated by a perceived need of special attention to cross-cutting issues, and interdisciplinary communication across sub-domains and technical areas. We found several tasks of normal requirements engineering practice, such as change management, scoping and specification quality enforcement that is viewed as particularly challenging in the studied large-scale setting, and therefore included in the responsibilities of requirements architects acting as senior coordinators of the requirements engineering process. We also found expressions for specific quality aspects of the requirements architecture itself that are viewed as important to support an effective and efficient management of an increasingly large and complex repository.

In relation to the concept of requirements architecture, we highlight the

following areas to be considered in further research:

- Continued conceptual and empirical investigation of the notion of requirements architecture
- Investigations on features of computer-aided tools for managing requirements architectures
- Studies of the organisational and process aspects in relation to requirements architectures
- Development of assessment instruments for requirements architecture quality and competence certification of requirements architects
- Analysis methodology and visualisation models for requirements architectures in large-scale product line engineering

Acknowledgments

This work is supported by VINNOVA (Swedish Agency for Innovation Systems) within the UPITER project. Special thanks to the anonymous interviewees for their valuable time and knowledge. Thanks also to Thomas Olsson and Lena Karlsson for the initial input on a draft version of this paper, and to Lars Nilsson for valuable language comments

References

L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice Second Edition*. AddisonWesley, 2003.

P. Clements. Being proactive pays off. *IEEE Software*, 19(4):28-30, 2002.

M. Q. Patton. *Qualitative Research & Evaluation Methods*. Sage Publication Ltd, 2002.

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

B. Regnell, R. Berntsson Svensson, and K. Wnuk. Can we beat the complexity of very large-scale requirements engineering? In *Lecture Notes in Computer Science*, volume 5025, pages 123-128, 2008.

C. Robson. *Real World Research.* Blackwell Publishing, 2002.

Paper II

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
Dept. of Computer Science, Lund University, Sweden
{krzysztof.wnuk,bjorn.regnell}@cs.lth.se

In Proceedings of the Third International Workshop on Variability Modelling of Software-intensive Systems(VaMoS2009), January 2009, Sevilla, Spain

ABSTRACT

Efficient variability management is a key issue in large-scale product line engineering, where products with different propositions are built on a common platform. Variability management implies challenges both on requirements engineering and configuration management. This paper presents findings from an improvement effort in an industrial case study including the following contributions: problem statements based on an interview study of current practice, an improvement proposal that addresses the challenges found, and an initial validation of the proposal based on interviews with experts from the case company.

1 Introduction

Software Product Lines have already proven to be a successful approach in providing a strategic reuse of assets within an organization (Pohl et al. 2005). In this context, variability management is considered as one of the key for successful product lines and concerns in all phases of the software product line lifecycle (Bosch et al. 2002). We experience considerable growth of the amount of variability that has to be managed and supported in software assets. Inspired by the previous fact, we have conducted an industrial case study focusing on the process of variability management at one of our industrial partners in the mobile phone domain. The topic of our investigation was an established product line engineering process (Pohl et al. 2005) in a company that sells over 50 products every year worldwide in millions of exemplars. Our goal for this study is to increase the knowledge of how the products are configured by studying current issues and if possible proposing and evaluating improvements. To address the goal we have formulated three research questions:

- **Q1:** How are variability requirements and variability points managed in software product lines in practice?
- **Q2:** What are the problems with managing variability requirements and product derivation?
- **Q3:**What improvements can be made in managing variability?

The first two questions were addressed by an interview study, were we have investigated the process of product derivation (Deelstra et al. 2000) and the concept of managed variability (Pohl et al. 2005). By using managed variability we refer to defining and exploiting variability throughout the different life cycle stages of a software product line (Pohl et al. 2005). In total 29 persons working with requirements engineering, implementation and testing were interviewed in order to understand how the variability is represented, implemented, specified and bound during the product configuration. As a result, a set of challenges is defined and presented in this paper.

To address Q3, we have proposed and evaluated improvements to the current way of working. Our main proposal includes a new structure of variability information that aims at enable linking product configuration to the initial requirements. It includes splitting the configuration into two levels of granularity. Additionally, we propose to use a main product specification with entities that can be consistently applied throughout the whole organization and will address current documentation issues.

Finally, we have empirically evaluated our improvement proposals by applying them to the existing configuration structure in a pilot study. Additionally, we have conducted a survey by sending questionnaires about the potential benefits and drawbacks of our proposal. 28 out of 34 persons

have answered our questionnaire. Most of the respondents expressed positive opinions about the proposal and did not express any major obstacles that may apply to it.

The reminder of this paper is organized as follows. In Section 2, we describe the industrial context of the case study. In Section 3, we provide a description of research methodology. In Section 4, we discuss identified problems and issues. In Section 5, we describe improvement proposals, which we evaluate in Section 6. Section 7 presents related work and the paper is concluded in Section 8.

2 Industrial Context

The case study was performed at the company that has more than 5 000 employees and develops embedded systems for a global market. The company is using a product line approach (Pohl et al. 2005). Each product line covers different technologies and markets. The variability of the software product lines in our case are organized in two dimensions. The first dimension represents product segments or product technologies, and the second represents the code base that evolves over time. In each of the clusters there is one lead product built from the platform representing most of the platform functionality. The lead product is scaled down to create sub-products and new variants for other markets and customers. Some of the sub-products originating from the main product contain new features (Pohl et al. 2005). The platform development process is separated from the product development process as described by Deelstra et al. (2000).

Organization. There are three groups of specialists working with the requirements part of the platform project: *Requirements Engineers, Requirements Coordinators* and *Product Requirements Coordinators*. Each technical area in the products domain has a requirements engineers group responsible for covering the progress in the focused field. Their involvement in the projects is mainly focused on the platform project where they supply high level requirements derived from roadmaps, product concepts and customer requirements. They are also main responsible for the scoping process of the platform. Requirements coordinators work between requirements engineers and developers. Their main role is to communicate requirements to the developers and assist with creating detailed design documents and requirements. Product requirements coordinators are responsible for the communication of the requirements between the product planner and requirements engineers on the specific product level.

The *Development Teams* are responsible for implementing the software in the platform. They review the requirements and estimate the effort needed for implementation. Each new functionality is assigned to a primary development team which is responsible for its implementation in the software modules. Newly implemented functionality is later tested before final de-

livery to the platform. The different modules need to be integrated and compiled to a full system. This stage is done by the *Product Configuration Managers (PCMs)* team which manages the different variants and versions of the products created from the platform. The compiled system is tested by a product focused testing organization, *Product Software Verification*.

Requirements Management Process. The company is using two types of containers to bundle requirements for different purposes: *Features* and *Configuration Packages (CPs)*. As a feature we consider in this case a bundle of requirements that we can estimate market value and implementation effort and use those values later in the project scoping and prioritization. Configuration packages are used to differentiate the products by selecting different packages for different products. The company is using the similar approach to CPs as described in (Bosch 2000), where a configuration package is a set of requirements grouped to form a logical unit of functionality. Every requirement has to be associated with one or more CPs. The requirements engineers list the changes and CPs in their area of expertise in the *Configuration Package Module*. These modules have dependencies between each other and some of them are mutually exclusive (Bosch 2000). CPs that are common for all products in a product line are marked with an attribute stating that these packages cannot be removed from a product configuration. Hardware dependencies, which make individual requirements valid or invalid for different products, are also specified by the use of *Configuration Dependencies* on the requirements level. The model is similar to the Orthogonal Variability Model proposed by Pohl et al. (2005).

Product Planning. *Product Planners* are responsible for defining products from the platform available in a product line. They belong to the marketing division in the company so their task is to create an attractive product offer (Linden et al. 2007) rather than to perform the actual configuration of it. The product planers establish a concept of a new product which induces commercial overview, price range, competitor analysis and gives an overview of the high level requirements. This document serves as a basis for the *Product Configuration Specification*, which specifies the product based on capabilities offered by the platform. The product configuration specification specifies the configuration of a product concerning both software and hardware using the configuration packages defined in the configuration package modules including configuration dependencies. This model is also similar to the Orthogonal Variability Model proposed by Pohl et al (2005). The product configuration specification corresponds to the application variability model of the Orthogonal Variability Model.

Product Configuration Management. Product Configuration Management teams are responsible for integrating, building and managing variants in the product line. When configuring a new product from the product line, the product configuration manager uses hardware constraints derived from a hardware specification for each product in a cluster to set and configure the software. At this stage, the traceability from the configuration

parameters to the requirements is crucial. This part of the context is the subject for the improvement proposal in Section 5.

3 Research Methodology

In order to get a comprehensive picture of how variability management is performed at our case company, we decided to conduct a set of interviews with various employees in various positions within the company. The requirements management tool architecture was also explored to understand how variability is defined at the requirement level. During this phase the persons involved in process improvement for the requirements process were interviewed and consulted with during the exploration of the requirements management process.

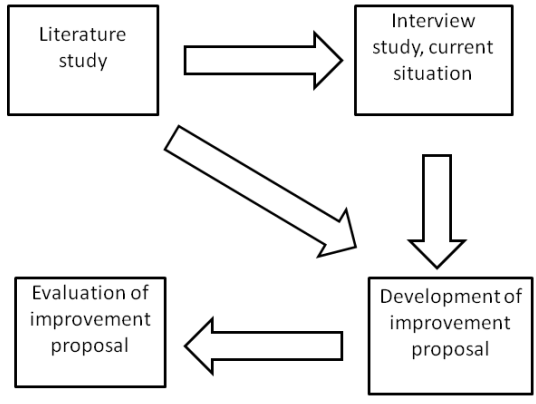

Figure 2.1: Research methodology.

The next step was to select key personnel to interview in order to get as many different perspectives how variability is managed and how products are configured as possible. By analyzing the case company's product configuration interface, the amount of variation for different development groups was established. One group with a large amount of product variations and one group with a small amount were selected for further investigation. To cover the whole process of variability, we have involved Product

Planners, Requirements Engineers, Requirements Coordinators, Developers and System Testers in our study.

The interviewed persons were selected based on their position in the company. Some persons were recommended by already interviewed. In some cases the person that was asked to participate in our study suggested a colleague as a replacement with the motivation that he was more familiar with the area. In total, 27 persons were interviewed. The interviews were semi-structured in order to allow the interview to change direction depending on the interviewee's answer, and adapted for the different roles and the progress of the interview study. This approach balances between early interviews that were more focused on the general aspects with later more specific interviews. The interviews took approximately one hour. During this time interviewers took notes continuously which were later summarized. During summarization, discrepancies between interviewers interpretation were discussed and, if needed, formulated as questions that were later sent to the interviewee. Apart from the summary, the interviewee also received a model of how he or she perceived the process of variability management. After interviewee approval, which sometimes was done after some minor changes, the data was ready to be analyzed. After interviewing 27 persons, it was decided that the received overview of the current process was satisfactory to proceed with analysis and propose improvements. Sample questions used at the interviews and distribution of interviewed personnel can be accessed at (Wnuk 2010c).

4 Results

In this section we present the results from our interview study. We describe the different perspectives on the configuration process, configuration activity measurements, and finally the problems that were identified.

4.1 Perspectives on the Configuration Process

Most of the stakeholders have a common view of how products are created. The product projects create a product concept, which is then used by requirements engineers in defining platform requirements. Later in the process the product planners are involved in creation and configuration of new products by creating change requests issues regarding both new and existing functionality. When previously created formal change request is accepted, it is send to the assigned developers team which performs implementation or configuration changes. The differentiation achieved in this manner is not explicitly documented in product specification but only in the minutes from the change board meetings. In the next section, the deviation from this common view is described, as well as the differences from the documented process model.

Product requirements coordinators, requirements coordinators and requirements engineers have limited knowledge about how variability is achieved due to their focus on the platform. They also state that developers do receive most of the configuration instructions through bug report issues from product planners, customer responsible and testers. We discovered that some variability is stated in the requirements' text in an implicit way creating problems with recognition and interpretation at the development phase. Product planners' knowledge about configuration packages is limited and they have not experienced the need for a better product documentation than what is delivered in the concept definition.

The developers express the opinion that information regarding variability is not communicated in a formal way. Instead, they get information about variability through their team leaders in a form of change requests at the late stages of development. These change requests are often used to configure products. The creation of new variation points is done in the platform project, and is therefore often based on assumptions made by the developers out of the previous experiences and informal communication with people involved in the process. The main opinion is that the information about what value that should be assigned to a variation point is possessed by individuals. The information is also not documented sufficiently in formal documents. Requests for new variation points or values are forwarded to the product configuration managers.

Product Configuration Management Perspective. We discovered that the product derivation process is iterative and similar to the one described by Deelsta et al. (2000). When a main product for a product line is created from the platform, it is based on the existing configuration of the previous similar product. This configuration is adjusted to the new hardware specification for the platform. Since the amount of configuration parameters in the configuration file has increased significantly, and they are not sufficiently documented product configuration managers are unable to keep track of all changes.

When a new product has been set up, it is built and sent to the product testers. Their task is to test the product and to try to discover software errors and functionality that might be missing. At this stage it is often difficult for the testers to determine whether errors depend on faulty configuration or software errors. Therefore they create a bug report towards the developers to initiate investigation of the reason of the failure. The errors are corrected by developers and new source code is later sent back to the product configuration manager, which is merging the delivered code from all development groups.

When the sub-product is created, the most similar product configuration is copied from the previous products. Next, the configuration manager responsible for the sub-products is trying to configure the product by checking product structure documentation and other relevant information. The required information is gained from multiple sources, which leads to

the double maintenance problem described by (Babich 1986), where uncertainties about the values of variation points are concluded by comparing with other projects. As a result a time consuming investigations have to be perform and very often influences the speed and correctness of the product configuration.

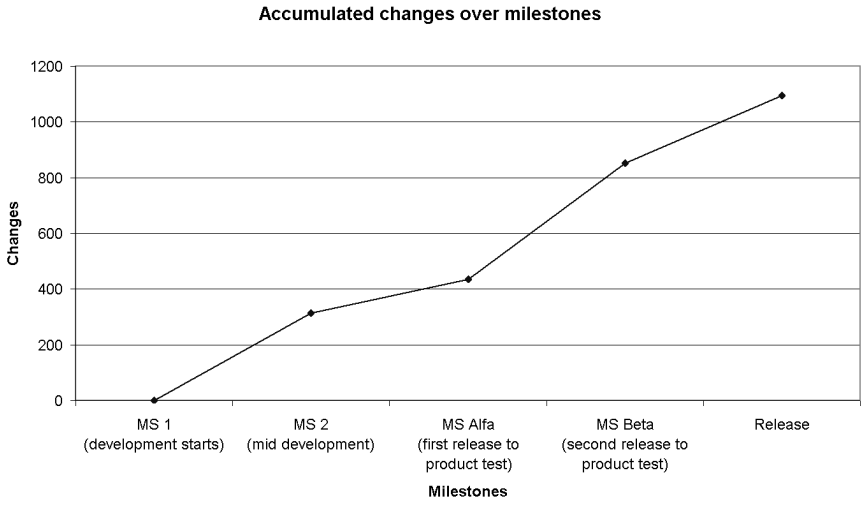

Figure 2.2: Accumulated changes to the configuration over milestones.

4.2 Configuration Activity Measurements

In order to understand how the configuration is changed over time, change related measurements were defined. The configuration file was chosen for each label of the code base in the product line. Labels are used to tag revisions of files produced by developers that are to be used by product configuration manager. The differences between each configuration file were calculated in order to get measurements describing how many parameters that were added, deleted or changed. The results are visualized in Figures 2.2 and 2.3. Note that over 60% of the configuration changes are done after the software has been shipped to the testers (MS Alfa).

The results support our previous observations derived from interviews, where developers admit that they configure the products based on bug reports and change requests. At the time this study was performed, the

configuration had over one thousand different parameters available at the product level, spread across a configuration file of thousands of lines. These parameters were controlling over 30 000 variation points in the source code with different levels of granularity. Further analysis showed, that one configuration parameter controls an average of 28 variations points, which suggests that most of the variability is quite fragmented. The source code consists of millions of lines of code in more than 10 000 files, giving an average 250 lines of code per variation point.

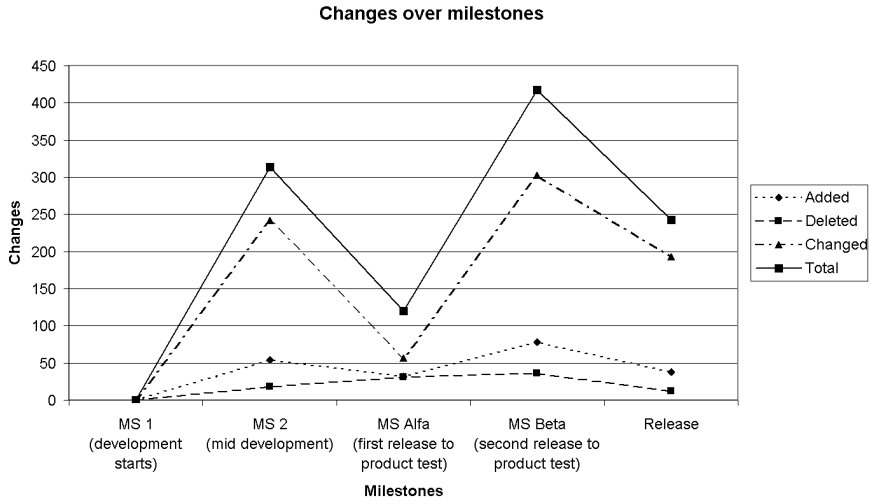

Figure 2.3: Changes to the configuration over milestones.

4.3 Problems Identified

According to Van Der Linden et al. (2007), the configuration manager should be responsible for maintaining the configuration of all variants and ensuring that the functionality for all products is covered. In our case it remains unclear who is responsible for binding the variation points of the platform to create a specific products. As a result, we experience creation of variation point that have no specific owner. Furthermore, since most of the development and architectural activities are platform focused and a role such as Application Architect or Product Architect responsible for binding variation points of the platform to create specific products is not

present in the current organization (Pohl et al. 2005). The lack of clear responsibilities results in an absence of clear, specific and strategic goals and long term improvements.

The configuration of new products is achieved in an iterative manner between developers, configuration management and testers (Deelstra et al. 2000). Due to the lack of a specific ownership, the configuration is not always properly reviewed, which is often a reason for missing functionality. As a result, testing and maintenance efforts may increase. The knowledge about product derivation and variability is not formalized (Deelstra et al. 2000, Bosch 2000).

As mentioned previously, the unrestricted rules for creating and managing variation points results in their excessive creation. Many variation points become obsolete either due to the fact that they were not created for product configuration purposes or because of the complex dependencies. It is undefined who is responsible for removing these obsolete variation points from the configuration file. This fact makes the configuration file hard to manage and overview.

In our case, the flexibility that needs to be copied by standardization of the product line (Pohl et al. 2005), in the sense of amount of variation points is too great and offers many more configuration capabilities than is needed for product configuration and differentiation. The number of variation points, and their structure is too complex to be managed by the people responsible for the product configuration and differentiation. The variability capabilities need to be more standardized and less detailed to handle the costs associated with the flexibility.

The biggest challenge throughout the organization turned out to be the lack of complete product specifications, which may lead to the following problems:

- Time consuming d̈etectiveẅork where information is gathered through informal communication and unofficial documents
- Faulty bug reports.
- Double maintenance of fragmented product information that exists in different documents and versions throughout the organization
- Faulty configuration
- Critical knowledge about variability configuring products possessed by individuals
- Increased effort in verifying the configuration of a product

These problems is tackled by the use of unofficial documents specifying the product characteristics for both hardware and software. The documents are created in an informal way and are neither reviewed nor a part

of the formal approval process, but still used throughout the organization. These documents and the related process can be improved with respect to configuration management, as uncontrolled documentation procedures may result in unintended product configurations.

5 Improvement Proposal

In order to improve the issues presented in Section 4.3, we have developed a set of improvements regarding variability documentation, granularity and management.

Improved traceability between requirements and variants. Our proposal will reuse the configuration package concept, described in Section 2, to associate the configuration parameters with the requirements. The configuration packages should be used by the product planners to configure the products. By associating the configuration packages with the configuration parameters, traceability links to both requirements and configuration parameters will be established. The division into configuration packages should be done in cooperation between developers and requirements engineers to fully capture all possible aspects of variability. Newly created variation points should be explicitly documented and spread across all stakeholders. This approach will result in a more complete traceability between the configuration packages and the configuration interface, and can be a step towards the automatic generation of a product configuration directly from the product configuration specification in the future.

Abstraction layer. The overview of the proposed abstraction level is described in Figure 2.4. In the current structure the configuration file contains all detailed feature configuration on a very low level for all products defined. The file is edited by both product configuration managers and developers and because of its size and granularity it is vulnerable and subject to merge conflicts. Our proposal introduces a new abstraction layer, CP-Conf, between the product configuration interface and the software modules. The low level configuration is moved into the lower layer, and a high level product configuration based on the configuration packages is used on the product configuration level. In this solution, the developers are becoming responsible for the CP-Conf layer and the modules associated with it. The product configuration manager is only responsible for the high level product configuration. To be able to introduce an abstraction level, configuration parameters in the configuration file need to be moved to a separated files where a parameters belonging to a certain development team reside. The specification of selected modules needs to be in these separated files too, since it depends on the selected configuration packages. Also, when this abstraction layer is introduced and the parameters are named according to the configuration packages, there should be no need to change the existing variation point naming since the parame-

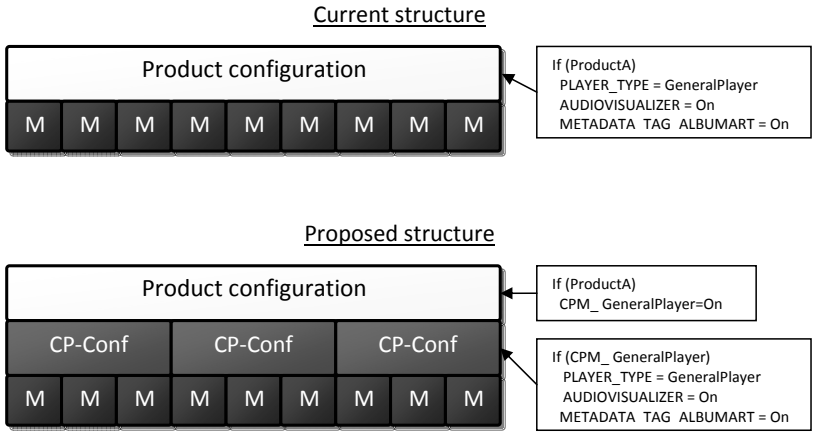

Figure 2.4: Overview of the proposed abstraction layer.

ters will be moved out from the main configuration file. The solution is described in Figure 2.5.

New configuration parameters. Today the naming of the configuration parameters includes a feature description indicating what functionality the parameter affects. However, the features in the configuration parameters are not mapped to the requirements by including an identifier connected to a specific requirement. Since the feature names originate from two sources, traceability is based only on human reasoning. We propose a new standard for configuration parameters where four types of parameters are available:

- The existing low level parameters which are presently used for product configuration. To remove or change these parameters is an infeasible work
- The existing parameters which define the hardware properties of the product should be assigned a prefix CFG_HW. Today many of the parameters created are hardware dependent and could therefore be removed by using the hardware properties instead of creating new parameters. The syntax of the parameters should include the serial number from the hardware requirements specifying its value.
- A new type of parameter for configuration dependencies. The name

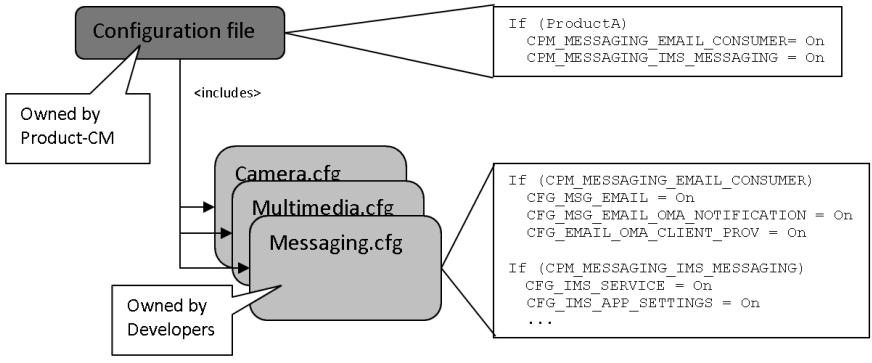

Figure 2.5: Configuration is distributed into configuration files according to the concept of Configuration Packages.

should include the dependency type (HW/FormFactor/Customer/-Market). The syntax can e.g. be CD_<TYPE>_<NAME>.

- An internal binding should be used when software varies non-significantly

Documenting variability. Currently, the documentation of variation points is not mandatory and resulting in its incompleteness. Since developers in our proposal will be responsible for the lower levels of variability, the documentation process will be simplified by responsible stakeholders' constraining. By introducing traceability between the product level configuration interface and the configuration packages, no further documentation is needed on the higher level. The name standard will be descriptive and in line with the configuration packages. It will enable stakeholders to find more information in the requirements management system, where the configuration packages are defined, described and associated with requirements.

Managing obsolete configurations. Many parameters in the configuration file are obsolete. Because of that we propose that the configuration file should be locked for changes. Parameters that do change but have the same value for all products should be moved to the development team's

specific file, and should not be a part of any configuration package. Similar to the configuration parameters, obsolete configuration packages that are not used in any product should be moved out from the software product line. If a configuration package is used in any product it should be incorporated into the platform and removed from the set of CPs. In the same fashion as the configuration packages, the high level hardware parameters should be left at the product configuration level, while its associated low level parameters should be moved to the proposed low abstraction layer and owned by the developers.

Availability of product specifications. All available configuration packages in the platform should be included in the product configuration specification, and a connection to the previously mentioned abstraction layer should be made. By applying this approach, the task of configuring a new product will be simplified and could possibly be automated in the future. The automatic configuration file generation can be based on the configuration packages defined in the requirements management tool.

6 Evaluation of the Proposals

The evaluation of the implemented proposals was carried out as a desktop pilot (Glass 1997), where the new structure was applied to the existing structure. The desktop pilot was run on a subset of the configurations belonging to two development teams. Two developers from each team, chosen based on their knowledge about configuration parameters, have participated in the redefinition part of the evaluation. The configuration packages defined by requirements engineers were used to group the existing low level configuration parameters, as described in the proposal. This was done in cooperation with the developers. When parameters could not be linked to a certain existing configuration package, the developers had to consider defining a new configuration package, configuration dependencies or hardware requirements. From these lessons learned we can conclude that:

- Packages need to be complemented with a more complex version for greater differentiation possibilities
- Some packages need to have defined dependencies to other packages
- The differences between some of the similar configuration packages need to be described by requirements engineers
- One package may in the future need to be split into several packages that contain end-user functionality and one common package that does not offer any end-user benefits. This one package is dependent on others previously described.

- Problems may arise when new configuration packages need to be created instantly. In this case the bottleneck will be the communication with requirements engineers.
- There are packages that can be removed from the product due to strong dependencies. In this case, product planners should not be allowed to de-select these packages.

After the redefinition of the configuration, the developers were asked to fill in the evaluation form (Wnuk 2010a), answering questions concerning the improvement proposal and its possible benefits and drawbacks. To get as many answers as possible, the information was held short and concise. The evaluation form was also sent out to all members in the first development group and to half of the members in the second group, totaling with 34 persons. 28 out of 34 persons have answered and the detailed results are accessible in (Wnuk 2010b).

From the evaluation it can be seen that the participants have been involved in the product configuration. They also see problems with how it is handled today. The proposal was considered as easy to understand and implement.

Some responders mentioned that customer specifications were not addressed enough. One participant also addressed a need for training in variability management. Most of the participants thought that the responsibilities and the separation of product and feature configuration is easy to understand. In the qualitative part of the results, it was confirmed that the workload will be reduced by improved division of responsibilities.

Most responders strongly agreed to that our proposal should increase the quality of products. On the other hand, a few responders claimed that the quality of the products is now high enough and that our proposal will not make any significant difference. The question addressing improvement in the configuration efficiency scored above average, which indicates that this proposal would have a significant effect on efficiency in the way of working rather than end-product quality. This was emphasized by some people who stated that the configuration would become more manageable and less time consuming.

On the question regarding drawbacks there were concerns that the configuration packages may get too large and fail to offer the needed from market perspective detailed level of configuration. It was also mentioned that there will be a stabilization period until the CPs are clearly defined. One responder expects that quick fixes will be hard to handle using CPs, and that there therefore could lead to the "quick and dirty" solutions which are hard to maintain. There is a risk that the number of CPs will increase and that the same problems will arise again. Some responders were also worried about customer specific configurations, which the proposal does not specify in detail. Most participants stated that their work will not be affected negatively. Moreover, they stated that there will be less work for the

developers with the proposal. The developers would have fewer responsibilities and for some participants their responsibility for product configuration will be completely removed. Overall, the proposal was considered as a better solution than the current way of working.

In the evaluation with the configuration management strategists the responses were positive. Among the positive comments were the possibilities to define a clear process with unambiguous responsibilities, to automate product derivation and verification and to improve the product derivation process. The concerns regarded the need for a streamlined process for managing the configuration packages, including exception handling. Possible dependency problems when a configuration package spans many development teams were also discussed. The overall impression was very positive.

Threats to validity. The way how people were chosen to participate in the interviews can lead to insufficient results. By getting recommendations to which people to interview the risk of getting a subjective picture increases.

The results can be biased by continuous communication with the contact person in the company or by the fact that some concerned stakeholders might have been overlooked in different parts of the case study.

When performing these kind of evaluations, it is difficult to cover all aspects. We are aware that this evaluation only takes a few of the affected development teams into account, and therefore some important information may not be reached. Furthermore, the amount of variation points that each development team is responsible for or shares with other groups varies. Therefore, the scale of affection of the proposal on each development team may vary.

We have not yet performed any evaluation among other stakeholders, like product planning and requirements engineers. Although they are not involved in the technical parts of the proposal, they are part of the process associated with the proposal and it is therefore a drawback not to have these stakeholders represented in the evaluation.

We also see some challenges concerning the ability to maintain the new way of working. It is important that the configuration packages only reflect the current needs for variability and that the configuration packages are not created proactively in the same manner as variation points are created today. It is also important to educate people in order to consistently convince them of the gains achieved about the new praxis.

7 Related Empirical Work

Industrial case studies in existing literature (Brownsword and Clements 1996, Jaaksi 2002, Linden et al. 2007, Clements and Northrop 2002a;b, Deelstra et al. 2000) describe the process of introducing product lines. These

studies report similar problems to those reported in this paper appear. For example, in the Thales case (Deelstra et al. 2000) documentation of the platform has deviated from the actual functionality as the platform has evolved. In other cases, (Brownsword and Clements 1996, Clements and Northrop 2002a) the enormous amount of low level variability in software was reported. Clements et al. (2002b) reported that the variability was present only on the files and folders level. In the Philips case (Linden et al. 2007), the problem of too many dependencies between components, resulting in much time spent on integration problems, was reported. Patzke et al. (2006) discovered that many of the differentiation point were actually obsolete and not used any more. The company was also struggling with outdated documentation that was not updated regularly.

In most cases a product line approach was introduced in an evolutionary way, apart from one example (Clements and Northrop 2002a), where all ongoing projects were paused and the resources were moved to the introduction of the product line project. In some cases, the product line was developed around a new architecture, while assets were derived from an existing base e.g. (Linden et al. 2007). Sometimes, a new product line was based on the most similar product from the most recent project. Some cases, like Brownsword and Clements (1996), claim that their main success was achieved in the architecture and reorganization, and resulted in the change of the hardware to software cost ratio from 35:65 to 80:20.

The issue of improved traceability between requirements models and variant has been addressed in the literature. For example, Clotet et al. (2008) present an approach that integrates goal-oriented models and variability models while Alfarez et al. (2008) present a traceability meta-model between features and use cases. Both example cases seem to be domain independent but are evaluated on relatively small examples which leaves the question of applicability in a large-scale industrial context open.

8 Conclusions

As mentioned in Introduction, software product lines improves the quality of the products and reduces the time spent on a product development. However, managing a product line and its variation points efficiently requires a consistent way of working and clear responsibilities. In this case study it has been found that new products are derived by copying the most similar configuration from previous products and iteratively configuring the product between developers, CM and testers. The variability is neither clearly specified nor documented. The responsibilities are unclear. There is no connection between the requirements and the configuration possibilities in the product line. These aspects affect negatively the possibilities to verify the configuration and the time spent on product configuration.

To be able to cope with these issues, improvement consisting of an ab-

straction layer in the configuration interface have been proposed. This abstraction separates the low level feature configuration from the high level product configuration, and establishes a traceability from requirements to configuration. To clarify the product configuration and ensure that everyone is working consistently, we propose that a product specification, based on these configuration packages, is used throughout the company. Below, we summarize identified problems and corresponding possible improvements:

- **Large number of variation points with an unmanageable granularity.** Variation points are encapsulated into configuration packages, separating the high level configuration from the low level configuration, and resolving the granularity issues.
- **Unclear responsibilities and unstable process for the product configuration.** By dividing the configuration into different layers and proposing responsibilities are clarified.
- **No clear traceability between configuration parameters and initial requirements.** By introducing an abstraction layer based on configuration packages, the configurations are directly linked to the initial requirements.
- **No complete product specification available.** A new and managed product specification based on configuration packages are spread throughout the organization and used by all stakeholders.
- **Products are configured in an inefficient and iterative process without using the initial requirements.** By the use of a complete product specification and a configuration interface based on the same configuration packages, the configuration can be done at early stage.

The evaluation of our proposal shows that the developers are coherently positive to the suggested improvements. To validate out proposals, the changes were simulated together with two development teams. The results showed no major obstacles, but emphasized the importance of cooperation between the requirements engineers and the developers in the definition of the configuration packages. The expectations of this proposal are as follows:

- To reduce effort and time spent on iterative configuration
- To ensure a higher product quality by improved product verification
- To state more clear responsibilities among stakeholders
- To make the information concerning variability within the company more accessible

It is stated in (Pohl et al. 2005) that explicit documentation of variability can help to improve making decisions, communication and traceability. Following Pohl et al. (2005) we can also conclude that introducing abstraction levels for variation points and variants improves understanding and management of software product line variability. As a result, we conclude, that our improvement proposals may be relevant for other contexts by addressing the general issue of variability in software product lines with abstraction mechanisms on both requirements and realization level (Bosch et al. 2002).

This paper contributes in a detailed investigation on product derivation from a large software product line, which addresses research question Q1. Question 2 is addressed in Section 4.3 as a set of challenges in practice. Finally, Q3 is addressed by the improvement proposals, described in Section 5 that may increasing product quality and decreasing the effort needed for product derivation.

Acknowledgments

This work is supported by VINNOVA (Swedish Agency for Innovation Systems) within the UPITER project. Special acknowledgments to Per Åsfält for valuable contributions on problem statements and research direction.

References

M. Alfarez, U. Kulesza, A. Moreira, J. Araujo, and V. Amaral. Tracing between features and use cases: A model-driven approach. In *Proceedings of the Second International Workshop on Variability Modelling of Software-intensive Systems*, pages 81–88, 2008.

W. A. Babich. *Software configuration management: coordination for team productivity*. Addison-Wesley Longman Publishing, 1986.

J. Bosch. *Design and Use of Software Architectures Adopting and evolving a product-line approach*. ACM Press/Addison-Wesley, 2000.

J. Bosch, G. Florijn, D. Greefhorst, J. Kuusela, J. H. Obbink, and Klaus Pohl. Variability issues in software product lines. In *Lecture Notes in Computer Science*, volume 2290, pages 303–338, 2002.

L. Brownsword and P. Clements. A case study in successful product line development. Technical Report CMU/SEI-96-TR-016, Carnegie-Mellon Software Engineering Institute, Pittsburgh, 1996.

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

P. Clements and L. Northrop. A software product line case study. Technical Report CMU/SEI-2002-TR-038, Carnegie-Mellon Software Engineering Institute, Pittsburgh, 2002b.

R. Clotet, D. Dhungana, X. Franch, P. Grunbacher, L. Lopez, J. Marco, and N. Seyff. Dealing with changes in service-oriented computing through integrated goal and variability modelling. In *Proceedings of the Second International Workshop on Variability Modelling of Software-intensive Systems*, pages 43–52, 2008.

S. Deelstra, M. Sinnena, and J. Bosch. Product deriviation in software product families: a case study. *The Journal of Systems and Software*, 74(2):173–194, 2000.

R. L. Glass. Pilot studies: What, why and how. *The Journal of Systems and Software*, 36(1):85–97, 1997.

A. Jaaksi. Developing mobile browsers in a product line. *IEEE Software*, 19 (4):73-80, 2002.

F. J. Linden, K. van der Schmid, and E. Rommes. *Software Product Lines in Action The Best Industrial Practice in Product Line Engineering*. Springer-Verlag, 2007.

T. Patzke, R. Kolb, D. Muthig, and K. Yamauchi. Refactoring a legacy component for reuse in a software product line: a case study. *Journal of Software Maintenance and Evolution: Research and Practice*, 18(2):109–132, 2006.

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

Krzysztof Wnuk. Evaluation form can be accessed at. `http://www.cs.lth.se/home/Krzysztof_Wnuk/VaMoS_2009/EvaluationForms.pdf`, January 2010a.

Krzysztof Wnuk. Results of evaluation can be accessed at. `http://www.cs.lth.se/home/Krzysztof_Wnuk/VaMoS_2009/ResultsOfTheEvaluation.pdf`, January 2010b.

Krzysztof Wnuk. The interview's instrument and participants distribution can be accessed at. `http://www.cs.lth.se/home/Krzysztof_Wnuk/VaMoS_2009/InterviewInstumentAndDistribution.pdf`, January 2010c.

Paper III

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 Man 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 3.1, 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

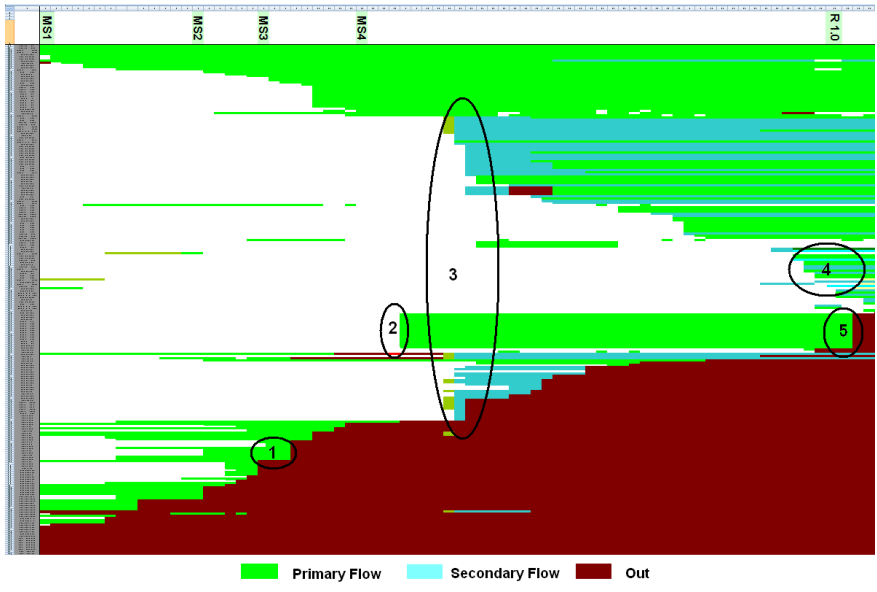

Figure 3.1: Feature Survival Chart for project A.

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 3.1 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 3.1 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-

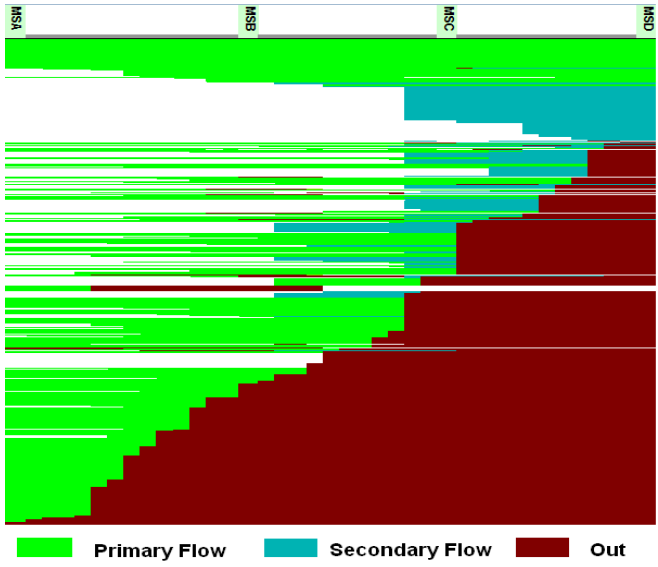

Figure 3.2: FSC for project B.

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 3.1, 3.2 and 3.3 we present one FSC respectively for three projects denoted A, B and C. Additional information about the projects is presented in Table 3.1.

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-

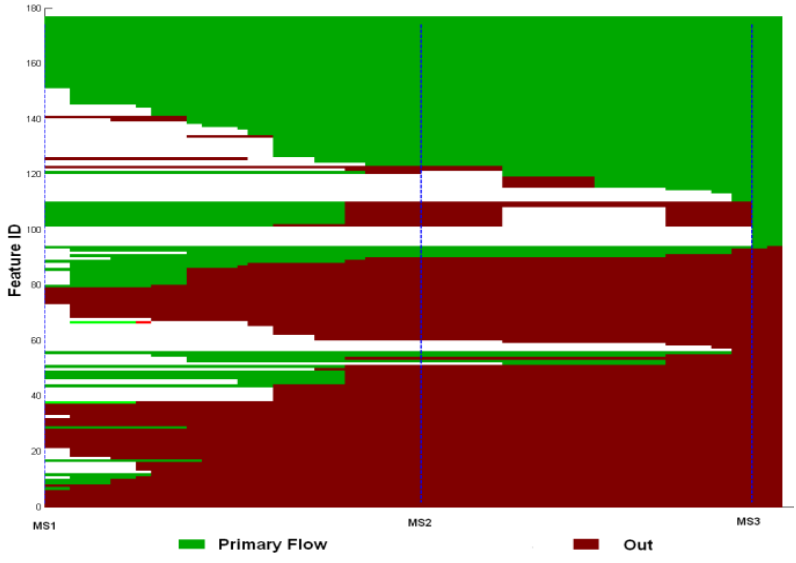

Figure 3.3: FSC for project C.

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 et al. (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 3.1: 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 3.2 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 3.2, 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 3.4 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 3.4 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 3.5 is showing that many features were

Table 3.2: Results from a theoretical analysis of proposed measurements, by # we mean 'number of'

Measurement	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	# 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	<>,=

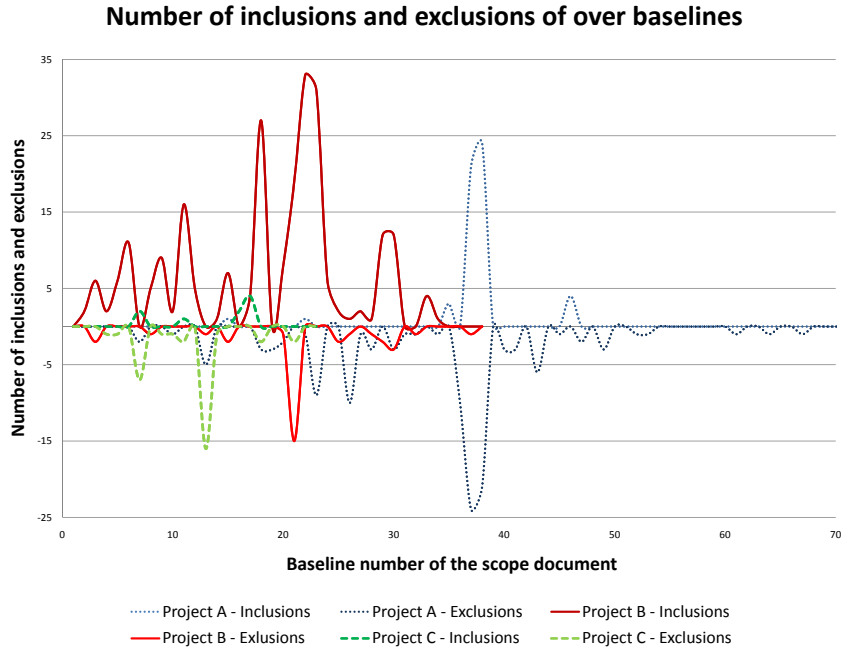

Figure 3.4: Number of positive and negative changes as a function of a baseline number (M1).

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 3.6, 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-

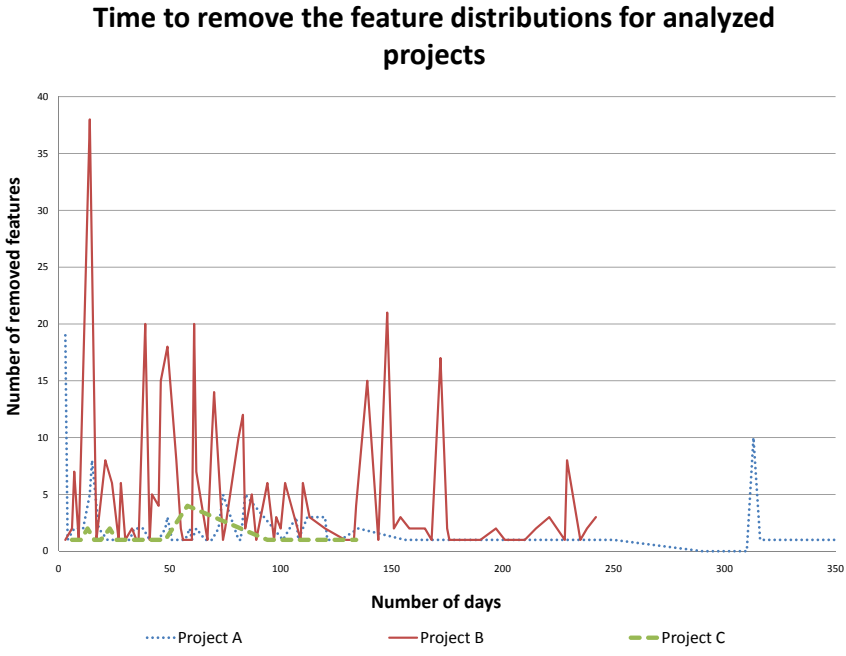

Figure 3.5: Distributions of time to remove the feature (M2).

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 3.7 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 3.7 reveals that

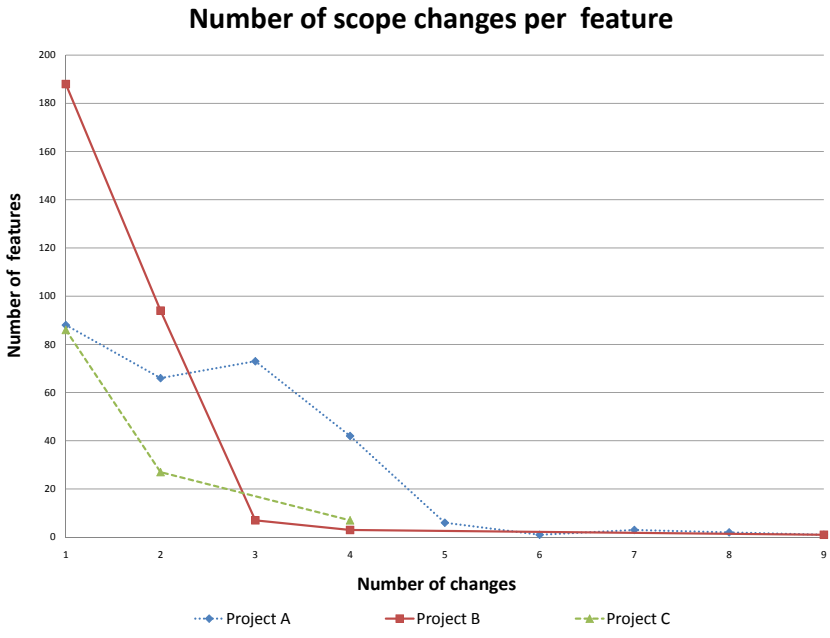

Figure 3.6: Distribution of number of changes per feature (M3).

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 3.1. 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 3.1, 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

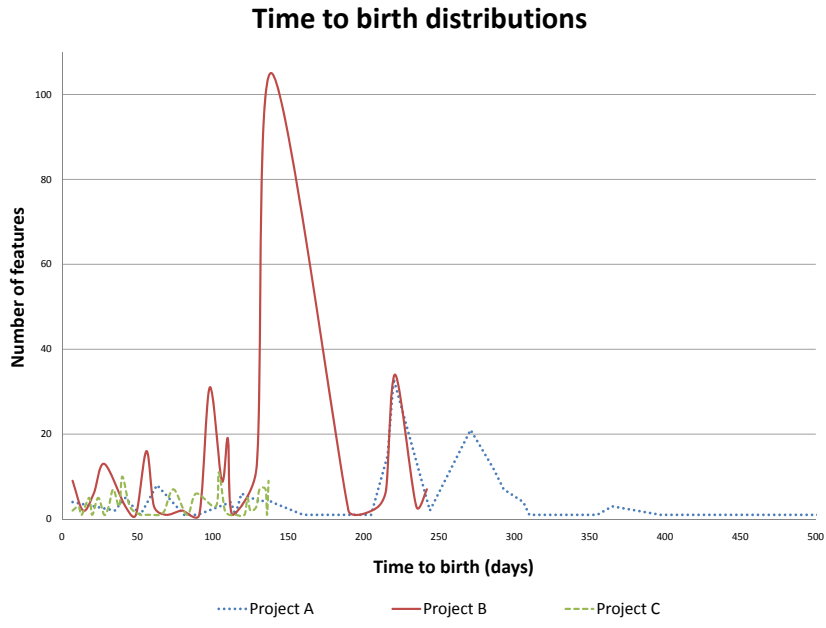

Figure 3.7: Time to birth distributions (M4).

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 3.3: 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 3.1, 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 descoping in Project A. The results are presented in Figure 3.8. We have analyzed 120 de-scoping decisions that belong to project A. The result is shown in percentages in Figure 3.8, 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

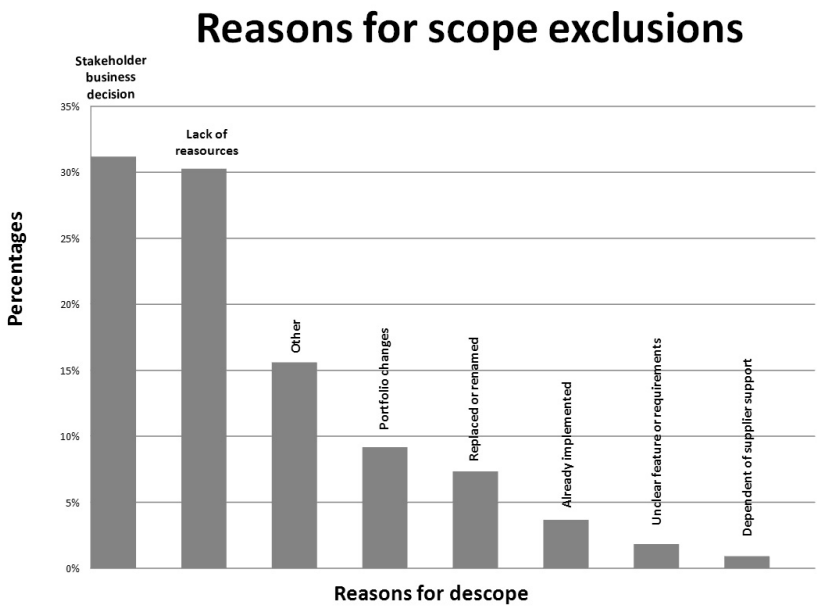

Figure 3.8: Quantitative analysis of reasons for removing features from the scope of the Project A.

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 et al. (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 scop-

ing 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.

References

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

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

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

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

N. E. Fenton and S. L. Pfleeger. *Software Metrics A Rigorous & Practical Approach*. Thomson Publishing, 1996.

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

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

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

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

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

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

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

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

K. Wnuk, B. Regnell, and L. Karlsson. 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)*, pages 41–50, 2008.

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

C. Wohlin and A. Aurum. 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)*, pages 246–255, 2005.

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

Paper IV

Feature Transition Charts for Visualization of Cross-Project Scope Evolution in Large-Scale Requirements Engineering for Product Lines

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
Fourth International Workshop on Requirements Engineering Visualization (REV09),
September 2009, Atlanta, USA

ABSTRACT

In large-scale multi-project software engineering it is a challenge to provide a comprehensive overview of the complexity and dynamics of the requirements engineering process. This paper presents a visualization technique called Feature Transition Charts (FTC) that gives an overview of scoping decisions involving changes across multiple projects based on previous work on within-project visualization of feature survival. FTC is initially validated using industrial data from the embedded systems domain in a multi-project product line engineering context in dialogue with practitioners. The initial validation provided specific improvement proposals for further work and indicated a positive view on the general feasibility and usefulness of FTC.

1 Introduction

Requirements for software intense embedded systems can often be counted in thousands and often describe cutting edge functionality which can have many complex dependencies with other parts of the system. In this case, the decision about which candidate requirements should be included into the scope of the project and which should not is not always obvious. It is strongly emphasized by researchers, like for example Boehm et al. (2003) and Boehm and Huang (2003), that the inclusion process should be value-based, while others, for instance Wohlin et al. (2005), argue that a good understanding of the underlying decision-making process is needed so that researchers can support it in the best possible way.

In product line engineering, the selection process is often called scoping and is considered crucial for achieving economic benefits (Pohl et al. 2005). Furthermore, many embedded systems companies are releasing their products to an open market in a mode often called Market-Driven Requirements Engineering (MDRE) (Carlshamre and Regnell 2000). In this case, the complexity and uncertainty of scoping decisions may increase even more and may result in a situation where the decisions have to be made a priori with limited knowledge about their market value and their cost for implementation. Decisions often need to be revised due to changes in the market situation (DeBaud and Schmid 1999) or other unplanned constraints. Therefore, some requirements are deferred to later releases for a number of reasons (Wnuk et al. 2008). In de-scoping there can be both rejected requirements (that for example are out of scope of the current strategy) and postponed requirements (that for example are delayed because of lack of resources). Large scope changes with many deferred requirements may significantly delay a project's overall business value, and are thus interesting to track in project and product management.

In one of our previous papers, (Wnuk et al. 2008), we have analyzed three large platform projects to test the applicability of our visualization technique denoted Feature Survival Charts (FSC) on empirical data. In the case of the company under study, the decision process is based on bundles of requirements, called features, rather than single requirements. A single scoping decision may concern one or many features and their dependencies. In our earlier work, we have experienced a very large number of features that were de-scoped from analyzed projects (Wnuk et al. 2009). Due to one of the limitations in our previous work, namely the fact that the Feature Survival Charts only show a single project during analysis, it is not possible to analyze if some of de-scoped features are moved to another projects. In a real product development setting, we can assume that many scope changes span across several projects. Scope changes that seem to result in rejected features may in fact concern postponed features that appear in later projects. Based on these observations, the presented work addresses the following two research questions (Q2 is a refinement of the

more general Q1):

- Q1: How can scope changes across projects be visualized?
- Q2: Which visualization mechanisms are effective in providing an overview of the timing and magnitude of feature transitions across projects in a large-scale setting?

The main contribution presented in this paper case study is a prototype visualization technique called *Feature Transition Charts (FTC)* that can show features transitions across projects, while scaling to hundreds of features. FTC has been initially validated using real data from a large-scale product line engineering case in the domain of embedded systems, and iteratively refined in dialogue with domain practitioners.

The paper is structured as follows: Section 2 provides related work. Section 3 describes background information about the context of our industrial case study. Section 4 describes the methodology used in this study. Section 5 explains our visualization technique. Section 6 describes the results from applying our technique to two industrial projects. Section 7 defines and evaluates the results. Section 8 provides conclusions and discusses their limitations.

2 Related Work

Current state-of-the-art research in software engineering has established an opinion that decision processes are the driving forces to organize a corporation's success (Gregorio 1999). Researchers have already contributed in creating better support for decision making based on best knowledge and experience, computational and human intelligence, as well as a suite of sound and appropriate methods and techniques (Ruhe 2003). The decision-making process has also been addressed by researchers working in the requirements engineering field, since it is dependent on requirements being captured, analyzed and specified before any decision about implementation can be made. The contributions in this area are visible within different aspects of requirements management, namely prioritization (Karlsson and Ryan 1997, Karlsson et al. 1997) or understanding of requirements dependencies (Carlshamre and Regnell 2000, Dahlstedt and Persson 2003). Others have worked on connecting requirements engineering processes to decision making (Aurum and Wohlin 2002; 2003, Regnell et al. 2001). Finally, some effort has already been dedicated to the understanding of release planning (Carlshamre 2002), while others proposed the usage of generic algorithms to plan for different releases (Ruhe and Greer 2003).

The reasons behind scope changes have been discussed in (Wohlin and Aurum 2005, Wnuk et al. 2009). Others have investigated the root cause for changing requirements, namely requirements uncertainty (Ebert and

Man 2005). Selecting the appropriate set of requirements has also been addressed by researchers related to the product line community. However, the main research stream is, according to Schmid (2002), focused on the identification aspect of scoping (Kishi et al. 2002, Savolainen et al. 2007) and does not address changes beyond formal decision to approve the scope of the project.

In the requirements visualization field, the research effort is focused mainly on three aspects of requirements engineering (Gotel et al. 2007). The first aspect is addressing the problem of creating a visual representation of requirements and their attributes based on a formal language (Teyseyre 2002, UML 2010) or even visualizing these representations (Konrad et al. 2006). The second aspect addressed in the requirements visualization literature is focusing on visualization of the structure and relationships between requirements (Duan and Cleland-Huang 2006, Ozakaya 2006, Sellier and Mannion 2006). Finally, the third aspect, which is most relevant to the work presented in this paper, is addressing elicitation (Pichler and Humetshofer 2006) and decision-making activities (Feather et al. 2006).

Thus the work has been conducted on release planning itself, and scoping as its vital part. However, to our best knowledge no studies have actually looked into the phenomenon of postponing features for the next release, and no studies have made an attempt to create a visual support for this aspect of product development that may help to assess its scale in real projects.

3 The case of the company under study

Our results are based on empirical data from industrial projects at a large company that is using a product line approach. 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 several 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 enables flexibility and possibilities for adaptation of the platform project, while the major release is dedicated to address functionality of the highest importance.

The company uses a stage-gate model with several increments (Cooper 1990). There are *Milestones (MS)* and *Tollgates (TG)* to control the project progress. In particular, there are four milestones for the requirements management and design before the implementation starts: MS1-MS4. The scope of the project is constantly changing during this process. In this case,

the project management makes scoping decisions based on groups of requirements that constitute new functionality enhancements to the platform, called features. The scope of each project is maintained in a document called the 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.

According to the company guidelines, most of the scoping work should be finished before reaching the second milestone of the process. After this milestone, the content of the main release of the platform project should be well defined and remain stable so that more effort can be addressed towards the preparation for the implementation phase. Therefore, minor releases are introduced to enable necessary adaptations that related product projects require. The product projects start approximately at MS2.

After MS4, the project starts its implementation phase. Even though the scope of the platform projects together with their minor releases should be defined and approved at this stage, some important changes may still happen and decisions about how to address them must be made. The changes may be related to unplanned issues with the development of previously approved features or they may be requested by important customers as a result of a rapidly changing market situation. These late changes or adaptations are usually handled by adaptations of the platform required by certain platform project releases.

4 Research Methodology

At the beginning of this study, a set of research questions and assumptions was formulated by the researchers. Researchers assumed in this case that the feature transition is a phenomenon that can have a significant representation in real life projects. It was also assumed that there is an impact of these types of changes on the quality attributes of the requirements management processes and the resulting products that should not be neglected in conscious product management. As a result, three types of transitions were defined as the most important and they are discussed in Section 5. As a next step, the empirical investigation of previously derived assumptions in the given company context. In this step, we have analyzed two large platform projects. The projects under consideration contained hundreds of features, and they were related in such as way that the first one was a direct ancestor of the following one. On a set of two large projects, a name matching algorithm, checking for multiple occurrences of the same feature id among the analyzed projects, was applied to find possible reoccurrences of the same features between the projects. The result is visualized in Figure 4.1, a distinction between forward and backward transitions has been

made, and it (the distinction) is followed by a description of the transitions in Section 6.1. In the next step, each single project was analyzed for possible internal transitions. Many transitions were discovered and they are visualized in Figures 4.2 and 4.3 followed by analysis in Section 6.2. Finally a multiple-step feature transition graph was proposed and it is presented in Section 6.3.

As the final step of the methodology, an interview study with two practitioners, namely one requirements management process manager and one requirements engineer, was performed. The interviews lasted for about one hour each and were semi-structured. Before conducting interviews, the list of questions to ask was prepared based on the initial assumptions described in this section and in Section 1. Researchers have evaluated their pre-understanding of the feature transitions phenomenon, and feedback from practitioners in the form of their suggestions for improvements was thus collected. Some of the important aspects of the discussion were the usefulness of the visualization technique presented and the importance of the need to quickly spot feature transitions in the case of the company under study. The results from this step are presented in Section 7.

5 Feature Transition Types

In this section, different types of transitions are discussed. For each platform project, there is one release, called the major release that provides the main part of the functionality, while other minor releases focus on adaptations and additional functionalities needed for certain products associated with them. In this context, the distinction can be made between *within-project, cross-project and multiple transitions*. Each type of transition is defined and described respectively in the following sections. another platform project. This type of transition is defined in Section 5.3.

5.1 Cross-project Feature Transitions

A cross-project transition occurs when a feature is moved between two platform projects in one step. In case that a feature is moved to the following platform project, it may be included into the earliest possible release of the next platform project (the main release). However, the destination release may not always be the main release. There may also be a situation where one feature first gets internally moved to another release of its original platform project and then later moved to another platform project. This type of transition is defined in Section 5.3.

There may be various reasons that cause cross-project transitions. Firstly, features may simply be moved to the next platform project due to resource constraints, secondly due to a lack of proper hardware. Thirdly because of the unfinished functionality it is difficult to minimize all non-functional

issues so features may be rescheduled for a later project where they can possibly be mitigated. The decision to perform a cross-project transition should be made after a careful analysis of the impact of the transition on the included features' market-values and possible efforts for implementation. Cross-project decisions also require impact analyses to ensure that for example other features that enable new functionality to work in a new context are available. The decision should also be confirmed with a business plan for the considered functionality so that no crucial market opportunities will be missed by a decision.

5.2 Within-project Feature Transitions

This type of transition occurs between two releases within one platform project. Features are moved between releases in one step. Each platform project has in our case a set of consecutive releases that differs in providing functionality. Apart from the main release, always scheduled at the beginning of the platform project, all other releases are introduced and scheduled later in the project. Internal transitions may be caused for reasons similar to those for the external transitions: lack of resources, dependencies on suppliers or other constraints. The basic difference is that a feature internally moved is staying in the scope of its platform project while being rescheduled to a usually later release. From a business value perspective, we believe that this type of transition can be considered as less critical and to some extent positive since it enables a quicker and more flexible response to rapidly changing market situations or unplanned project difficulties.

5.3 Multi-step feature Transitions

The last type of transition may happen both between platform project releases and the platform projects. The main difference between the previously described types of transitions and this type is that a transition is made multiple times either within one project or between different projects. The situation where a feature is moved multiple times only between the platform project's releases or between platform projects can also be classified as a multi-step transition, but we assume that it may be rare in industrial projects. Multi-step transitions can significantly influence the market-value of moved features and their cost of implementation. The management of a project can benefit from careful analysis of this type of transitions and tries to assess the impact of the transition on involved features' market value and, if applicable, their implementation cost. This type of transitions is visualized and described in Section 6.3.

Table 4.1: Characteristics of analyzed projects.

Project	Nbr. of features	Percentage of internal feature transitions	Percentage of external feature transitions
A	206	17%	8%
B	568	6%	0.5%

6 Visualizing Feature Transitions on the Industrial Example

In order to confirm or reject our pre-understanding about described types of feature transitions, we applied a new visualization technique to data from an empirical set of two large platform projects. The characteristics of analyzed projects are presented in Table 4.1. The projects differ significantly in the number of features ever considered in their scope, but have a similar number of associated technical areas.

An initial analysis of transitions present revealed that internal transitions represent 17% of all scope changes for Project A, and 6% of all scope changes for Project B. On the other hand, external transitions turned out to be 8% of all scope changes for Project A and only 0,5% of all scope changes for Project B. The numbers presented are, however, influenced by the fact that only two projects were analyzed. In general, each project will have two, or even more, associated projects; one from which the project is receiving backward transitions from and one or more to which forward transitions are sent to.

All types of transitions are visualized using a modified concept of *Feature Survival Chart (FSC)* presented in (Wnuk et al. 2008), namely *Feature Transition Chart* (FTC). The FSC, shows scope changes over time, which is illustrated on the X-axis. Each feature is positioned at a specific place along 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 in color. Based on discussions with practitioners, we decided to use the following coloring scheme: green for features considered as a part of the primary flow, red for features considered as de-scoped and, if applicable, orange, yellow, pink and cyan for other 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 śtepsẃith areas of different colors. The larger the red areas 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.

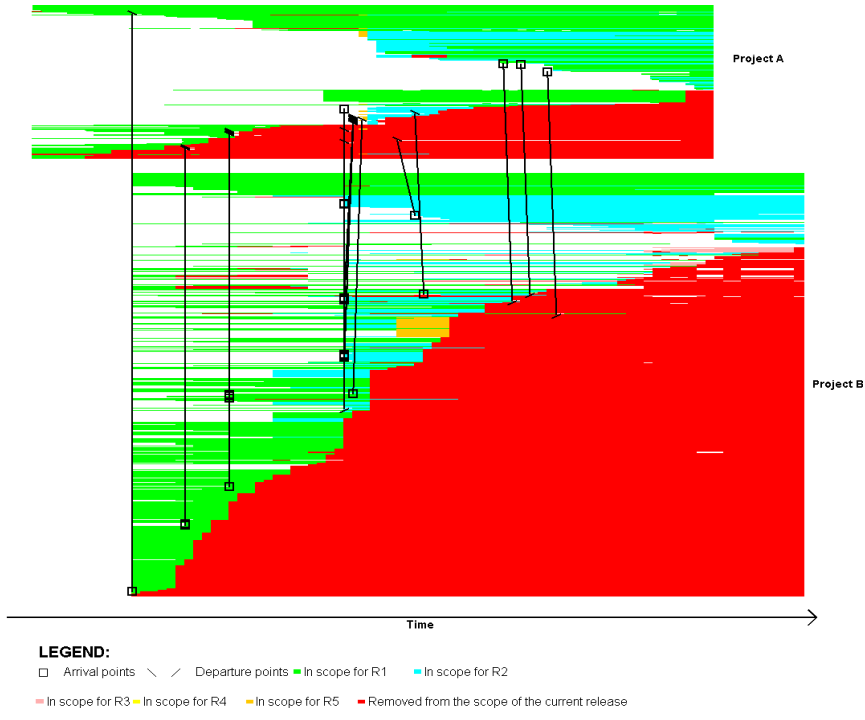

Figure 4.1: Cross-project transitions between Projects A and B (see Section 5.1). The full-size color figure can be found at http://www.cs.lth.se/home/Krzysztof_Wnuk/REV09/Figure1.bmp

The FTC is complementing the original FSCs by marking transitions of features together with their departure and arrival points. In order to find external transitions, we have searched feature identifiers involved in both projects for exactly matching names. This technique resulted in a significant fraction of features transmitted between the projects. In order to indicate the transitionś departure and arrival points, a set of the following symbols is used. The departure points are marked by 45° lines leaning down if the transition is forward, or leaning up if the transition is backward. The destination points are marked by a rectangle. This technique enables the representation of the magnitude of concurrent changes in analyzed projects, which pure lines can not adduce. It may however be inefficient when many changes happen at the same time due to the overlap of symbols. Various releases within the analyzed projects are represented by various colors. Features removed from the scope of the release that they finally arrived at, or even belonged to, are marked red.

6.1 Cross-projects Feature Transitions

We have found 21 forward transitions (from Project A to Project B) in the analyzed dataset and only 4 backwards transitions (from Project B to project A). The results are depicted in Figure 4.1. The backward transitions are interesting to analyze since they mean that features were moved to an earlier platform project. The lines depicting transitions are not always orthogonal, which means that there has been a delay in transitions.

In order to analyze the reasons for external feature transitions, we have checked the decision logs for both projects for the descriptions of proposed changes. The analysis of forward transitions revealed that seven transitions were caused by stakeholders business decisions. The decision in these cases was to refine the features and accept only a limited scope in the next project. In three other cases, lack of development resources caused the features to be moved to the next project. On the contrary, in two other cases the resources were available, but the time schedule was too tight to be ready with the implementation of given features. In two other cases, dependencies on either suppliers or other features caused the external transitions. In one case, the feature failed compliance testing with a certain standard required by the customer so it had to be moved to the next release of the project for improvements. Finally, in two cases features were only partly ready for the original project deadline and therefore were moved to the next release. The interesting information here is that most of the functionality was available, but the company decided to postpone the commercial availability of features until the complete feature implementations were ready. The analysis of backward transitions revealed that for all cases there was a request to provide the functionality in an earlier project. The requests were accepted after checking that the development teams were capable of meeting the new deadlines and that the new features were technically compatible with the destination project's source code.

6.2 Within-projects Feature Transitions

Next, we have visualized internal transitions within both projects A and B. The results are depicted in Figures 4.2 and 4.3. Various platform project releases are placed next to each other in the graphs. The time offset is not present in this case, so that all transitions are represented by orthogonal lines and transition symbols similarly to across-project transition visualization. Due to the doubling of data points (only for the features that have been moved within-project) in this type of graph, the data has been minimized by removing data points from after the transition for the source project and before the transition for the destination project. As a result, a more accurate picture of the size of various platform project releases can be achieved.

In the case of Project A, we experienced in total 34 within-project tran-

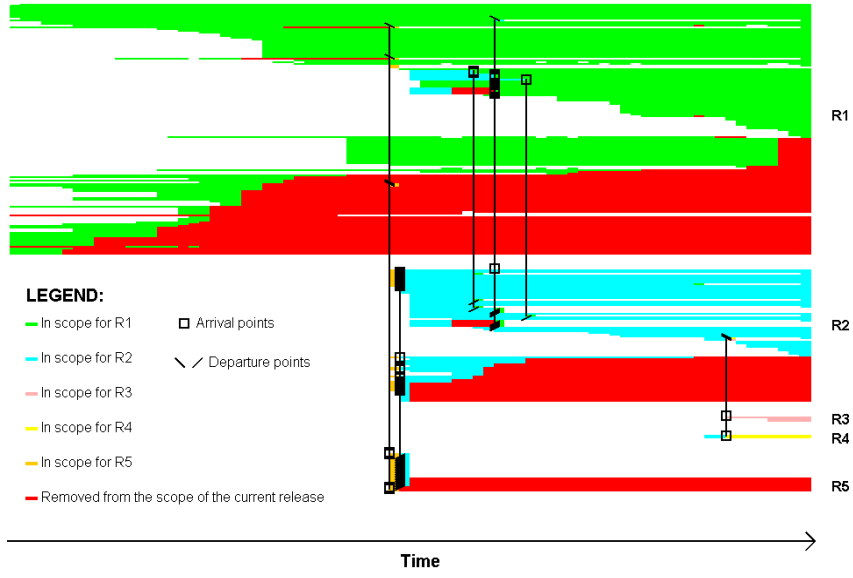

Figure 4.2: Within-project transitions for Project A (see Section 5.2). The full-size color figure can be found at http://www.cs.lth.se/home/Krzysztof_Wnuk/REV09/Figure2.bmp

sitions. 18 of them turn out to be originating from R1 and 15 from R2. All mentioned transitions are targeted to later releases. On the other hand, one transition is originating from R4 and is directed towards an earlier scope release. In the case of Project B, 36 within-project transitions were found in total. The interesting observation here is that 19 transitions are originating from release R5, another 10 from release R2. Both groups of transitions are targeted to earlier releases. In this case, only five transitions originated from release R1 and only two from release R2.

6.3 Visualizing multiple transitions.

The last type of visualization is representing only features that have been transferred multiple times. Due to the fact that these transitions are complex, the visualization used here considers only mentioned transitions. All single transitions, as well as features that were not moved anywhere, are excluded from the graph. The results from visualizing this type of transitions on the industrial data are depicted in Figure 4.4. In our case, only five features happened to behave in this way. For all discovered cases the

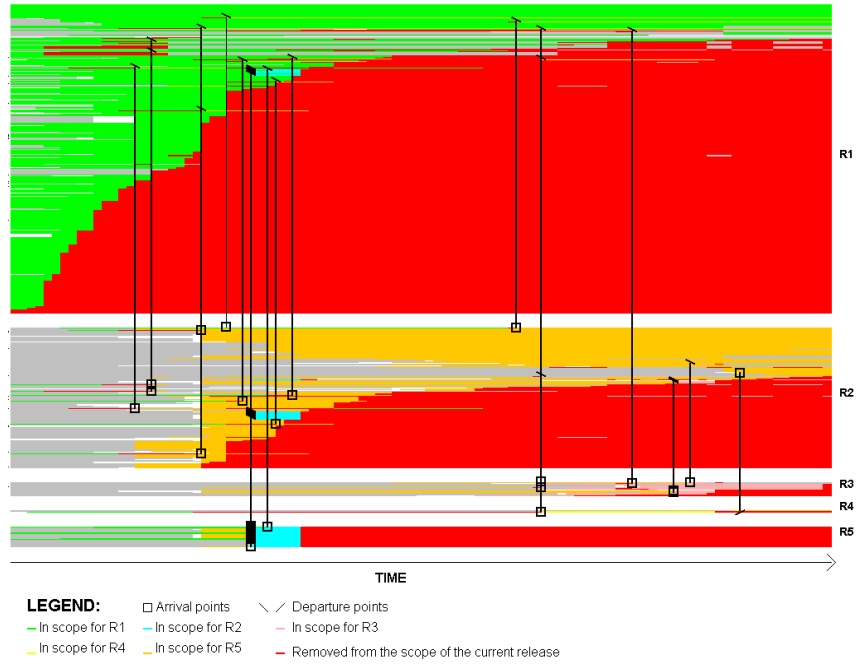

Figure 4.3: Visualizing within-project transitions for Project B (see Section 5.2). The full-size color figure can be found at http://www.cs.lth.se/home/Krzysztof_Wnuk/REV09/Figure3.bmp

scenario is the same, the features were first moved internally to an early project release within the same platform project, and then moved to the second release of the following project. To emphasize multiple transitions, a new symbol was added to the graph, namely the interim transition symbol. As a result of its design, this view cannot visualize the magnitude of multiple transitions compared with all transitions in the project. It is instead focusing on paths for multiple transitions.

7 Initial Validation

As an initial validation step, interviews were conducted with two practitioners from the case of the company under study, one person working with requirements engineering process improvement and one person working with scope management. The questions were asked to confirm or reject the assumptions that the researchers had before applying visualizations to the empirical data. As one of the first questions, interest in visualizing feature transitions was discussed. Both responders expressed

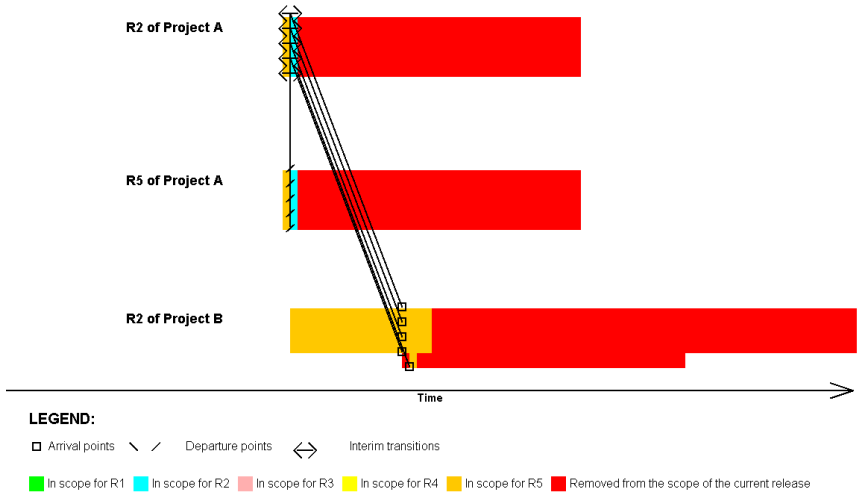

Figure 4.4: Multiple transitions between projects A and B visualized with the exclusion of non-transitions. The full-size color figure can be found at http://www.cs.lth.se/home/Krzysztof_Wnuk/REV09/Figure4.bmp

their interest in visualizing cross-project transitions and also reported that current tool support cannot provide this functionality. Neither of the responders could give an accurate estimate of the scale of this phenomenon in the case of the company under study, but they agreed that there are many changes that would be valuable to visualize and analyze.

Our responders confirmed our assumptions that feature transitions may sometimes heavily influence the market value of affected features. This is because for each feature there is an optimal market window for an estimated profit. If, for some reasons, a feature is delivered to the market outside its optimal market window new estimations of its market value need to be made. In addition, cost of implementation may be affected, although market value implications were considered more important. The relation to the cost of implementation is, according to one of our responders, dependent on when the feature was moved (to which project) since that may either reduce or increase the cost for implementation. Both responders expressed that it is crucial to visualize the transitions because of so called enablers: features that are prerequisites of other valuable features, but that might not have a great market value on their own. Enablers often have to be implemented before, or in conjunction with, the features that rely on them. Therefore, all backward feature transitions should be analyzed to

ensure that dependencies to required enablers related to moved features are available. In some cases, feature transitions may involve large architectural changes while the impact may be minimal in others. For forward transitions, enablers should not be rejected in order to make sure that support for the transferred features still persists. In the event of backwards transitions, it is important to check that support for the new functionality is available and thus may also require the backward transition of related enablers. Being able to trace features between the projects was considered as very valuable and desired.

Questions regarding usefulness and applicability of each type of the visualization were also asked. The external transitions graph was considered useful by our responders (by one responder the most useful graph). The meaning of the backward transitions was discussed together with the time delay between the exclusion and inclusion. As our responders mentioned, sometimes it is undesirable to remove the feature from the original scope until the final decision to transfer is made. For the backward transitions, the lead-time can be shown (Figure 4.1) representing the time needed to analyze the feature. The internal transition visualization turned out not to be as useful as the external version. Responders mentioned that the fact that each data point is placed twice on the graph (to distinguish among releases) may lead to wrong conclusions. It was also mentioned that in their company only one person is responsible for one project meaning that this person would usually be aware about the number of internal transitions in the project under his or her management.

Finally, the multiple transitions view was discussed. The responders found it less useful than the external transitions graph. One responder would like to have all features in the graph, not only the transitions, to be able to compare the scale of the project to the overall size of the project.

8 Conclusions

In this paper, we present a technique for visualization of the scope dynamics of changes within and across multiple projects called Feature Transition Charts (FTC), an extension of Feature Survival Charts (FSC) (Wnuk et al. 2008). We have applied FTC post-mortem to real-world data from two large projects. FTC was initially validated in dialogue with practitioners, indicating that while FTC may be both feasible and useful, additional research could enhance the features in terms of interactive zooming and enhanced user configurability. The main findings are summarized in relation to research questions from Section 1:

- (Q1) FTC can visualize scope changes across the projects by aligning a set of FSC and depicting transfers using special markers and lines. The visualization can scale to large projects (at least in the projects we have tested), which can be counted as its advantage over a tex-

tual representation of scope dynamics The practitioners believe that FTC can give a comprehensive overview of scoping dynamics that have not previously been made explicit, and that the concept of FSC (Wnuk et al. 2008; 2009) is extended in a useful way. FTC can be used by both requirements engineers and process managers to gain valuable information about the presence and nature of scope changes across projects or projects' releases.

- (Q2) The proposed visual symbols for departure and arrivals of feature transitions can be useful in providing an effective overview of the timing and magnitude of feature transitions. However, in a very large scale projects, many adjacent transitions can overlap and future work thus may include experiments with interactive zooming and filtering features.

Limitations. Our study has some limitations. Firstly, even if the case of the company under study 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. Secondly, our initial validation of FTC is limited to a static post-mortem analysis and because of that it could not be applied in a proactive manner and no feedback from ongoing projects could be gathered. Thirdly, when the size of the projects grows, our visualization technique should be complemented by zooming and interactive features so that the holistic picture can be perceived, while the details are available on demand.
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. Other means of marking the departure and arrival points should be evaluated. Finally, additional work should be performed to address the applicability of FTC in other contexts, for example other domains, such as information systems, and other development modes, such as single product development or agile development.

Acknowledgments

This work is supported by VINNOVA (Swedish Agency for Innovation Systems) within the UPITER project. Special acknowledgments to Thomas Olsson for valuable help with Section 7, and to Lars Nilsson for valuable language comments.

References

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

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

B. W. Boehm. Value-based software engineering. *Software Engineering Notes*, 28(2):1–12, 2003.

B.W. Boehm and L. G. Huang. Value-based software engineering: A case study. *Computer*, 36(3):33–41, 2003.

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

P. Carlshamre and B. Regnell. 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*, pages 961–965, 2000.

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

Å. Dahlstedt and A. Persson. Requirements interdependencies - moulding the state of research into a research agenda. In *Proceedings Ninth International Workshop on Requirements Engineering (REFSQ 2003)*, pages 71–80, 2003.

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

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

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

M. S. Feather, S. L. Cornford, J. D. Kiper, and T. Menzies. 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)*, pages 80–89, 2006.

O. C.Z. Gotel, F. T. Marchese, and S.J. Morris. On requirements visualization. In *Proceedings of the Second International Workshop on Requirements Engineering Visualization (REV 2007)*, pages 80–89, 2007.

G. De Gregorio. Enterprise-wide requirements and decision management. In *Proceeding of the 9th International Symposium of the International Council on System Engineering*, pages 1–7, 1999.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

UML. The unified modeling language webpage. `http://www.uml.org`, January 2010.

K. Wnuk, B. Regnell, and L. Karlsson. 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)*, pages 41–50, 2008.

K. Wnuk, B. Regnell, and L. Karlsson. 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)*, pages 89–98, 2009.

C. Wohlin and A. Aurum. 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)*, pages 246–255, 2005.

Paper V

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

Krzysztof Wnuk, Martin Höst, Björn Regnell,
Dept. of Computer Science, Lund University, Sweden
{krzysztof.wnuk,martin.host,bjorn.regnell}@cs.lth.se

To be submitted to the Empirical Software Engineering Journal (ESEJ)

ABSTRACT

Requirements management in large-scale contexts demands using effective methods that can cope with its inherently complex and challenging tasks. This paper presents a replicated experiment with a linguistic tool for consolidation of requirements sets. The core of the experiment is the requirements consolidation task, where new requirements are analyzed against those already present in a requirements database, and similarities are discovered and recorded. As a result, a significant effort can be saved by finding which new requirements are actually already implemented or analyzed. In this replication, 45 subjects used two methods for the consolidation task. The first method, also called the assisted method, uses linguistic engineering techniques to calculate the degree of similarity between requirements implemented in a tool called ReqSimile. The second method, namely the manual method, comprises searching and filtering capabilities provided by the Telelogic Doors tool. The manual method has been changed from the original experiment. The results of this replication follows the original experiment outcomes, where the linguistic method helped to access more correct links and miss fever links than the manual method. However, the performance improvement achieved for the linguistic method used in the original experiment is not confirmed by this replication, which may indicate that advanced searching and filtering functionalities may also be helpful in the requirements consolidation task.

1 Introduction

Requirements engineering in a market-driven context (Regnell and Brinkkemper 2005) can be characterized by: continuous elicitation, time-to-market constraints, and strong market competition (Natt och Dag 2006). In this context, requirements are arriving constantly 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 arriving to the company need to be analyzed both from the new market opportunity and the technical compliance perspectives. In a case when a company is large and develops complex software solutions to a global market, the quantity of information to constantly analyze and assess may severely impede the analytical capacity of requirements engineers and managers. The result is a need for methods and tools that will assist in analyzing large amounts of natural language requirements for the purpose of finding and recording similarities between them. As a result, the efficiency of requirement management activities can be significantly improved. The importance of the mentioned issue increases with the volumes of requirements to analyze.

The process of analyzing requirements incoming from customers or 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. The core in 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 amount 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. This problem was identified in the original experiment report as a relevant industrial problem in a large company, caused by complex stakeholders set ups and many projects running in parallel (Natt och Dag et al. 2006). 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 the method showed that its usage 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 fever requirements links. However, the unsupported method used in the original experiment was limited to a simple search functionality, while most of currently available requirements management tools offer more advanced filtering and

searching techniques. Thus, this replication has been designed in order to asses whether the tool, with a linguistic analysis of the similarity between requirements, can still perform better than a currently available commercial requirements management tool in the task of requirements consolidation. A replicated experiment has been chosen due to its falsifiable nature, which provides 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 were asked to consolidate two requirements sets by finding and linking requirements that address the same underlying functionality. The presented replication reuses the original procedures in terms of the study design and experimental steps (Natt och Dag et al. 2006), but uses another set of subjects and changes one of the experimental objects. In the light of the previous facts, this replication can be classified according to Shull et al. as an exact replication (Shull et al. 2008). The unchanged object in this replication, also called the *assisted* method, is a research prototype tool, called ReqSimile (Natt och Dag 2010), that utilizes linguistic analysis to assist in the task of finding similar requirements. The second object, being changed compared with the original experiment, is also called the *manual* method, and it utilizes searching and filtering functionalities implemented in a tool called Telelogic Doors (IBM 2010), which has currently 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. The objectives of the replication constitute the following research questions:

Q1: Can significant difference between the *assisted* and the *manual* methods that were achieved in the original experiment be confirmed in a replicated experiment where *manual* method is provided by a commercial requirements management tool?

Q1a: Is the *assisted* method significantly more efficient in consolidating two requirements sets?

Q1b: Is the *manual* method significantly more correct in consolidating two requirements test by assessing more correct links?

Q1c: Does the *assisted* help to miss less requirements links?

Q2: How do the results of the original study for each method correspond to the results for the same method in a replicated study?

Q2a: Is there any difference between the results for the *assisted* method between the two studies?

Q2b: Is there any difference between the results for the *manual* method between the two studies?

The aim of RQ1 is to assess if results obtained in the original experiment holds even if one of the tools is changed (Natt och Dag et al. 2006). Also, to better compare the results from the original and replicated experiments, the research question Q1 was divided into three sub-questions, were each of them is explicitly addressing various quality aspects of the consolidation process. Question Q2 aims at assessing if there are any differences between the two experiment runs for the same treatments. The possible differences provide valuable input regarding the nature of the consolidation task and the subjects used in both experiment runs.

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

2 Related Work

Replications play an important role in software engineering by allowing 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. A recent survey of controlled experiments in software engineering revealed that replications are still neglected by empirical researchers since only 18% of the surveyed experiments are reported as replications (Sjoberg et al. 2005). Moreover only 3,9% of analyzed controlled experiments can be categorized according to the IEEE taxonomy as requirements/specification related (Sjoberg 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 have to be guided by practitioners. The possibilities exemplified above 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 using ambiguity rates to sentences depending on the degree of syntactic and semantic uncertainty (Macias and Pulman 1995), or detecting ambiguities by applying an inspection technique

(E.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 is aiming for analyzing and improving, Fabbrini et al. (2001) proposed a tool that assess 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). Furthermore, Edwards et al. (1995) present a tool that uses rule-based parsing to translate requirements from natural language and by that helps with requirements analysis and maintenance tasks throughout the system life-cycle. 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 of natural language requirements which is considered as having low computational and human costs.

Apart from the quality evaluation and assurance tasks, NLP techniques have also been applied for a task of extracting abstractions from textual documents (Aguilera and D.Berry 1991, Goldin and Berry 1997) and helping synthesizing 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 in a way that is tolerant of both the volume and the quality of the text being analyzed. Rayson et al. (2000) reported experiences from one of the projects where probabilistic NLP techniques were used. They presented two experiments using tools that they have developed, namely part-of speech and semantic taggers, which suggested that the tools are effective in helping to identify and analyze domain abstractions. Their results were further supported by a later study by Sawyer et al. (Sawyer and Cosh 2004), where ontology charts of key entities were produced using collocation analysis. On the other hand, Gervasi et al. (1999) used lexical features of the requirements to cluster them according to specific criteria and 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.

A different angle of research in using NLP techniques for requirements engineering is presented by Luisa et al. (2004), who investigated the economical advantages of developing a CASE tool that integrates linguistic analysis techniques for documents written in lateral language. Even though the economic advantages of extending current requirements management tools by the NLP based features seems to be clear, not all tool vendors found these features interesting to have in their products. Moreover, the INCOSE report (INCOSE 2010) which provides a comprehensive compar-

ison of the functionality of requirements management tools, has not clearly defined a new class of features related to NLP techniques. However, among currently defined features are some that may be related to NLP techniques. One of these features, namely "the automatic parsing of requirements" is described as "a mechanism for automatic identification of requirements by keywords, structure, unique identifiers etc.". For this category, 54% of all analyzed tools have reported to have full support and 9% only partial support, while 36% reported no support at all. Another category that may be related to NLP techniques is entitled "Interactive/semi-automatic requirement identification" which is described as the ability to identify requirements from a text file via interactive means such as mouse highlighting of the requirement text or by letting the system to prompt "is this a requirement?". In this category, 59% of all analyzed tools have full support, 36% declared no support and 14% partial support. Another category is "identify inconsistencies", which states that the tool should allow the user to identify inconsistencies such as unlinked requirements or system elements (orphans). In this category, 24 tools have reported to have a full support (54%) 16 no support (36%) and 4 partial support (9%). Finally, the last category that may be related to NLP is the "quality and consistency checking" which is described in the report as "support for the document quality and consistency checking through spell checking, data dictionaries, acronym tables, etc.". In this case category, 22 of reported tools have full support (50%), 16 no support (36%) and 6 partial support (14%).

The fact that in this replication an open source tool is compared with a commercial tool, makes the open source research literature as a part of related work. The current state-of-art in the open source software research discipline provides a large amount of analyses and comparisons of open source software . On the other hand, Stol et al. (2009) analyzed four editions of the International Conference on Open Source Systems and only 28% of analyzed articles were classified as empirical studies. Furthermore, Stol et al. (2009) categorized analyzed papers in four categories: communities and the largest category, development and maintenance, diffusion and adoption as the second largest category and characteristics of OSS as the last category. Surprisingly, the category related to the analysis of the functionality that open source software is providing is not present in Stol et al.'s classification. Out of a small number of research papers that in general compares proprietary software to open source software, Machado et al. (2007), who compared user experience between the leading proprietary solution and open source solution of learning management systems. Others, for example Selvi et al. (2008), present a roadmap to analyze open source software and proprietary software using performance testing. As a result, a comparison of performance of proprietary versus non-proprietary software may be achieved. Performance is also the quality attribute of the software solutions compared by Patton et al. (2000), who compared firewall solutions based on the open source Linux operating system to the

commercial solution from Cisco using the Cisco IOS firewall features set.

3 Industrial Problem Description

In this section, we provide an industrial background for the problem of requirements consolidation on an example of a large software company that develops embedded systems to a global market. The importance of tackling the problem has been identified in the original experiment (Natt och Dag et al. 2006) and is also repeated here. Requirements are constantly arriving to the company's requirements database from multiple sources, namely requirements engineers, requirements management as well as key customers, and subcontractors. Another part of the company, which is responsible for providing new ideas, is called Application Planning, and is responsible for the different application areas of products under release. The primary customers are in this case selling the products to the end users. They are responsible for a large set of incoming requirements to the new products as well as for requests for specific product adaptations. In order to acquire knowledge in the technical capabilities of the company's products, key customers submit a Request for Information (RFI). The request for information process is depicted in Figure 5.1.

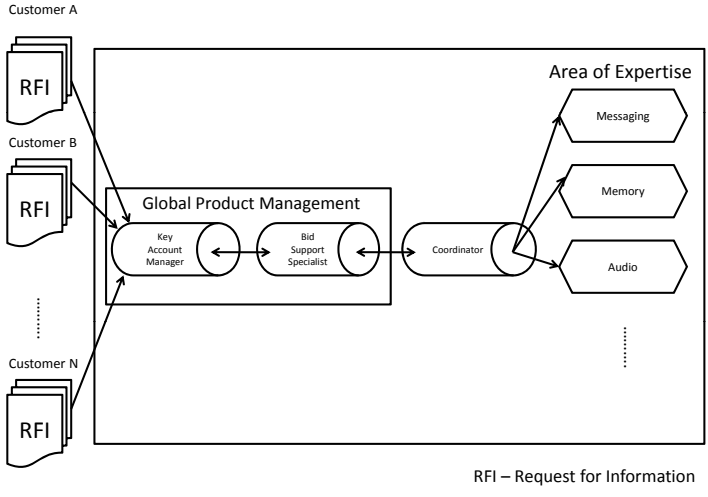

Figure 5.1: The request for information process.

Each year, each key customer submits a couple of RFIs. The RFIs arrive

to the Key Account Managers, one for each major customer, in different document formats (PDF, MS Excel, MS Word, etc.) and at different times. The main specification technique for the RFI requirements is feature style (Lauesen, 2002) using natural language as the specification language. The Key Account Manager passes the RFI on to a Bid Support Specialist, who reviews the RFI from a market point of view and decides which products shall be considered when dealing with the RFI. The Bid Support Specialist then passes the document on to the Coordinator, who analyzes the RFI and the accompanying instruction and distributes relevant parts of the RFI to different Areas of Expertise. An Area of Expertise consists of a Function Group and a Technical Work Group. The Technical Work Group focuses on road maps (i.e., future functions) and the Function Group is dedicated to implementation and testing. When the Areas of Expertise have stated the compliance to each requirement, they send the RFI reply back to the coordinator. The coordinator reviews the answers and sends the replies on to the Bid Support Specialist, who also checks the answers. If the RFI originates from a major customer, a meeting is held with the Global Product Management, the coordinator, and experts from the Areas of Expertise, in order to discuss the answers which are to be submitted back to the customer. The RFI reply is then sent back to the customer by the Key Account Manager. The RFIs play an important role in the customer's strategic planning.

The efficiency of the RFI process, in which requirements are analyzed and checked against product features, is however severely impeded. The experts that analyze the requests are often concerned with their primary assignments in development and testing and have troubles in finding the time required to analyze the RFIs. An even bigger issues is that the experts get frustrated as they have to state the compliance to the same or very similar requirements multiple times. Large parts of the new versions of RFIs arriving from the same customer are typically the same as previous versions. Furthermore, it is often the case that the same and very similar requirements appear in the RFIs from different customers. The mentioned problems can be generalized into other than requirements consolidation task, where large amounts of information have to be gathered and analyzed for multiple reasons. In this case, computer based analysis methods and tools can significantly decrease the time needed to analyze the mentioned amounts of information. Providing a tool support for automatic identification of similar requests is the topic of the original experiment and remains in the case of this replication.

4 Experimental Design

In this section, the outcome of the experiment planning phase is presented. The goal of this experiment is to assess which method: *assisted* or *manual* perform best in the task of requirements consolidation. The central part of

the requirements consolidation task is finding similarities between the two sets of requirements described in details in Section 3. The methods evaluated in this replication were implemented in two tools, namely ReqSimile and Telelogic Doors. 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 initial experiment design (Natt och Dag et al. 2006) where, when possible, the original material is reused and extended according to the guidelines of designing experiments presented in (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.

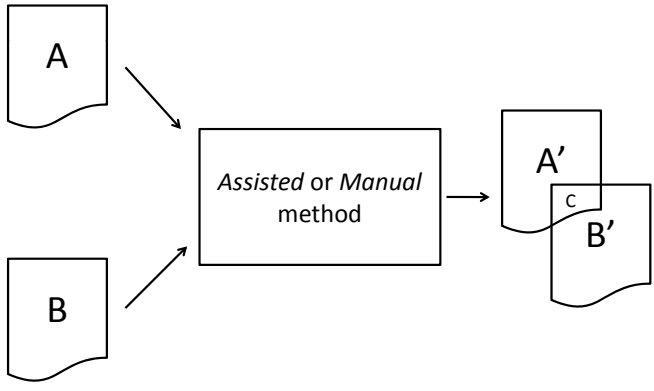

Figure 5.2: The process of using the support tool for requirements consolidation.

Due to the exact replication nature of this study (Shull et al. 2008), the evaluation of the experiment design for the replication sake was limited to checking additional changes and new elements. The changes concerned questionnaire improvements and a new instruction regarding the usage of Telelogic Doors (IBM 2010). The experiment design was evaluated by an independent researcher before executing the experiment. The same researcher participated in the pilot study where he used both tools to find similar requirements and create links between them. Comments and sug-

gestions regarding readability and understandability of the laboratory instructions were given and later implemented. Finally, since the requirements sets used in this replication were the same, the correct answer to the consolidation task remained unchanged.

Similarly to the original experiment, this replication was also conducted in a laboratory experiment, since it captures the consolidation problem in an untainted way. Figure 5.2 depicts the conceptual solution of the consolidation activity. To the left in Figure 5.2, two requirement sets A and B are shown. They represent two consecutive submissions of Requests For Information (RFIs) from the same key customer. We can also assume that the earlier RFI is set A in this case, and that it would have already been analyzed and the result from the analysis should be available in the central requirements database. The coordinator uses the support tool, which in the case of this replication can be either Telelogic Doors for the *manual* method (IBM 2010) or ReqSimile for the *assisted* method (Natt och Dag et al. 2006), to find requirements in the B set that were already analyzed in the A set and to mark them by assigning a link between them. The output of the process is shown to the right in Figure 5.2. The subset A' comprises all requirements that were previously analyzed but are no longer requested by the customer. 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 RFI that previously have been analyzed. The coordinator 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 subset C.

4.1 Goals, Hypothesis, Parameters and Variables

The variables in this replication 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 to the original experiment design. 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 does 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 does 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 does the *manual* method.

H_0^4 - The *assisted* method results in the same share of incorrectly linked requirements, N_{il}/N, as does 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 where $N_{cl} + N_{il} + N_{cu} + N_{iu} = \text{N}$.

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

4.2 Subjects

In this study, a different set of subjects comparing to the original experiment, but from the same kind of population was used. The sample includes participants of the course in Requirements Engineering at Lund University (ETS170) (Lund University 2010b). The course is an optional master-level course offered for students at several engineering programs including computer engineering and electrical engineering. It gives 7,5 ETCS points (ECTS 2010) which corresponds to five weeks full time study. 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 sub-

Table 5.1: The number of years of experience in software development in pairs of subjects. The remaining pairs of subjects exhibited no industrial experience for both pair members.

Pair of subjects	*Experience of the first subject in the pair (in years)*	*Experience of the second subject in the pair (in years)*
1	0.5	1.5
2	1	1
3	1	2
4	1	2
5	0.25	0
6	0.5	1
7	0	1
8	0.25	1.5
9	0.5	1

jects, 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 between four months and two years with an average value 11 months. At this stage, the most important measure to avoid biased pairs of subject was to ensure that persons with a lot of experience will not cooperate with persons with no experience and by that the whole process may be slowed down. The analysis of industrial experience in pairs of subjects revealed that nine pairs were having some degree of industrial experience which not always was equal. The remaining pairs had no experience at all. The analysis of the experience of both pair members and the difference in the experience between them is depicted in Table 5.1. The difference in experience varied between three months and 15 months with an average value of nine months. Therefore, we can assume that the difference in subject's professional experience had a minor impact on the results.

A similar analysis was performed for the question regarding experience of subjects from the course that the requirements used in this replication originate from (ETS032) (Lund University 2010a). The results revealed that 22 out of the 45 subject have not taken the course that the requirements originate from at all, 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 originates from in the pairs formed by subjects were analyzed. For most cases (16 out of 22), the pairs were formed by an inexperienced person and an experienced person from the course. This has 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. On the other hand, 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 six 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 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 subjects, three had experience from both courses and industry, and all were paired with persons with experience only from industry. Furthermore, three other subjects had only less than a year of experience from industry. Two of them were mixed with subjects having experience from both industry (also less than a year) and courses, while the third one was teamed up with a person having only experience from courses. Two other subjects reported no experience at all. Finally, one subject reported more than a year of industrial experience and the subject was teamed up with a person without any experience. Because of the fact that this significant difference occurs only for one couple it is reasonable to state that differences in experience of analyzing and reviewing requirements have only a minor impact on the results.

A further question concerned the subject's experience with the tool that implements the *manual* method, namely 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. Based on this, we can conclude that the subjects are very homogenous in this matter and that we can exclude this treat from aspects influencing the results.

4.3 Objects

The objects of this replication are methods used in supporting the requirements consolidation task. One of the methods, namely the *assisted* method, was implemented in the ReqSimile tool which uses linguistic engineering to calculate the degree of similarity between requirements using lexical similarity as a way of approximating semantic similarity (Natt och Dag et al. 2004). The other object, namely the *manual* method comprises searching and filtering functionalities provided by the Telelogic Doors tool. The goal of this replication 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 5.2. Comparing to the original experiment, one of the object was kept unchanged while the second one was changed. The change comprises substituting ReqSimileM from the original design (Natt och Dag

Table 5.2: The difference between the objects used in original and replicated experiment.

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

et al. 2006) by Telelogic Doors (IBM 2010). More information regarding tools used in the replication can be found in Section 4.3.2. The requirements specifications were kept unchanged comparing to the original experiment. Together with the reused requirements set the original tacit analysis information was used. The key with the correct answer was provided by an expert in the requirements domain and was created prior to any analysis of the subjects' assigned links in order to reduce any related validity threats (Natt och Dag et al. 2006).

4.3.1 Requirements

Two requirements sets were reused from the original experiment. The requirements specifications have been produced as a part of a course "Software Development of Large Systems" (ETS032) (Lund University 2010a). 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 written using natural language. Two sets of requirements that contained respectively 139 and 160 requirements were imported to ReqSmilieA and Telelogic Doors. As a result, each tool comprises the two remaining sets of requirements. An example of requirements from specification comprising 139 requirements is depicted in Table 5.3. More details about the requirements set can be found in the description of the original experiment (Natt och Dag et al. 2006).

4.3.2 Tools

In this replication, one tool remained unchanged from the original experiment while the other was changed. The tool that implements the *assisted* method, namely ReqSimile (Natt och Dag 2010), was kept unchanged. Re-

Table 5.3: Example requirements from a 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.

qSimile provides a linguistic tool support that aims for assisting in the requirements consolidation task. The user interface of ReqSimile is presented in Figure 5.3. The left side of the top pane of the window presents a list of requirements. 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 has 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). Before using linguistic support the user has to pre-process the requirements by selecting this option from the menu.

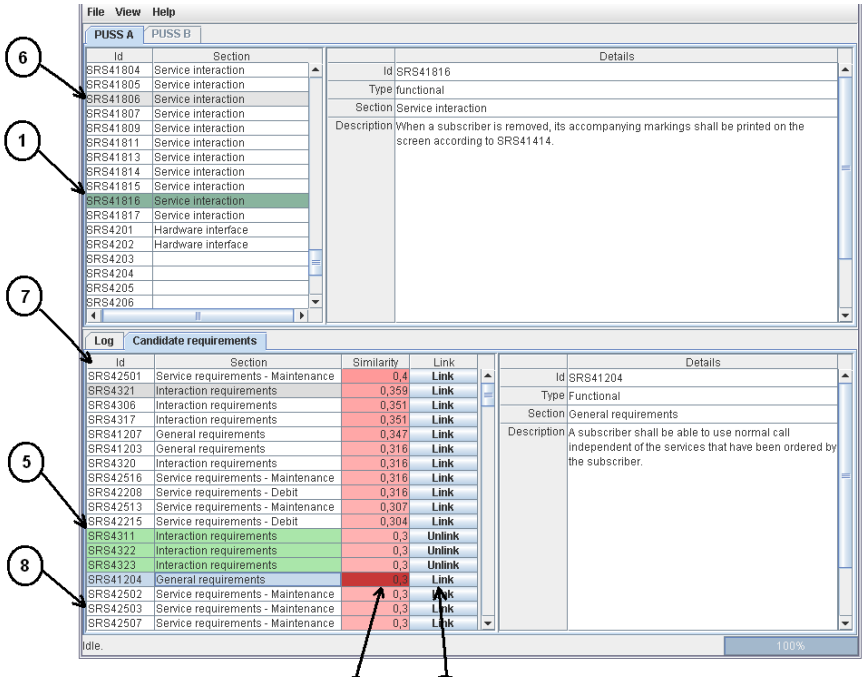

Figure 5.3: The user interface of ReqSmiliA used in the experiment.

The second tool, described by Natt och Dag et al. (2006) as ReqSimileM was changed in this replication to Telelogic Doors (IBM 2010). Telelogic Doors is one of the market leading commercial requirements management tools that, according to the vendor's information, provides powerful capabilities for capturing, linking analyzing and managing changes to requirements and their traceability. The tool has recently changed its vendor and

is currently called Rational DOORS. However, since in this experiment the Telelogic Doors version 8.3 was used, we will refer to it as Telelogic Doors. The user interface of Telelogic Doors is shown in Figure 5.4. The analyzed two sets or requirements were opened in Doors from separated modules and placed next to each other on the screen. Figure 5.4 comprises one of the requirements sets opened in a module. This orientation is similar to the ReqSimile's view and enables easy visual comparing between the two sets of requirements. To perform the consolidation task in Telelogic Doors, its finding and filtering capabilities were used. 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 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. This fact was addressed by the experiment's documentation.

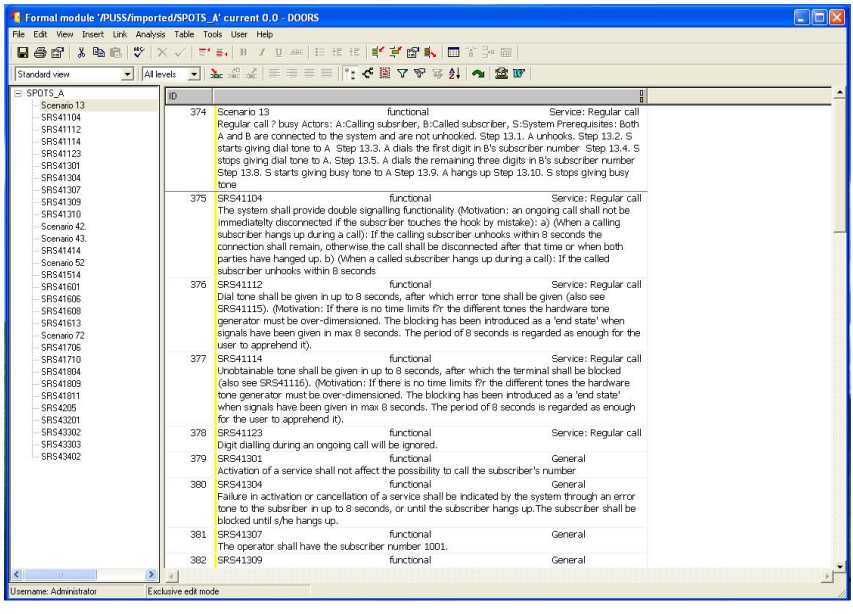

Figure 5.4: The user interface of Telelogic Doors used in the experiment.

4.3.3 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).

4.4 Instrumentation

In this replication, most of the original experiment's guidelines were kept unchanged. In particular, the instruction about how to use the *assisted* method (ReqSimile tool) was reused. A new instruction describing how to use the *manual* method (Telelogic Doors) to find similar requirements and assess links between them was developed and evaluated by an independent researcher. Since Telelogic Doors has a much more complex user interface, the instruction was significantly longer, consisting of four pages of text and figures. Due to its length, it was decided that subjects should get more time to read through the instruction. This fact was taken into consideration by adjusting the time needed to read the instruction for the groups working with Telelogic Doors during the experiment execution. The pre- and post-test questionnaires were updated according to the changes made from the original study. In the pre-test questionnaire, authors added one question about the experience in 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 the fact that the subjects of the experiment will only read requirements so their skills in writing are not as relevant. As a result, a questionnaire (pre-test) including five questions about the subjects' industrial experience in software development, experience from analyzing and revising requirements and possible knowledge and skills in Telelogic Doors was prepared. Before collecting the data, one independent researcher evaluated the questionnaire by checking 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 in the assignment. This difference from the original experiment design, where subjects were performing the task individually, 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.

4.5 Data Collection Procedure

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

4.6 Validity Evaluation

As for every experiment, the question about the validity of results has to be raised here. Threats to validity are presented and discussed using classification of threats to conclusion, internal, construct and external validity Wohlin et al. (Wohlin et al. 2000).

Conclusion validity. Due to the fact that the design of the replication kept the Null hypotheses unchanged from the original experiment, the same type of error, namely Type-II-error, may occur Wohlin et al. (2000). The probability of this type of error, may be expressed as:

P(type-II-error) = P (not rejecting H_0 | H_0 false)

In order to maximize the power of a statistical test in this case, which is defined as 1-P(type-II-error) the Type-II-error should be addressed. This threat is addressed by using normal probability plots to check that parametric tests can be used. This means that as powerful as possible test are used. Furthermore, one of the threats with respect to the subjects is, as in the original study, also limited since the subject groups are rather homogeneous. The subjects have attended the same education program for 2.5 years. On the other hand, the threat related to the fact that subjects were asked to work in pairs did not exist in the original design, and therefore has to be addressed here. This fact affects in particular the random heterogeneity of subjects, since created pairs may manifest differences in

industrial experience or experience from the previous courses. This threat is addressed by the analysis of the pre-study questionnaire results in Section 4.2. 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 question 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. This threat was addressed by performing a pre-test and a post-test questionnaires to help with assessing the mentioned factors and minimize dropouts. Although all subjects have taken the same education program for 2.5 years, the individual differences in industrial experience, experience from the course where requirements originates from, 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 the fact that the original measurements are reused also for the replication case. Finally, in order to minimize random irrelevance in experimental setting, the experiment moderators ensured that the subjects were not be disturbed during the task and that any discussions in pairs of subjects should be taken as silently as possible.

Internal validity. Similarly to the original experiment study, also in this study threats related to the history, maturation, morality etc. have to be mentioned. They are addressed by the fact that the experiment was run during a two hour period. The instrumentation threat is addressed in two ways: (1) by reusing the original experimentation instrumentation, if no changes were needed, and (2) by a review of the instrumentation documentation performed by an independent research. On the other hand, 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 Section 4.2 and in Section 7. The selection threat, similarly to the original design, may influence the results since the subjects are not volunteers and the laboratory session where experiment was performed is a mandatory part of the course. Finally, the incentives of participants is, next to their experience, an important factor that may influence the results of this study. According to the classification presented in Host 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 have typically 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 3 months recent industrial experience where recent means less than two years ago. Since all of the students were on

their third year of studies, we can assume that their working experience is more than two years in all cases. Although the identical comparison of two I2E1 cases in not present in Host et al. (2005), the example of two experiments classified as E1 I1 shows no significant difference in their outcomes. The E1 I1 level is defined as a combination of an isolated artifacts where subjects have no prior knowledge of the studies artifact and no experience of the subjects. Moreover, three other pairs of experiments, classified in the same category, also shows the same outcomes (Höst et al. 2005). The previous facts may lead to the conclusion that the incentive and experience influence on the results threat of this study can be addressed.

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 the speed or preparation to the experiment. Although in the dissimilarity with the original experiment design the experiment groups are not separated, no information about which method is actually better was revealed to the subjects. The possibility to look at other subjects' results during the experiment execution is minimized by the fact that the subjects took places in a way that separated two treatments by another treatment. Compensatory rivalry may be a problem in this case, since the group that will use an 'open-source' solution or the commercial solution may try to perform much 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 favor or assumingly better method. The potentially more problematic threat is the fact 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. This threat is, similarly to the original experiment design, addressed through the pretest where we asked about their ability to read and write common and technical English (Natt och Dag et al. 2006).

Construct validity. In this case, the theory is that the *assisted* method implemented in ReqSimile tool provides a better assistance to a particular task than the second method implemented in Telelogic Doors. In contrast to the original experiment design, none of the authors have developed any of the tools. On the other hand, the originally mentioned threat related to the awareness of subjects about their own errors is still present in the case of this replication. This 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 get more aware of the time spend and the performance results may be affected (Natt och Dag et al. 2006). Finally, the fact that exactly the same requirements sets were used for the replication may still not give the answer whether the results would be the same if the requirements set were altered. On the other hand, keeping the requirements sets unchanged opens up the possibility to discuss other factors and differ between the original and replicated experiment and address their influence on the results.

External validity. The largest threat in this category is the number of analyzed requirements. Since only a relatively small number of requirements is analyzed during the experiment, it is hard to generalize the results on a very big set of requirements, which often is the case in industry setting. 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. Also, 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. The time spent on the task is also among potential threats to external validity. In order to reduce the fatigue effect, the number of prioritized requirements was lower than in a real industrial setting, which is a serious threat in extending the result to the case with large amounts of 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 and the goal of the experiment was 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 biasing, no hypotheses were revealed and it was made very clear that it is not known which approach will perform better. This approach is similar to the original experiment execution described in (Natt och Dag et al. 2006). The only difference here is the fact that the experiment was performed in two occasions instead of at one occasion in parallel.

After the presentation, the subjects were assigned to the methods. Due to the fact that only one laboratory room could be used for one session, 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 of other indicators of the method on the laboratory desks when subjects took their seats in the laboratory room. Therefore, subject could not take a preferable method seat or be attracted by the name on the desk. Students were asked to discuss the solutions only within their own pair. Since the nearest group was not using the same object, the possibility of comparing or discussing results was avoided. The subjects were allowed to ask questions to the supervisor, if they experience any problems. Only answers related to the difficulties of using tools were given straightforward. 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 5.4 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. (2) A general task description. (3) Detailed tasks with space for noting down start and end times. (4) A short FAQ with general questions and answers about the requirements. (5) A screen shot of the tool user interface with the description of the different parts in the ReqSimile case or a four pages instruction with screen shots from the steps needed to analyzed requirements and make links using Telelogic Doors.

- The instruction to the students was as follows: (1) Walk through as many of the requirements as possible from the list of 30 requirements shown in the tool. For each investigated 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 post-test questionnaire described in Section 4.4. 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 a given task and, if applicable, propose improvements. Right after executing the experiment, it was known which data points had to be removed due to the used tool problems or subjects attitude. Three groups had problems with the tools used which resulted in loss of data and one group performed unacceptably slow analyzing only three requirements during 45 minutes and making only two links. All four groups were removed from the analysis, considered as outliers.

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 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 due to the fact that subjects formed pairs for the experiment in order to assess how this may influence their performance. Hypotheses were analyzed separately, while any relations and accumulated results are presented in Section 7. Similarly to the original experiment from which requirements were reused, also in this replication one analyzed requirement can be linked to several others. Therefore, any conclusions regarding functional relationships between the number of analyzed requirements and other measurements can not be drawn. The results from measuring dependent variables can be found in Table 5.4. 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 from the *manual* method). The dependent variables are described in Section 4.1.

Rows highlighted in Table 5.4 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 authors to remove subjects A8 from the analysis, since the links were not saved in the tool. Finally, group A7 was removed due to their lack of commitment to the task.

The time spent on the task is presented in column 2 of Table 5.4. 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.

The results for the number of analyzed requirements per minute are depicted as a box plot in Figure 5.5. It can be seen that there is no statistically significant difference in the number of analyzed requirements between 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 the 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.

The results for the number of correct links assigned by subjects are depicted in Figure 5.6. 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

Table 5.4: The results from measuring dependent variables.

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

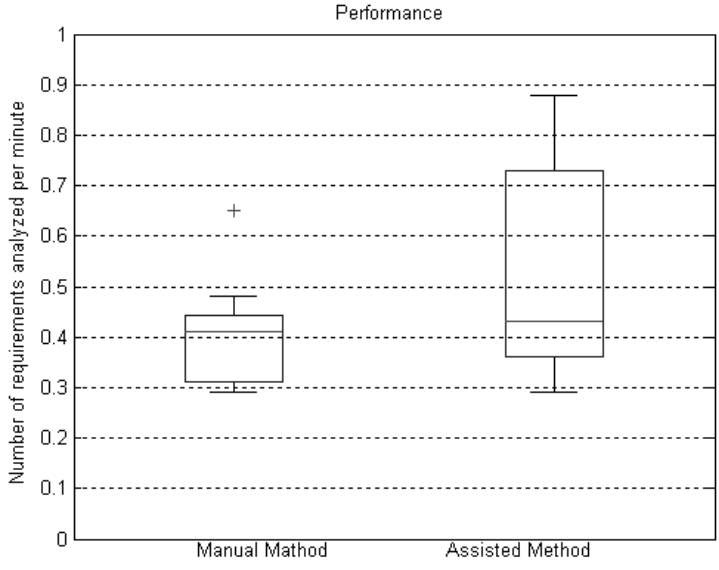

Figure 5.5: The results for the number of analyzed requirements.

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 is possible to reject hypothesis H_0^2.

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 was not, was calculated. In a 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 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 5.6. 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 orignal experiment's conclusions by making the conjecture that the *assisted* method helps the subjects to miss significantly fewer requirements links.

For the number of incorrectly assigned links (N_{il}), the t-test resulted in a p-value of 0.14, so the hypothesis H_0^4 can not be rejected. Furthermore, for the hypothesis H_0^5 the t-test gave the p-value 0.62 and for the hypoth-

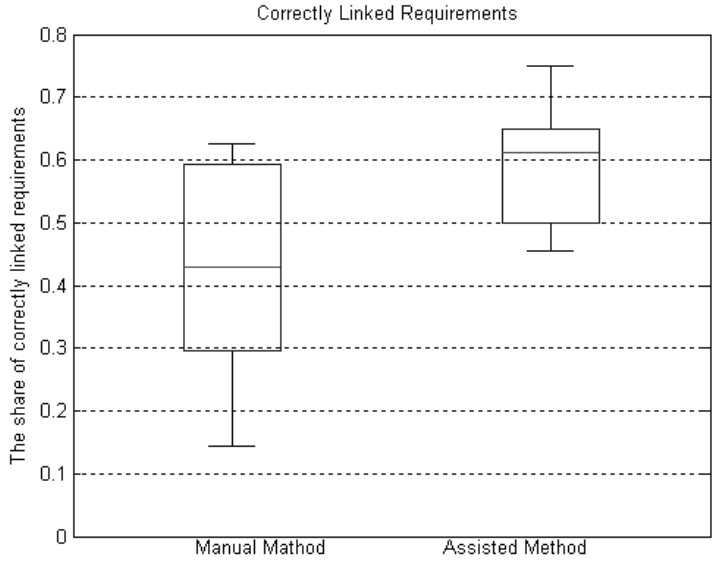

Figure 5.6: The results for correctly assigned links.

esis H_0^6 the t-test resulted in the p-value 0.72. Comparing to the original experiment, this replication confirmed 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 original and the replicated experiments is depicted in Table 5.5.

7 Experiment results interpretation

This section presents the interpretation of results. As already mentioned, this replication was conducted on a set of students. Therefore, 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). This section discusses the results of this replication, as well as compasses the results discusses to the replicated study.

7.1 Interpretation of this replication

The results achieved in this replication allow for the rejection of two out of six stated null hypotheses (see Table 5.5). As for the H_0^1 (performance) hypothesis, a lack of statistically significant difference can be explained

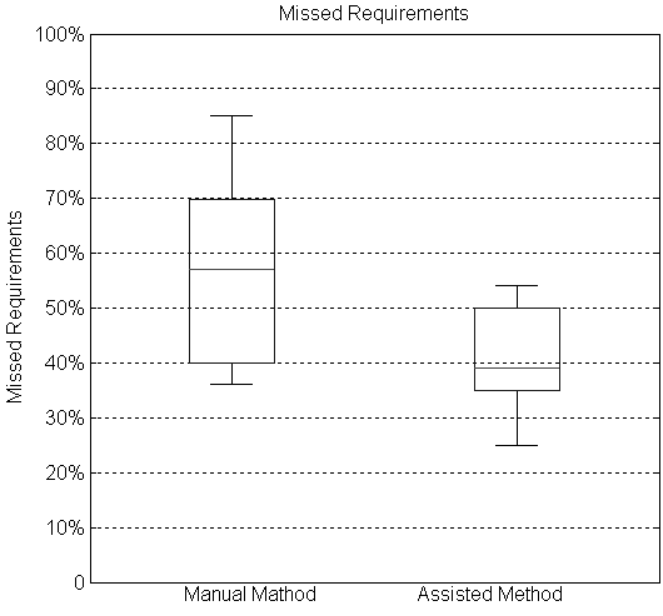

Figure 5.7: The results for the number of missed links.

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 for both groups there are similar numbers of more, as well as motivated subjects. In this case, the practical significance of the results is that the *assisted* method gives the possibility to achieve much higher values of the performance than the *manual* method. Assuming that the top scores for both methods correspond to the most motivated pairs of subjects, the *assisted* method can provide them a better support in increasing their performance with the given task. Furthermore, in real industrial situations, subjects are usually rather motivated in doing their task, since it is a part of their job, which supports our interpretation of the practical significance of received results. On the other hand, 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 exam-

Table 5.5: 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 Speed	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

ple 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 results concerning the number of correct links can be interpreted as follows. The group that used the *assisted* method assigned in average 58% of the correct links, while the group that used the *manual* method assigned in average 43% of the correct links. The results of the t-test allows to reject H_0^2 . This 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 a better support in 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) the fact that 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 confirms the results of the original experiment. The t-test confirms that the *assisted* method can help to miss fever requirements links than the *manual* method. Missing less 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 to the analyst which results in a decrease of the amount of

Table 5.6: The results of the t-tests for the original and the replicated experiments for the same methods.

Hypotheses	Assisted old/new (p-value)	Manual old/new (p-value)
H_0^1 Speed	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

missed links.

Similarly to the original experiment, the results from this replication can also not reject hypotheses H_0^4 , H_0^5 and H_0^6. The lack of statistically significant differences in these cases may be interpreted as a possible existence of additional factors that affect the consolidation of requirements process which were not controlled in this replication. For example, the fact that it is much easier to make a link in ReqSimile may affect the number of incorrect links, precision and accuracy.

7.2 Interpretation of the analysis of both cases

From the results of the t-tests between the same methods depicted in Table 5.6, we can see no significant difference for any of the cases. At the same time, other interesting differences between the experiments emerge. As it can be seen in Figure 5.8, the results for the speed of the *assisted* method in this replication has a much larger range of values, which may be the reason why the hypothesis H_0^1 could not be rejected. On the other hand, one of the possible explanations why the difference between the performance in this replication and in the original experiment may be a more advanced searching and filtering functionality that has been used in the *manual* method. Contrary to the original experiment, the *manual* method in this replication uses advanced searching and filtering functionalities which may to some extend be comparable with the linguistic similarity analysis because they also present only a subset of analyzed requirements. The analyst using the filtering and searching functionality has to provide a meaningful search string to filter out similar requirements, while in the linguistic similarity case the analysis is done automatically. Therefore, the filtering method has a higher degree of uncertainty which is shown by the results for the accuracy.

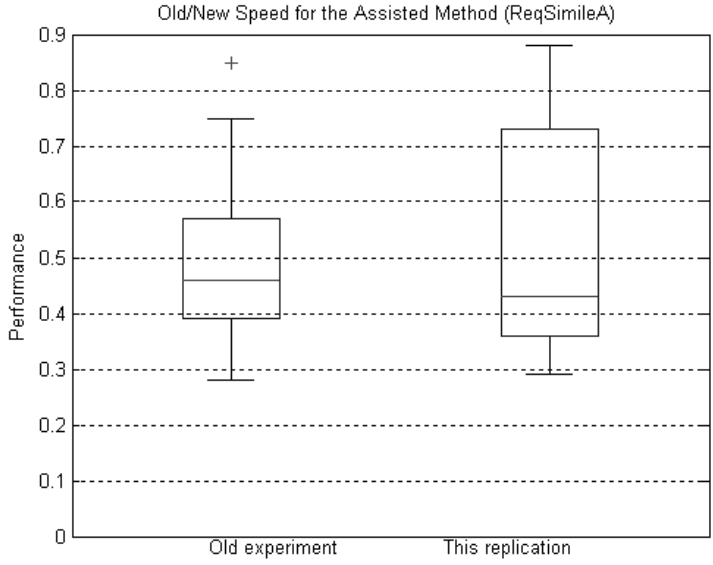

Figure 5.8: The result of comparing the speed for the *assisted* method in two experiment runs.

8 Conclusions

In this paper, we present a replicated experiment which aims to assess whether a linguistic method supports a specific large-scale requirements management activity, namely requirements consolidation, better than a searching and filtering method. In this replication, 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, subjects were assigning links between the requirements. The main conclusions of this paper are as follows:

- The *assisted* method does not seem to be more efficient in consolidating requirements than the *manual* method (question Q1a), which is a contradictory result compared to the original study

- The *assisted* method confirms to be significantly more correct in consolidating requirements than the *manual* method (question Q1b), which confirms the result from the original study

- The *assisted* method helps to miss fever requirements links than the *manual* method (Q1c), which confirms the result from 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) could also not be rejected by the result of this replication, which leaves this issues to be further investigated (Q2)

To summarize, the results from the original experiment corroborate five out of six hypotheses stated in the original experiment. 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 experiments runs.

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, and to Lars Nilsson for excellent language comments.

References

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

A. Begel and N. Nachiappan. Pair programming: What's in it for me? In *Proceedings of the 2008 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 120–128, 2008.

ECTS. The ects grading systems defined by european commission. `http://en.wikipedia.org/wiki/ECTS_grading_scale`, January 2010.

M. L. Edwards, M. Flanzer, M. Terry, and J. Landa. 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*, pages 278–281, 1995.

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

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

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

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

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

L. Goldin and D. M. Berry. Abstfinder, a prototype natural language text abstraction finder for use in requirements elicitation. *Automated Software Engineering*, pages 375–412, 1997.

M. Höst, C. Wohlin, and T. Thelin. Experimental context classification: Incentives and experience of subjects. In *Proceedings of the 27th International Conference on Software Engineering (ICSE 2005)*, pages 470–478, 2005.

H. Hulkko and P. Abrahamsson. A multiple case study on the impact of pair programming on product quality. In *Proceedings - 27th International Conference on Software Engineering (ICSE 2005)*, pages 495–504, 2005.

IBM. Rational doors (former telelogic doors) product description. http://www-01.ibm.com/software/awdtools/doors/productline/, January 2010.

IEEE. The ieee keyword taxonomy webpage. http://www.computer.org/mc/keywords/software.htm, January 2010.

INCOSE. The incose requirements management tool survey website. http://www.incose.org/ProductsPubs/products/rmsurvey.aspx, January 2010.

B. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, E. Emam, and J. Rosenberg. Preliminary guidelines for empirical research in software engineering. *IEEE Transactions on Software Engineering*, 28(8):721–734, 2002.

S. Lauesen. *Software Requirements – Styles and Techniques*. Addison–Wesley, 2002.

M. Luisa, F. Mariangela, and I. Pierlugi. Market research for requirements analysis using linguistic tools. *Requirements Engineering Journal*, 9(1):40–56, 2004.

ETS032 Lund University. The software development of large systems course page (ets032) at lund university. http://www.cs.lth.se/ETS032/, January 2010a.

ETS170 Lund University. The requirements engineering course (ets170) at the lund university. http://www.cs.lth.se/ETS170/, January 2010b.

M. Machado and E. Tao. Blackboard vs. moodle: Comparing user experience of learning management systems. In *Proceedings of the Frontier Education Conference (FIE 2007)*, pages 7–12, 2007.

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

L. Mich, J. Mylopoulos, and Z. Nicola. Improving the quality of conceptual models with nlp tools: An experiment. Technical report, University of Trento, 2002.

J. Natt och Dag. *Managing Natural Language Requirements in Large-Scale Software Development*. PhD thesis, Lund University, Sweden, 2006.

J. Natt och Dag. The reqsimile tool website. http://reqsimile.sourceforge.net/, January 2010.

J. Natt och Dag, V. Gervasi, S. Brinkkemper, and B. Regnell. 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)*, pages 283–294, 2004.

J. Natt och Dag, T. Thelin, and B. Regnell. 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, 2006.

A. Parrish, R. Smith, D. Hale, and J. Hale. A field study of developer pairs: Productivity impacts and implications. *IEEE Software*, 21(5):76–79, 2004.

S. Patton, D. Doss, and W. Yurcik. Open source versus commercial firewalls: functional comparison. In *In Proceedings of the 25th Annual IEEE International Conference on Local Computer Networks (LCN 2000)*, pages 223–224, 2000.

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

P. Rayson, L. Emmet, R. Garside, and P. Sawyer. 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*, pages 288–300, 2000.

B. Regnell and S. Brinkkemper. *Engineering and Managing Software Requirements*, chapter Market–Driven Requirements Engineering for Software Products, pages 287–308. Springer, 2005.

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

C. Rolland and C. Proix. A natural language approach for requirements engineering, 1992.

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

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

P. Sawyer and K. Cosh. Supporting measur-driven analysis using nlp tools. In *Proceedings of the 10th International Workshop on Requirements Engineering: Foundations of Software Quality (REFSQ 2004)*, pages 137–142, 2004.

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

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

R. T. Selvi, N. Sudha, and V. Balasubramanian. Performance analysis of proprietary and non-proprietary software. In *Proceedings of the International MultiConference of Engineers and Computer Scientists 2008 Vol I (IMECS 2008)*, pages 982–984, 2008.

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

D. I. K. Sjoberg, J. E. Hannay, O. Hansen, V.B. Karahasanovic, A. Liborg, and N. K. Rekdal. The survey of controlled experiments in software engineering. *IEEE Transactions on Software Engineering*, 31(9):733–753, 2005.

K. J. Stol, M. A. Babar, B. Russon, and B. Fitzgerald. The use of empirical methods in open source software research: Facts, trends and future directions. In *In Proceedings of the 2009 ICSE Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development*, pages 19–24, 2009.

L. Wilkinson. Statistical methods in psychology journals: Guidelines and explanations. *American Psychologist*, 54(8):594–604, 1999.

W.M. Wilson, L. H. Rosenberg, and L.E. Hyatt. Automated analysis of requirement specifications. In *Proceedings of the 1997 19th International Conference on Software Engineering (ICSE 97)*, pages 161–171, 1997.

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